A Dichotomy Hierarchy for Linear Time Subgraph Counting in Bounded Degeneracy Graphs*

Daniel Paul-Pena† C. Seshadhri†

Abstract

Subgraph and homomorphism counting are fundamental algorithmic problems. Given a constant-sized pattern graph H and a large input graph G, we wish to count the number of H-homomorphisms/subgraphs in G. Given the massive sizes of real-world graphs and the practical importance of counting problems, we focus on when (near) linear time algorithms are possible. The seminal work of Chiba-Nishizeki (SICOMP 1985) shows that for bounded degeneracy graphs G, clique and 4-cycle counting can be done in linear time. Recent works (Bera et al, SODA 2021, JACM 2022) show a dichotomy theorem characterizing the patterns H for which H-homomorphism counting is possible in linear time, for bounded degeneracy inputs G. At the other end, Nešetřil and Ossona de Mendez used their deep theory of "sparsity" to define bounded expansion graphs (which contains all minor-closed families). They prove that, for *all* H, H-homomorphism counting can be done in linear time for bounded expansion inputs. What lies between? For a specific H, can we characterize input classes where H-homomorphism counting is possible in linear time?

We discover a hierarchy of dichotomy theorems that answer the above questions. We show the existence of an infinite sequence of graph classes $\mathcal{G}_0 \supseteq \mathcal{G}_1 \supseteq \ldots \supseteq \mathcal{G}_\infty$ where $\mathcal{G}_0$ is the class of bounded degeneracy graphs, and $\mathcal{G}_\infty$ is the class of bounded expansion graphs. Fix any constant sized pattern graph H. Let $LICL(H)$ denote the length of the longest induced cycle in H. We prove the following. If $LICL(H) < 3(r+2)$, then H-homomorphisms can be counted in linear time for inputs in $\mathcal{G}_r$. If $LICL(H) \geq 3(r+2)$, then (assuming fine-grained complexity conjectures) H-homomorphism counting on inputs from $\mathcal{G}_r$ takes $\Omega(m^{1+\gamma})$ time. (Here, m denotes the number of input edges, and γ is some explicit constant.) Similar dichotomy theorems hold for subgraph counting.

1 Introduction

Counting the number of small patterns in a large input graph is a central algorithmic technique and widely used in both theory and practice [43, 19, 34, 22, 44, 2, 21, 54, 57, 55]. We express this problem as *homomorphism* or *subgraph* counting. The pattern is a simple constant sized graph $H = (V_H, E_H)$. The input simple graph is denoted by $G = (V_G, E_G)$. An H-homomorphism is a map $f : V_H \to V_G$ that preserves edges. So, $\forall (u, v) \in E_H$, $(f(u), f(v)) \in E_G$. If f is an injection (so distinct vertices of H are mapped to distinct vertices of G), this map is a subgraph. We use $\mathrm{Hom}_H(G)$ (resp. $\mathrm{Sub}_H(G)$) to denote the count of the distinct H-homomorphisms (resp. H-subgraphs).

Homomorphism and subgraph counting have applications in logic, graph theory, partition functions in statistical physics, database theory, and network science [18, 17, 26, 13, 54, 23, 52]. The topic of computing $\mathrm{Hom}_H(G)$ is itself a subfield of graph algorithms [37, 4, 17, 26, 24, 22, 13, 21, 14, 55]. The simplest non-trivial case is when H is a triangle, which has itself led to numerous papers.

In the case of subgraph counting, when H is part of the input size, the problem is exactly counting subgraph isomorphisms, which is $\mathbb{NP}$-hard. In many applications, the pattern is small and fixed. Let $n = |V_G|$ and $k = |V_H|$. When H is a k-clique, the problem of computing $\mathrm{Hom}_H(G)$ is $\#W[1]$-hard when parameterized by k [22]. So we do not expect $f(k) \cdot n^{O(1)}$ algorithms (for any function f). Nonetheless, the trivial bound of n^k can be beaten for specific H. The breakthrough result of Curticapean-Dell-Marx proved that if H has treewidth at most 2, then $\mathrm{Hom}_H(G)$ can be computed in $\mathrm{poly}(k) \cdot n^\omega$ time, where ω is the matrix multiplication constant [21]. Their result also showed that algorithms and lower bounds easily translate between homomorphism counting to subgraph counting. In the following discussion, we only refer to $\mathrm{Hom}_H(G)$. But all our questions and answers apply to $\mathrm{Sub}_H(G)$ with suitable modifications.

Homomorphism and subgraph counting have wide applications in network science, and there is a large study of practical algorithms for this problem (refer to tutorial [57]). For massive inputs like sparse real-world graphs,

*The full version of the paper can be accessed at https://arxiv.org/abs/2311.09584

†University of California, Santa Cruz. United States. Authors supported by NSF CCF-1740850, DMS-2023495, and CCF-1839317.

(near) linear time is likely a better mathematical abstraction for feasibility, than just polynomial time. A starting point for a theoretical investigation of linear time homomorphism counting on sparse graphs is a seminal result of Chiba-Nishizeki [19] that focuses on *graph degeneracy*. An input graph G has bounded degeneracy, if all subgraphs of G have bounded average degree. Chiba-Nishizeki proved that clique counting and 4-cycle counting can be done in linear time for bounded degeneracy graphs. The degeneracy has a special significance in the analysis of real-world graphs, since it is intimately tied to the technique of "core decompositions" [56]. The family of bounded degeneracy graphs is quite rich, and includes all minor-closed families, bounded treewidth classes, and preferential attachment graphs. Most real-world graphs tend to have small degeneracy ([35, 38, 58, 8, 12], also Table 2 in [8]), underscoring the practical importance of this class. Moreover, the best exact graph pattern counters are based on these algorithmic techniques [54, 51, 38, 52, 56].

For a deeper theoretical understanding, we are motivated by the following question.

Under what conditions on (bounded degeneracy) G and H, can $\mathrm{Hom}_H(G)$ be computed in near-linear time?

A series of recent subgraph counting advances provide dichotomy theorems characterizing the patterns H for which $\mathrm{Hom}_H(G)$ can be computed in linear time, when G has bounded degeneracy [14, 10, 11, 9]. Assuming fine-grained complexity conjectures, linear time algorithms exist iff the longest induced cycle of H is strictly less than 6. This is a surprisingly precise characterization, even though the final linear time algorithm is quite intricate.

At the "other end", early work by Eppstein showed that, for all fixed H, determining the existence of an H-homomorphsim is linear-time computable if G is planar [31]. These results were extended to bounded genus graphs [32]. In a grand generalization of these results, Nešetřil and Ossona de Mendez established the concept of *bounded expansion* graph classes [48]. These classes are defined using the theory of shallow minors. Bounded expansion classes are quite broad, and include all bounded degree graphs, include bounded tree-width graphs, and all minor-closed families. Bounded expansion graphs form a strict subset of bounded degeneracy graphs and are also nowhere dense [50]. They proved that for all fixed H, if G has bounded expansion, then one can count H-homomorphism/subgraphs in linear time (refer to Table 18.1 and Section 18.6 of [50] and [49]).

To summarize the above discussion, we have two ends of a spectrum. Assume some fine-grained complexity conjectures on triangle counting. Suppose G has bounded degeneracy. Then $\mathrm{Hom}_H(G)$ is linear-time computable iff the longest induced cycle of H is strictly less than 6. On the other hand, if G has bounded expansion, then for all H, $\mathrm{Hom}_H(G)$ can be computed in linear time.

What lies in between? Is there some class of graphs between bounded degeneracy and bounded expansion graphs where, say, 9-cycle homomorphisms can be counted in linear time?

1.1 Main Result We give an interpolation between the bounded degeneracy results of [11, 9] and bounded expansion results of [48]. There is an infinite hierarchy of classes between bounded degeneracy and bounded expansion graph classes. For any pattern H, we can precisely point out the largest class of the hierarchy where $\mathrm{Hom}_H(G)$ is linear-time computable.

These graph classes are defined using a concept called the *r rank greatest reduced average degree* (or r-grad) of a graph [50]. The definition is technical and explained in the next section. For any $r \in \mathbb{Z}^+$, the quantity $\nabla_{r/2}(G)$ denotes the $r/2$-grad of G. This is a well-defined graph quantity. Also, $\nabla_0(G)$ is the maximum average degree of any subgraph of G, which, up to constant factors, is the graph degeneracy (or arboricity) (Theorem 4 in [56]). Moreover, for any $r < s$, $\nabla_{r/2}(G) \leq \nabla_{s/2}(G)$.

To give our main lower bound, we use the following common conjecture from fine-grained complexity, the Triangle Detection Conjecture. The constant γ in the conjecture is believe to be $1/3$.

Conjecture 1.1. *(Triangle Detection Conjecture [1]) There exists a constant $\gamma > 0$ such that in the word RAM model of $O(\log n)$ bits, any algorithm to detect whether an input graph on m edges has a triangle requires $\Omega(m^{1+\gamma-o(1)})$ time in expectation.*

Consider a class of input graphs with bounded $\nabla_{r/2}$. We use $LICL(H)$ to denote the length of the longest induced cycle in H. Our main result is the following. If $LICL(H) < 3(r+2)$, then $\mathrm{Hom}_H(G)$ can be counted in linear time. If $LICL(H) \geq 3(r+2)$, then, assuming the Triangle Detection Conjecture, any algorithm counting $\mathrm{Hom}_H(G)$ for graphs with bounded $\nabla_{r/2}$ requires $\Omega(m^{1+\gamma})$ time. (Here, m refers to the number of edges in G.)

Theorem 1.1. (Main Theorem) *Fix any $r \in \mathbb{Z}^+$ and let H be a pattern graph. Let $\nabla_{r/2}(G)$ denote the $r/2$-grad and m the number of edges of the input graph G. Let γ be the constant from the Triangle Detection Conjecture.*

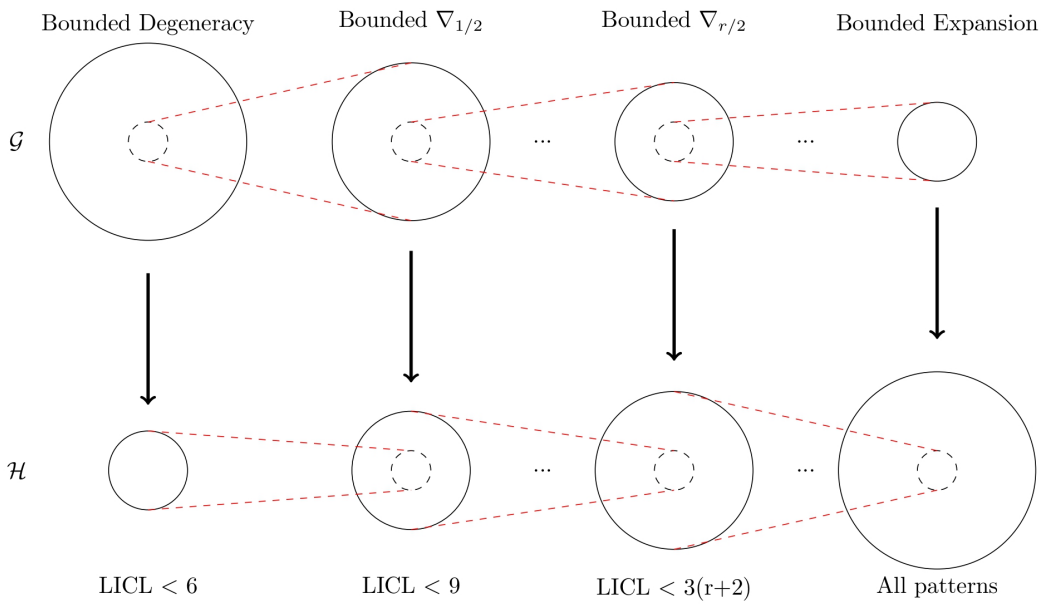

Figure 1: A visualization of our main result. There is a decreasing hierarchy of input graph classes between bounded degeneracy and bounded expansion. There is a corresponding increasing hierarchy of pattern classes, based on the $LICL$. As we look at more restrictive graph classes, we can count homomorphisms of more patterns in linear time.

- *If* $LICL(H) < 3(r+2)$*: there exists an explicit function* $f : \mathbb{N} \to \mathbb{N}$ *such that there is an algorithm that computes* $\mathrm{Hom}_H(G)$ *for all graphs* G *in time* $f(\nabla_{r/2}(G)) \cdot m$*.*
- *If* $LICL(H) \geq 3(r+2)$*: Assume the Triangle Detection Conjecture. For any function* $g : \mathbb{N} \to \mathbb{N}$ *and any* $\varepsilon < \gamma$*, there is no algorithm that computes* $\mathrm{Hom}_H(G)$ *for all graphs* G *in time* $g(\nabla_{r/2}(G)) \cdot m^{1+\varepsilon}$*.*

Remark 1: The algorithm above is randomized, but the only use of randomness is in building hash tables for a polynomial sized universe. Replacing the hash tables with van Emde Boas trees, we can get a deterministic algorithm running in time $O(m \log\log m)$. Many of the techniques used by the algorithm are generalizations/formalizations of heuristics used in practice [54, 52].

Remark 2: One can also express the previous theorem in terms of graph classes. Let r be a positive integer and let $\mathcal{G}_r$ be the family of all graph classes of bounded $r/2$-grad.

- If $LICL(H) < 3(r+2)$: then for each class $\mathcal{C}$ in $\mathcal{G}_r$, the problem of counting H-homomorphisms for $G \in \mathcal{C}$ can be solved in linear time.
- If $LICL(H) \geq 3(r+2)$, then there is a class $\mathcal{C}$ in $\mathcal{G}_r$ such that no linear time algorithm exists for the problem of counting H-homomorphisms for $G \in \mathcal{C}$, assuming the Triangle Detection Conjecture.

The hierarchy for linear-time counting. The theorem above can be informally visualized as Fig. 1. Consider an infinite hierarchy of nested graph classes[1] $\mathcal{G}_0 \supseteq \mathcal{G}_1 \supseteq \mathcal{G}_2 \ldots \supseteq \mathcal{G}_\infty$, where $\mathcal{G}_\infty = \bigcap_{r \in \mathbb{Z}^+} \mathcal{G}_r$. The class of bounded degeneracy graphs is $\mathcal{G}_0$ and the class of bounded expansion graphs is $\mathcal{G}_\infty$. (Recall that even $\mathcal{G}_\infty$ contains all minor-closed families; so it is really a big graph class by itself.) Formally, $\mathcal{G}_r$ is the class of graphs where $\nabla_{r/2}$ is bounded.

Now consider an "opposite" hierarchy of pattern classes $\mathcal{H}_0 \subseteq \mathcal{H}_1 \subseteq \mathcal{H}_2 \ldots \subseteq \mathcal{H}_\infty$, where $\mathcal{H}_\infty$ is the set of all patterns. For every $r \in \mathbb{Z}^+ \cup \{\infty\}$, for all patterns $H \in \mathcal{H}_r$, there is a linear time algorithm computing $\mathrm{Hom}_H(G)$ where $G \in \mathcal{G}_r$. Moreover, for all $H \notin \mathcal{H}_r$, one requires $m^{1+\gamma}$ time to compute $\mathrm{Hom}_H(G)$ for $G \in \mathcal{G}_r$.

[1] Technically, these would be families of graph classes. For example $\mathcal{G}_1$ will be the family of graph classes with bounded $\nabla_{1/2}$. We we will refer as then simply as classes.

Specifically, for $r = 0$, $\mathcal{G}_0$ is the class of bounded degeneracy graphs, and $\mathcal{H}_0$ is the set of patterns with $LICL(H) < 6$.

The obstacle of long induced cycles. Theorem 1.1 implies that long induced cycles are the obstruction towards efficient (near-linear) algorithms. There is a curious jump of 3 for the $LICL$ at every "level" of this hierarchy. While this may appear to be some artifact of the algorithm, this jump is matched by the hardness results of Theorem 1.1. We find it quite striking that the multiples of 3 are exactly the transition points for the hardness of homomorphism counting. The graph classes $\mathcal{G}_r$ are defined by the $r/2$-grad values, which seem to have no connection to these multiple of 3 transition points.

The dichotomies for subgraph counting. Subgraph counts can be easily represented as linear combinations of homomorphism counts, using inclusion-exclusion. Hence, algorithms for the latter can be used for subgraph counting. To count H-subgraphs, we count homomorphisms of all patterns formed by specific mergings of H. Remarkably, a result of Curticapean-Dell-Marx showed that this procedure is actually optimal [21]. Meaning, *lower bounds* for homomorphism counting translate to subgraph counting exactly as the upper bounds go. Using their techniques, we can adapt Theorem 1.1 to subgraph counting dichotomies.

For a pattern H, the $Spasm(H)$ is the set of all graphs obtained by taking any partition of V_H into independent sets and contracting each independent set in the partition into a single vertex. (These are the graphs that H has an injective homomorphism to. Note that an H-homomorphism may map an independent set to the same vertex of G.) Abusing notation, let $LICL(Spasm(H))$ denote the largest $LICL$ value among all patterns in $Spasm(H)$. To get our hierarchical dichotomies for subgraph counting, we simply replace $LICL(H)$ in Theorem 1.1 by the larger quantity $LICL(Spasm(H))$.

THEOREM 1.2. (DICHOTOMIES FOR SUBGRAPH COUNTING) *Fix any $r \in \mathbb{Z}^+$ and let H be a pattern graph. Let $\nabla_{r/2}(G)$ denote the $r/2$-grad and m the number of edges of the input graph G. Let γ be the constant from the Triangle Detection Conjecture.*

- *If $LICL(Spasm(H)) < 3(r+2)$, there exists an algorithm that computes $\mathrm{Sub}_H(G)$ for all graphs G in time $f(\nabla_{r/2}(G)) \cdot m$ for some explicit function $f : \mathbb{N} \to \mathbb{N}$.*
- *If $LICL(Spasm(H)) \geq 3(r+2)$: Assume the Triangle Detection Conjecture. For any function $g : \mathbb{N} \to \mathbb{N}$ and any $\varepsilon < \gamma$, there is no algorithm that computes $\mathrm{Sub}_H(G)$ for all graphs G in time $g(\nabla_{r/2}(G)) \cdot m^{1+\varepsilon}$.*

1.2 Shallow Minors and Greatest Reduced Average Density To explain what $\nabla_{r/2}$ means, we introduce the fundamental concept of *shallow minors*. Recall that a minor of G is a graph F formed as follows. Each vertex of F represents a connected subgraph of G. All of these connected graphs are vertex disjoint. An edge in F represents an edge in G connecting the corresponding subgraphs. (Usually, a minor is described in terms of deletions and contractions. The connected subgraphs described above are contracted to the single vertices of F.)

In a shallow minor at depth d, the connected subgraphs have radius at most d. This section is taken from Sections 4.2 and 4.4 of [50].

DEFINITION 1.1. (SHALLOW MINOR) *The graph G' is a* shallow minor of G at depth d *if there exists a collection of disjoint subsets $V_1, \ldots, V_p$ of vertices in G such that:*

- *Each graph induced by V_i has radius at most d: in set V_i, there is a vertex x_i such that every vertex in V_i is at distance at most d from x_i in the graph induced by V_i. This x_i is called the* center *of V_i.*
- *G' is a subgraph of the graph G with $\mathcal{P} = \{V_1, \ldots, V_p\}$ contracted: each vertex v of G' corresponds to a set $V_{i(v)}$, and edge (u, v) in G' corresponds to two sets $V_{i(u)}$ and $V_{i(v)}$ linked by at least one edge.*

We use $G' \in G \triangledown d$ to denote that G' is a shallow minor of G at depth d.

We can also define shallow minor at half-integer depths. Suppose $G' \in G \triangledown d$. There is a subgraph of G that is a *witness*, which essentially contains the subgraphs of radius d induced by the V_i's (corresponding to vertices of G') connected by certain edges (corresponding to the edges of G'). The latter edges are called *external* edges of the witness. The graph induced by each V_i is called a *bush*.

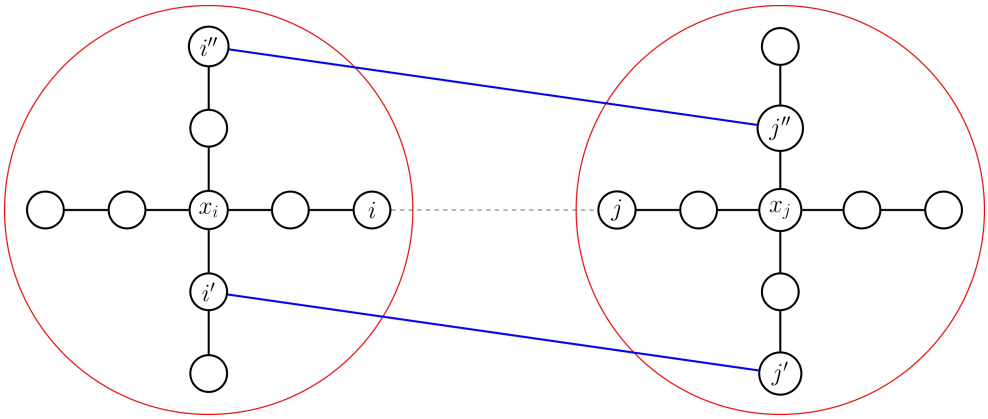

Figure 2: An example of a shallow minor of depth 3/2. Each of the red circles correspond to one of the two *bushes* forming the minor, centered at x_i and x_j respectively. the blue edges represent the *external* edges. We can see how at least one of the endpoints in each blue edge is not at a distance of 2, satisfying the condition of Definition 1.2. The dashed edge (connecting i to j) would not satisfy the condition as both endpoints are at a distance of 2 from their respective centers, hence adding it would result in a shallow minor of depth 2.

DEFINITION 1.2. (HALF-DEPTH SHALLOW MINOR) *A minor $G' \in G \triangledown d$ is said to have* depth $d-(1/2)$ *if the following holds. There exists a subgraph of G witnessing the G' minor, such that for every external edge (i,j): let B_i and B_j be the corresponding bushes containing i and j respectively. Let x_i and x_j be the corresponding centers. Then, either the distance of x_i to i, or the distance of x_j to j, is strictly less than d.*

Let us unpack this definition. Each bush is a graph of radius d. But to witness the minor G', the external edges need not be "maximally far" from the centers. So the minor is considered to have depth $d-(1/2)$, less than just d. An example can be seen in Fig. 2. Additionally, note that the distance between the centers must be strictly less than $2d+1$. While this may seem like an extremely technical condition, the half-integer depth minors play a crucial role in Theorem 1.1. To precisely capture the linear-time hardness of homomorphism counting, we need the classes defined through half-integer depth minors.

We define the central concept of the greatest reduced average density.

DEFINITION 1.3. (GRAD) *Let r be a non-negative half-integer. The rank r* greatest reduced average density (grad) *of a graph G is defined as:*

$$\nabla_r(G) = \max_{G' \in G \triangledown r} \left\{ \frac{|E_{G'}|}{|V_{G'}|} \right\}$$

In words, the rank r grad is the maximum average degree over all minors of G of depth r.

The classes of the hierarchy defined by Theorem 1.1 (and Fig. 1) are bounded $\nabla_{r/2}$ graph classes. Consider the simple case of $r = 0$. A depth 0 minor is just a subgraph. The rank 0 grad is the maximum average degree over subgraphs, which is (up to constant factors) the graph degeneracy.

A graph class with bounded ∇_0 is a class where all subgraphs of graphs in the class have bounded average degree. This is precisely $\mathcal{G}_0$ in our hierarchy. A graph class has *bounded expansion* if ∇_r is bounded for all r.

2 Main Ideas

Our result has many moving parts. In this section, we give a high-level overview with a focus on various obstacles we faced. Many new concepts and definitions were introduced to overcome these obstacles. Our result is obtained from marrying techniques from three sources: the deep theory of sparsity of Nešetřil and Ossona de Mendez [50], the DAG-treewidth of Bressan [14, 15], and the unique reachability and induced cycle obstructions of Bera et al. (denoted BPS and BGLSS) [11, 9]. Our lower bounds are fairly direct adaptations of techniques from BPS and BGLSS.

Graph orientations. Arguably, the starting point for any work on subgraph counting related to graph degeneracy is the clique-counting work of Chiba-Nishizeki [19]. A number of results recognized that the Chiba-Nishizeki ideas can be recast in terms of *graph orientations* [46, 56]. The primary challenge for homomorphism counting on sparse graphs is the presence of high-degree vertices. Such vertices kill any simple brute force BFS procedure to find homomorphisms. The idea is to orient/direct the edges of G into a DAG, such that *outdegrees* are bounded. We then search for homomorphisms/subgraphs in constant-radius outneighborhoods, which have bounded size.

A natural approach is to find an (acyclic) orientation that minimizes the maximum outdegree. The optimal quantity is called the *graph degeneracy*, and remarkably, there is a simple linear time procedure to find such a "degeneracy orientation" [46]. Moreover, this simple algorithm is intimately connected with $\nabla_0(G)$; the degeneracy is a 2-approximation of $\nabla_0(G)$. In words, all subgraphs of G have bounded average degree iff the degeneracy orientation has bounded outdegree. And we have assumed that $\nabla_0(G)$ is bounded, so we can orient G into a DAG $\vec{G}$ of bounded outdegree.

Homomorphism counting for bounded degeneracy graphs. We now outline the upper bound results of BPS and BGLSS, which fundamentally use Bressan's DAG-treewidth. Let us refer to an H-homomorphism/subgraph as a *match*.

Every H-match in G forms some directed match in $\vec{G}$. We can enumerate over all the (constant many) orientations $\vec{H}$ of H, and count $\vec{H}$-matches in $\vec{G}$. So we reduce to a directed acyclic homomorphism counting problem.

Suppose $\vec{H}$ has a single source vertex, so there is a rooted directed tree $\vec{T}$ spanning $\vec{H}$. Since $\vec{G}$ has bounded outdegree, there are $O(n)$ $\vec{T}$-matches in $\vec{G}$. (Which can be enumerated by a bounded depth outward BFS from each vertex.) We can enumerate over all these $\vec{T}$-matches, and see which of them induce $\vec{H}$-matches. When H is a clique, this recovers Chiba-Nishizeki's original algorithm. Moreover, this is probably one of the best practical algorithms for small clique counting [54].

The story gets interesting when $\vec{H}$ has multiple sources. In this case, $\vec{H}$ can be covered by a collection of rooted trees, one from each source in $\vec{H}$. These rooted trees are "fragments" of $\vec{H}$, which can be pieced together to yield an $\vec{H}$-match. For each fragment $\vec{T}$, we can enumerate all the $\vec{T}$-matches. The "piecing together" requires a careful indexing of all these matches.

When two fragment trees $\vec{T}$ and $\vec{T}'$ share a vertex (in $\vec{H}$), we have to select corresponding matches in $\vec{G}$ that share a vertex. It is challenging to index the tree matches appropriately to retrieve the relevant matches that might lead to an $\vec{H}$-match. A number of results designed ad hoc methods for orientations of various H [20, 54, 10]. A breakthrough was achieved by Bressan, who gave a systematic algorithm that indexes the fragments to efficiently count $\vec{H}$-matches [14]. He introduced a novel concept of the *DAG-tree decomposition*.

For a given $\vec{H}$, the DAG-tree decomposition is a tree $\mathcal{T}$ where nodes represent bags of sources in $\vec{H}$. Roughly speaking, each subtree of $\mathcal{T}$ represents a subgraph of $\vec{H}$ formed by all vertices reachable from the sources (in the bags) in $\mathcal{T}$. The subgraphs represented by independent subtrees can be counted/indexed independently. The non-trivial step is the "merging" of matches of children subtrees in $\mathcal{T}$. Suppose a node in $\mathcal{T}$ has two children, which represent the subgraphs $\vec{H}_1$ and $\vec{H}_2$. The parent node will represent a subgraph $\vec{H}'$ that contains $\vec{H}_1$ and $\vec{H}_2$. Roughly speaking, we construct $\vec{H}'$-matches by extending $\vec{H}_1$ and $\vec{H}_2$-matches through some shared vertices. These shared vertices are reachable from the sources in the bag represented by the parent node. The complexity of this step is determined by the bag size. The DAG-treewidth τ is the size of the largest bag, and the running time is $O(n^\tau)$. Relevant to us, when the DAG-treewidth is one, the algorithm runs in (near) linear time.

When is the DAG-treewidth of $\vec{H}$ one? This is precisely captured by BPS and BGLSS. If $LICL(H) < 6$, then for all orientations of $\vec{H}$, the DAG-treewidth is one. The proof of this fact involves a new concept of unique reachability graphs; but we defer the discussion of this point later.

The above summary gives the overall picture of proving the existence of linear time algorithms for H-homomorphism counting on bounded degeneracy graphs, where $LICL(H) < 6$.

2.1 The 6-cycle obstruction

We now explain the 6-cycle barrier. Consider the oriented 6-cycle $\vec{H}$ in the left of Fig. 3. It can be partitioned into three out-out wedges (paths of length 2), each corresponding to a unique source. Thus, $\vec{H}$ forms a "triangle" of out-out wedges. Counting $\vec{H}$-homomorphisms is equivalent to counting triangles in the following graph. In the oriented $\vec{G}$, enumerate all out-out wedges (u, v, w), where v denotes the

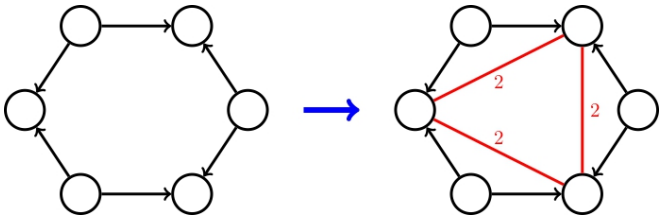

Figure 3: On the left, an oriented 6-cycle with 3 sources. This orientation has a DAG-treewidth greater than 1. It is not possible to decompose this pattern in a way that allows us to compute homomorphisms in linear time. The graph on the right is the result of connecting the endpoints of each out-out wedge, giving a fraternal augmentation. This new graph has a $LICL < 6$ and hence for any orientation of the red edges, the DAG-treewidth is 1.

wedge center. Create a new undirected graph G' with the edges (u, w). Since G has bounded degeneracy, $\vec{G}$ has bounded outdegree, and the number of out-out wedges is linear. So G' has $O(m)$ edges. Triangles in G' are precisely 6-cycles in $\vec{G}$. Indeed, this argument gives the hardness construction in BPS, reducing triangle counting in arbitrary graphs to 6-cycle counting in bounded degeneracy graphs.

This is the starting point for our investigation. Under what circumstance can 6-cycle counting be done in linear time? If the graph G' obtained above also had bounded degeneracy, then triangle counting in G' could be done in linear time (since G' has $O(m)$ edges). What condition does G need to satisfy for G' to have bounded degeneracy?

Enter shallow minors. Let us imagine contracting every alternate edge of the 6-cycle. This leads to a triangle minor. We can choose the centers of these contracted components with the following property. The three non-contracted edges are incident to some center. Hence, this forms a shallow minor of depth 1/2, according to Definition 1.2. Non-trivially, one can find a method of contracting G, so that all the 6-cycles in G are consistently contracted to triangles. Meaning, there is a 1/2-shallow minor G'' such that 6-cycles of G become triangles in G''. The shallow minor machinery of Nešetřil and Ossona de Mendez can be used to show if G'' has bounded degeneracy, then the graph G' (from the previous paragraph) also has bounded degeneracy.

Hence, if all 1/2-shallow minors of G have bounded degeneracy, then we can count triangles in G' in linear time. And the former condition is precisely saying that $\nabla_{1/2}(G)$ is bounded.

Implementing via fraternal augmentations. Let us implement the above approach so that it works for all H with $LICL(H) = 6$. We start with $\vec{G}$ and $\vec{H}$ as before, and assume that $\nabla_{1/2}(G)$ is bounded. We perform a series of *fraternal augmentations* in both $\vec{G}$ and $\vec{H}$. For every out-out wedge (u, v, w), we add the edge (u, w) to get the graphs $\vec{G'}$ and $\vec{H'}$. Note that new edges are undirected, so we try to orient them in $\vec{G'}$ so that the maximum outdegree is minimized. Denote this graph as $\vec{G''}$. We then enumerate over all orientations $\vec{H''}$ of the new edges in $\vec{H'}$. Finally, we count $\mathrm{Hom}_{\vec{H''}}(\vec{G''})$ and sum over all the $\vec{H''}$.

Since $\nabla_{1/2}(G)$ is bounded, we can prove that $\vec{G''}$ will have bounded outdegree. We can also show that $LICL(\vec{H''})$, treated as an undirected graph, will be strictly less than 6. The key is that the augmentations in $\vec{H}$ will reduce the length of *all* induced cycles. This is seen for the simple example of the 6-cycle in Fig. 3. Hence, the previous machinery of BPS and Bressan using width one DAG-tree decompositions can be applied to get a linear time algorithm. With some painstaking effort, one can push this approach to $LICL(H) < 8$. Essentially, fraternal augmentations in H reduce the $LICL$ to less than 6, at which point previous methods can run in linear time.

We note that the term "fraternal augmentation" was introduced by Nešetřil and Ossana de Mendes (Chap. 4 of [50]). But the idea is implicit in many previous results on subgraph counting in bounded degeneracy graphs [20, 54, 51].

2.2 More rounds of augmentations Consider the oriented 9-cycle pattern of Fig. 4. Let us perform a single round of fraternal augmentations, to get the red edges. The $LICL$ has now gone down to 6, so we still cannot count homomorphisms (of the resulting pattern) in linear time. Suppose we orient these new (red) edges, and perform another fraternal augmentation. This step adds the blue edges, and the $LICL$ is down to 3.

To count 9-cycle homomorphisms in linear time, we need to perform two rounds of fraternal augmentations

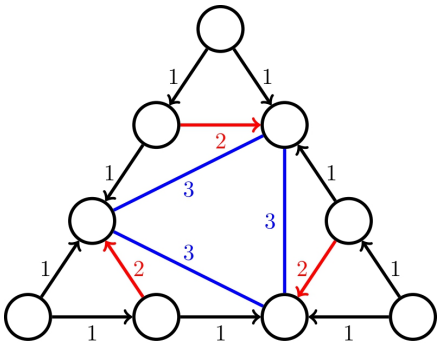

Figure 4: An example of performing fraternal augmentations on an oriented 9 cycle pattern (black edges). The first augmentation gives the red edges, reaching a situation analogous to the 6 cycle in Fig.3, an additional fraternal augmentation (blue edges) gives a pattern with an $LICL$ less than 6 and hence a DAG-treewidth of 1.

in G, and hope that the degeneracy of the resulting graph is bounded. One might imagine that if ∇_1 is bounded, then two rounds of augmentations will lead to a bounded degeneracy graph. It turns out the situation is far more nuanced. There are new obstacles for counting 9-cycle homomorphisms in linear time. This leads to the next technical tool.

Designing fraternity functions. Augmentations are really shortcuts in the graph; each augmentation represents a path of longer length. In general, we assume a bound on $\nabla_r(G)$ to get linear-time algorithms. Such a bound refers to r-shallow minors, which essentially contract paths of length at most $2r$. Our augmentations on such a graph should not shortcut a path that is longer than $2r$, since the bounded $\nabla_r(G)$ condition cannot say anything about such augmentations. Thus, we have to perform augmentations carefully so that the bounded $\nabla_r(G)$ condition can be used.

We discover that the way to perform such careful augmentations is by crafting specific *fraternity functions* of Nešetřil and Ossona de Mendez. This is a highly technical definition. At a high level, every augmented edge has a weight, which is (roughly) speaking the length of the path shortcut by this edge. Any subsequent augmentation is not allowed to exceed a weight threshold. We show an example of these weights in Fig. 4. The final augmentation is described by a fraternity function, which satisfies a number of consistency constraints. A deep result from the theory of sparsity is that if $\nabla_r(G)$ is bounded, then augmenting by a "$(2r+1)$-fraternity function" maintains bounded degeneracy. We apply this weighted fraternity function on both the input G and pattern H.

Maintaining homomorphism counts. There are some annoyances when performing augmentations for homomorphism counting. We explain these to motivate seemingly artificial technical conditions in our homomorphisms and final counting algorithms.

As we add more edges to $\vec{G}$, we may create "fake" $\vec{H}$ homomorphisms. On the flip side, when augmenting $\vec{H}$, some existing matches may be inadmissible (due to new edges in the pattern). We have a simple example in Fig.5 where augmentations do not preserve homomorphism counts.

We use two ideas to handle these problems. Firstly, we enforce that homomorphisms must be weight preserving, where the weights come from the fraternity functions described earlier. This prevents mapping of augmented edges to original edges and vice versa. Secondly, it is more convenient to create a new input instance from a graph product $G \times H$. We find H-homomorphisms in this product graph, where it is much easier to track the effect of augmentations on H-homomorphisms. All in all, we can then show direct correspondences between homomorphisms in the original graph G, and the homomorphisms in the final graph (constructed by graph products and a series of augmentation steps).

2.3 A major obstruction: extraneous induced cycles

So the overall story looks like the following. We have a graph G such that $\nabla_r(G)$ is bounded. We repeatedly perform augmentations, as long as they satisfy the constraints of a $(2r+1)$-fraternity function. Intuitively, one can think of $2r$ rounds of augmentations. The resulting graph G' has bounded degeneracy. One also performs similar augmentations on H to get the new pattern H'. (Of course, there is the extra complication of orienting every new edge that is created, but let us ignore that for now.)

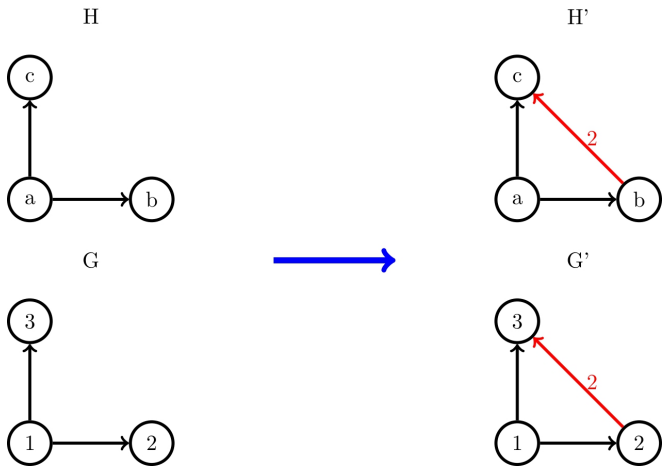

Figure 5: An example of how fraternal augmentations do not preserve homomorphisms. Consider the homomorphism ϕ from H to G with $\phi(a) = 1$ and $\phi(b) = \phi(c) = 2$. We can see how this will not be a valid homomorphism from H' to G' as the new edge connecting b and c is not preserved. However the number of subgraphs is preserved as the subgraphs $\{1, 2, 3\}$ in G and in G' are equivalent to H and H' respectively.

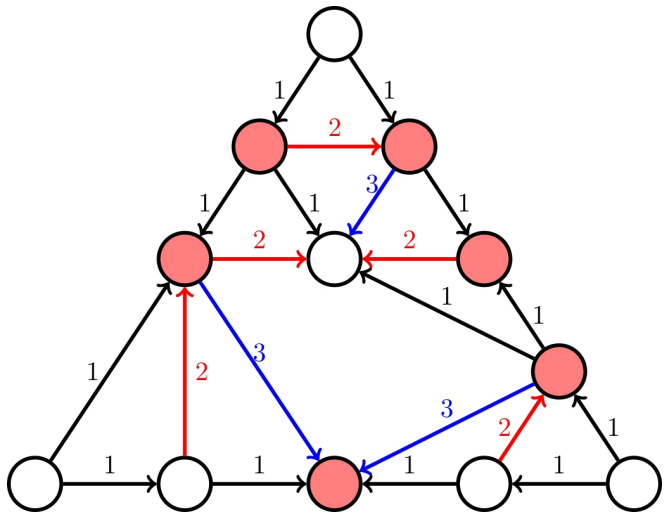

Figure 6: An example of a pattern where we have performed 2 iterations of the augmentation (we will call this a 3-fraternal extension of the pattern). As we can see the augmented pattern still has $LICL \geq 6$. However, the hub-treewidth is 1.

The hope is that $LICL(H')$ is strictly less than 6, in which case previous algorithms can count H'-homomorphisms in G' in linear time. Specifically, if $LICL(H) < 3(r+2)$, we would like $LICL(H')$ to be less than 6. We think that with sufficiently many rounds of augmentations, we can cut down the $LICL$ length.

And this is false. This statement fails, but only for a sufficiently complex example. The above approach does work for counting cycle homomorphisms, or when $LICL(H) < 8$. But there is a pattern H with $LICL(H) = 8$ where the approach breaks.

For ease of exposition, we present an example with $LICL(H) = 9$. Consider the pattern in Fig.6. The problem is that the newly added augmentation edges (given in blue and red) create a new induced cycle of length 6. This induced cycle is given by the red vertices. Unfortunately, we cannot guarantee a DAG-treewidth of one, so the existing algorithmic approach of BPS and BGLSS (as a black box) cannot yield a linear time algorithm.

At this stage, the authors thought an entire rethink was needed. Thankfully, that was not needed. The path around this obstruction is an unpacking of Bressan's algorithm and a deeper look into the BPS machinery. By getting to the core of these results, we can generalize them appropriately to deal with these "extraneous" induced cycle in the patterns.

Dealing with cyclicity. It turns out that a seemingly minor technicality is important to handling Fig.6. We started with a DAG $\vec{G}$ and a pattern $\vec{H}$. The reason to make G into a DAG $\vec{G}$ was that the degeneracy orientation was linear time computable and gave a DAG with constant outdegree. As a result, the pattern $\vec{H}$ is also a DAG, which motivated DAG-tree decomposition and DAG-treewidth.

When we augment, we add new *undirected* edges. To do a subsequent round of fraternal augmentations, we need to orient these edges, so that we can construct new out-out wedges. Every orientation has to keep the outdegree bounded. A natural approach is to extend the existing partial order (implied by the DAG $\vec{G}$). This actually *cannot* work. Meaning, if we want to keep the overall outdegree bounded after multiple augmentation rounds, then we must use cyclic orientations of G.

Hubsets to the rescue. So we need to deal with cyclic patterns $\vec{H}$, while Bressan's algorithm is tailored to DAG patterns. Our insight is that Bressan's algorithm is quite flexible, and we can generalize the concept of DAG sources to "hubsets". A hubset is a set of vertices from which all other vertices can be reached. The corresponding definitions of DAG-tree decomposition and DAG-treewidth all generalize to hubsets. Technically, the proofs of Bressan go through quite directly. But hubsets give us significantly more flexibility in minimizing the "hub treewidth".

Recall that the obstacle of Fig.4 has an induced cycle of length 6, and is not guaranteed to have DAG-treewidth one. But we can argue than the hub-treewidth is just one, which leads to a linear time algorithm for counting that pattern (when $\nabla_1(G)$ is bounded).

Extending BPS to hubsets. In order to take advantage of the hubsets we have to rework the machinery of BPS that related induced cycles to DAG-treewidth. All in all, we can prove the following. If $LICL(H) < 3(r+2)$, then (roughly speaking) after performing r rounds of fraternal augments, the resulting pattern $\vec{H}$ has a hub-treewidth of one.

2.4 Lower bounds and subgraphs The lower bounds closely follow the techniques of BPS and BGLSS [11, 9]. Using the tensorization techniques of Curticapean, Dell, and Marx, one can essentially show that the hardest patterns to count are cycles. The ideas in BPS and BGLSS are to use various graph products and manipulations, and they need to maintain the degeneracy of their various constructions. In our setting, we deal with more restrictive rank r bounded grad graphs, so we need some extra care in our arguments.

The hardness for cycle counting is fairly straightforward, and taken from [10]. We basically subdivide an edge into a longer path, and reduce triangle counting in arbitrary graphs to cycle counting in bounded grad graphs. We perform some calculations to show that the resulting graphs has bounded grad. The rank r determines the length of the subdivision, and hence the length of the cycle that a triangle is converted to. It suffices to show that the final graph has bounded rank r grad, which is quite direct. These simple constructions match the upper bounds of Theorem 1.1, completing the story for homomorphism counting.

The deep insight of Curticapean, Dell, and Marx is that, as their title says, homomorphisms form a good basis for subgraph counting [21]. Essentially, they show that for any quantity represented as a linear combination of homomorphisms, the complexity of computing that quantity is determined by the hardest homomorphism. It is fairly direct to see that H-subgraph counting can be done by an inclusion-exclusion on the various H'-homomorphisms (for $H' \in Spasm(H)$). Using techniques from [21], we can translate the inclusion-exclusion

algorithm into hardness for subgraph counting.

3 Related Work

The theory of sparsity is a deep topic at the intersection of graph theory, logic, and combinatorics. We refer the reader to the textbook [50]. Chapters 4, 5, and 7 contain most of the relevant background for our work.

We cannot do justice to the literature on homomorphism counting, which has an immense history. It was observed that the treewidth of the pattern plays a role in the final complexity. Díaz et al. [24] designed an algorithm with runtime $O(2^k n^{t(H)+1})$ where $t(H)$ is the treewidth of the target graph H. Dalmau and Jonsson [22] proved that such a dependence on treewidth is necessary. They show that that $\mathrm{Hom}_H(G)$ is polynomial time solvable if and only if H has bounded treewidth, otherwise it is $\#W[1]$-complete.

Relevant to our framework of restrictions on both G and H, Roth and Wellnitz [55] consider a doubly restricted version of $\mathrm{Hom}_H(G)$, where both H and G are from graph classes. They primarily focus on the parameterized dichotomy between poly-time solvable instances and $\#W[1]$-completeness.

Degeneracy is a measure of sparsity and has been known since the early work of Szekeres-Wilf [60]. We refer to reader to a recent short survey of Seshadhri on subgraph counting and degeneracy [56]. The family of bounded degeneracy graphs is quite rich: it involves all minor-closed families, bounded expansion families, and preferential attachment graphs. Most real-world graphs have small degeneracy ([35, 38, 58, 8, 12], also Table 2 in [8]).

Arguably the first work on exploiting degeneracy for subgraph counting is the seminal work of Chiba and Nishizeki [19]. Since then, it has been a central technique in theoretical and practical algorithms [30, 2, 40, 54, 51, 38, 52].

Bressan [14] introduced the concept of DAG-treewidth to design faster algorithms for homomorphism and subgraph counting problems in bounded degeneracy graphs. Bressan showed that for a pattern H with $|V_H| = k$ and an input graph G with $|E(G)| = m$ and degeneracy κ, one can count $\mathrm{Hom}_H(G)$ in $f(\kappa, k)O(m^{\tau(H)} \log m)$ time, where $\tau(H)$ is the DAG-treewidth of H. Assuming the exponential time hypothesis [36], the subgraph counting problem does not admit any $f(\kappa, k)m^{o(\tau(H)/\ln \tau(H))})$ algorithm, for any positive function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Recent work of Bressan, Lanziger, and Roth develops algorithms for pattern counting in directed graphs [16].

A focus on linear time algorithm in bounded degeneracy graphs was initiated by Bera, Pashanasangi, and Seshadhri [10]. They showed the lower bound for counting 6-cycles. That work was significantly generalized by BPS and BGLSS which completely characterized linear time homomorphism counting in bounded degeneracy graphs [11, 9].

There are numerous pattern counting results in Big Data models such as the property testing model [27, 28, 5, 29], the streaming model [6, 45, 41, 3, 39, 53, 47, 7, 12], and the Map Reduce model [20, 59, 42].

We now discuss the triangle detection conjecture. Itai and Rodeh [37] gave the first non-trivial algorithm for the triangle detection and finding problem with $O(m^{3/2})$ runtime. The best known algorithm for the triangle detection problem uses fast matrix multiplication and runs in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$ [4]. If $\omega = 2$, this yields a running time of $m^{4/3}$, which many believe to be the true complexity. The current best is $O(m^{1.41\cdots})$, using the best matrix multiplication algorithms. Any improvement on this bound would be considered a huge breakthrough in algorithms. Disproving the Triangle Detection Conjecture would require an algorithm that would go even beyond the best possible matrix multiplication based algorithm. We refer the reader to [1] for more details on Triangle Detection Conjecture.

4 Preliminaries

4.1 Graphs and Homomorphisms We use $G = (V_G, E_G)$ to denote the input graph, we will use $n = |V_G|$ for the number of vertices of G and $m = |E_G|$ for the number of edges. We use $H = (V_H, E_H)$ for the pattern graph and $k = |V_H|$ for the number of vertices of H, we consider k to have constant value. Both graphs are simple and undirected.

We will also have labeled graphs, a labeled graph is a graph $G = (V_G, E_G, L_G)$, where $L_G : V_G \to S$ is the label function that maps the vertices of the graph to a set of labels S. Additionally we will have weighted labeled graphs $G = (V_G, E_G, W_G, L_G)$ where $W_G : E_G \to \mathbb{N}$ is the weight function that maps the edges of the graph to the correspondent weight. We will use E_G^i to denote the the subset of edges of G with weight equal to i, that is, $E_G^i = \{e \in E_G : W_G(e) = i\}$.

A homomorphism from H to G is a mapping $\phi : V_H \to V_G$ where $\forall (u, u') \in E_H$ we have $(\phi(u), \phi(u')) \in E_G$. We use $\Phi(H, G)$ for the set of homomorphisms from H to G. We denote with $\mathrm{Hom}_H(G)$ to the problem of

counting the number of distinct homomorphisms from H to G, that is $\text{Hom}_H(G) = |\Phi(H, G)|$.

We extend these definitions for weighted and labeled graphs. Given two weighted labeled graphs H', G' we will define a homomorphism from H' to G' as a mapping $\phi : V_{H'} \to V_{G'}$ such that $\forall\ u \in V_{H'}\ L_{H'}(u) = L_{G'}(\phi(u))$ and $\forall (u, v) \in E_{H'}$ we have $(\phi(u), \phi(u')) \in E_{G'}$ and $W_{H'}((u, v)) \geq W_{G'}((\phi(u), \phi(v)))$. Similarly $\text{Hom}_{H'}(G')$ will correspond to the problem of counting the number of homomorphisms from H' to G'.

We use $LICL(H)$ for the largest induced cycle length of H, that is, the maximum length of any induced subgraph of H that forms a cycle. We use $Spasm(H)$ to refer to the spasm of H, that is, the collection of graphs obtained by contracting subsets of non adjacent vertices in H. $LICL(Spasm(H))$ will be the largest induced graph in all the graphs in the spasm of H.

4.2 Subgraph copies Given the graphs H and G we say that G' is a copy of H in G if G' is a subgraph of G such that there exists a 1:1 mapping from H to G' that preserves the edges. We use $\text{Sub}_H(G)$ for the problem of counting the number of distinct non-induced copies of H in G.

There is a direct relation between $\text{Sub}_H(G)$ and $\text{Hom}_{H'}(G)$ for the graphs $H' \in Spasm(H)$. The exact identity can be seen in [21], but we can express it as follows:

LEMMA 4.1. *Given two graphs G and H, for each graph $H_i \in Spasm(H)$ there exists a non-zero constant c_i such that:*

$$\text{Sub}_H(G) = \sum_{H_i \in Spasm(H)} c_i \text{Hom}_{H_i}(G)$$

4.3 Degeneracy and the degeneracy orientation A graph G is κ-degenerate if every subgraph of G has a minimum degree of at most κ. The degeneracy of G, $\kappa(G)$, is the maximum value of κ such that G is κ-degenerate. The degeneracy is also called the *coloring number*. A graph has bounded degeneracy when κ has constant value.

There is a way of orienting a graph acyclically, such that the maximum outdegree is upper bounded by the degeneracy of such graph. This is a classic result in graph theory (refer to Section 5.2 of [25] and survey [56]). To construct such orientation, one can generate an ordering of the vertices of G by iteratively selecting the lowest degree vertex in the graph and removing it. Hence the degeneracy orientation can be constructed in linear time.

FACT 4.1. *([46]) Given an undirected graph G with degeneracy $\kappa = \kappa(G)$, there exists an acyclical orientation $\vec{G}^{\kappa}$ of G such that the maximum outdegree of $\vec{G}^{\kappa}$ is κ. Moreover, this orientation can be computed in time $O(n+m)$.*

For a directed graph $\vec{G}$, we will use $\Delta^+(\vec{G})$ to refer to the maximum outdegree of any vertex of $\vec{G}$.

4.4 Shallow Topological Minors and Top-Grads We can define shallow topological minors, which play a useful role in our analysis.

DEFINITION 4.1. (SHALLOW TOPOLOGICAL MINOR [50]) *A shallow topological minor of a graph G of depth d is a graph G' obtained from G by taking a subgraph and then replacing an internally vertex disjoint family of paths of length at most $2d+1$ by single edges. G' is a shallow topological minor of G at depth d if there is a $\leq d$-subdivision of G' that is a subgraph of G.*

We use $G' \in G\ \tilde{\nabla}\ d$ to indicate that G' is a shallow topological minor of G at depth d.

Similar to grads, we can define the the topological greatest reduced average density:

DEFINITION 4.2. (TOP-GRAD [50]) *The topological greatest reduced average density (top-grad) with rank r of a graph G is defined as:*

$$\tilde{\nabla}_r(G) = \max_{G' \in G\ \tilde{\nabla}\ r} \left\{ \frac{|E_{G'}|}{|V_{G'}|} \right\}$$

Nešetřil and Ossona de Mendez proved that there is a polynomial relation between ∇ and $\tilde{\nabla}$ of a graph G, this is given by the following fact:

FACT 4.2. (COROLLARY 4.1 IN [50]) *For every graph G and every integer $r \geq 1$ it holds that*

$$\tilde{\nabla}_r(G) \leq \nabla_r(G) \leq 4(4\tilde{\nabla}_r(G))^{(r+1)^2}$$

This corollary directly gives us the following fact.

FACT 4.3. *For any constant r, the classes of bounded rank r grads and bounded rank r top-grads are equivalent. Moreover, for any graph G, $\nabla_r(G)$ is bounded if and only if $\tilde{\nabla}_r(G)$ is bounded.*

4.5 DAG-treewidth and Bressan's algorithm Bressan introduced the concepts of DAG-tree decomposition and DAG-treewidth of a directed acyclic graph $\vec{H}$ [14]. Before defining the DAG-tree decomposition of $\vec{H}$ we need to define a few concepts. We use $S = S(\vec{H})$ to denote the set of sources of $\vec{H}$, that is, the vertices in $\vec{H}$ with no in-edges. Given two vertices $u, v \in V_{\vec{H}}$, we say that v is reachable from u if there exists a direct path in $\vec{H}$ from u to v.

For a vertex $s \in \vec{H}$ we use $Reach_{\vec{H}}(s)$ to denote the set of vertices of $\vec{H}$ that are reachable from s. We can extend this definition to set of vertices, let $B \subseteq S$ is a set of vertices of $\vec{H}$, we use $Reach_{\vec{H}}(B) = \bigcup_{s \in B} Reach_{\vec{H}}(s)$ for the union of the reachability sets of the vertices in the set B. Additionally we use $\vec{H}(s)$ to represent the induced subgraph of $\vec{H}$ in the vertices of $Reach_{\vec{H}}(s)$.

We can now bring the definition of DAG-tree decomposition of $\vec{H}$:

DEFINITION 4.3. (DAG-TREE DECOMPOSITION[14]) *Let $\vec{H}$ be a directed acyclic graph with source set S. A DAG-tree decomposition of $\vec{H}$ is a rooted tree $T = (\mathcal{B}, \mathcal{E})$ with the following properties:*

1. *Each node $B \in \mathcal{B}$ is a bag of sources, $B \subseteq S$.*
2. *$\bigcup_{B \in \mathcal{B}} B = S$.*
3. *For all $B, B_1, B_2 \in \mathcal{B}$, if B is on the unique path between B_1 and B_2 in $\mathcal{T}$, then we have $Reach_{\vec{H}}(B_1) \cap Reach_{\vec{H}}(B_2) \subseteq Reach_{\vec{H}}(B)$.*

Bressan also defined the DAG-treewidth:

DEFINITION 4.4. (DAG-TREEWIDTH(τ) [14]) *The DAG-treewidth of a DAG-tree decomposition $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ is defined as the maximum bag size over all the bags of sources of $\mathcal{T}$:*

$$\tau(\mathcal{T}) = \max_{B \in \mathcal{B}} |B|$$

We also use $\tau(\vec{H})$ to refer to the DAG-treewidth of the directed graph $\vec{H}$, which is the minimum $\tau(\mathcal{T})$ over all possible DAG-tree decomposition $\mathcal{T}$ of $\vec{H}$.

Bressan introduced an algorithm that allows to compute the number of Homomorphisms between directed acyclical graphs $\vec{H}$ and $\vec{G}$. This algorithm uses the DAG-tree decomposition $\mathcal{T}$ of $\vec{H}$ to compute the homomorphisms from $\vec{H}$ to $\vec{G}$ from the homomorphisms of $\vec{H}(B)$ to $\vec{G}$ for each of the bags sources B in $\mathcal{T}$. When $\vec{G}$ has bounded outdegree and $\vec{H}$ has $\tau(\vec{H}) = 1$ this will take $O(n \log n)$ time. We can restate this result as follows:

LEMMA 4.2. ([14]) *Let $\vec{H}$ be a directed acyclic graph with $\tau(\vec{H}) = 1$ and let $\vec{G}$ be a directed acyclic graph with bounded maximum outdegree. There is an algorithm that computes $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in $O(n \log n))$ time.*

4.6 Graph Products We will use two different graph products in this work, they are standard definitions but we show them here for completeness:

DEFINITION 4.5. (CATEGORICAL PRODUCT) *Given two graphs G and G', we define their categorical product $G \times G'$ as follows:*

$$V_{G \times G'} = V_G \times V_{G'}$$

$$E_{G \times G'} = \{(\langle u, u' \rangle, \langle v, v' \rangle) : (u, v) \in E_G \text{ and } (u', v') \in E_{G'}\}$$

Definition 4.6. (Lexicographical Product) *Given two graphs G and G', we define their lexicographical product $G \bullet G'$ as follows:*

$$V_{G\bullet G'} = V_G \times V_{G'}$$

$$E_{G\bullet G'} = \{(\langle u, u'\rangle, \langle v, v'\rangle) : (u, v) \in E_G \text{ or } (u = v \text{ and } (u', v') \in E_{G'}\}$$

We can show that there is a direct relation between both products:

Fact 4.4. *For any constant $c > 0$: $G \times K_c = G \bullet \bar{K}_c$*

Proof. First, from both definitions we have that the vertex sets are equivalent: $V_{G\times K_c} = V_{G\bullet\bar{K}_c} = V_G \times V_{K_c}$.

We can also show that the edge sets are equivalent:

- $E_{G\times K_c} = \{(\langle u, u'\rangle, \langle v, v'\rangle) : (u, v) \in E_G \text{ and } (u', v') \in E_{K_c}\} = \{(\langle u, u'\rangle, \langle v, v'\rangle) : (u, v) \in E_G\}$.
- $E_{G\bullet\bar{K}_c} = \{(\langle u, u'\rangle, \langle v, v'\rangle) : (u, v) \in E_G \text{ or } (u = v \text{ and } (u', v') \in E_{\bar{K}_c}\} = \{(\langle u, u'\rangle, \langle v, v'\rangle) : (u, v) \in E_G\}$.

□

4.7 Fraternity Function Nešetřil and Ossona de Mendez introduced the notion of Fraternity Function in [50]:

Definition 4.7. (Fraternity Function [50]) *Let $\mathcal{V}$ be a finite set and let t be an integer. t-fraternity function is a function $\omega : \mathcal{V} \times \mathcal{V} \to \mathbb{N} \cup \{\infty\}$ such that for every $x, y \in \mathcal{V}$ one of $\omega(x, y)$ and $\omega(y, x)$ (at least) is ∞ and such that for every $x \neq y \in \mathcal{V}$:*

- *Either* $\min(\omega(x, y), \omega(y, x)) = 1$
- *Or* $\min(\omega(x, y), \omega(y, x)) = \min_{z\in\mathcal{V}\setminus\{x,y\}} \omega(z, x) + \omega(z, y)$ [2]
- *Or* $\min(\omega(x, y), \omega(y, x)) > t$ *and* $\min_{z\in\mathcal{V}\setminus\{x,y\}} \omega(z, x) + \omega(z, y) > t$.

Given a t-fraternity function ω we define the directed weighted graph $\vec{G}^\omega = (V_{\vec{G}^\omega}, E_{\vec{G}^\omega}, W_{\vec{G}^\omega})$ as the graph with vertex set $V_{\vec{G}^\omega} = \mathcal{V}$ whose edges are all the pairs $u, v \in \mathcal{V}$ such that $\omega(u, v) \leq t$ and for every edge (u, v) we have $W((u, v)) = \omega(u, v)$. We also define the directed graph $\vec{G}^\omega_i = (V_{\vec{G}^\omega_i}, E_{\vec{G}^\omega_i})$ as the graph with vertex set $V_{\vec{G}^\omega} = \mathcal{V}$ whose edges are all the pairs $u, v \in \mathcal{V}$ such that $\omega(u, v) = i$. We define $\Delta^+_i(\omega) = \Delta^+(\vec{G}^\omega_i)$.

We say that a directed weighted graph $\vec{G}$ forms a t-fraternity function if there is a t-fraternity function ω such that $\vec{G} = \vec{G}^\omega$. We call the fraternity function of G to ω in such cases.

5 Proving the Upper Bound

In this section we state the theorem that shows the upper bound of the Main Theorem and give a complete proof for it. In the following sections we will prove the different lemmas that compose the proof:

Theorem 5.1. *For all $t > 0 \in \mathbb{N}$, let G be an input graph with n vertices, m edges and bounded $\nabla_{(t-1)/2}(G)$. If $LICL(H) < 3(t + 1)$ then exists an algorithm that computes $\mathrm{Hom}_H(G)$ in $O(n \log n)$ time.*

Proof. Fix any $t > 0$.

First, we need to compute the labeled version of H and G, respectively H^L and F. This is because the fraternal extension procedures that we will perform in the following step will not preserve the number of homomorphisms if applied directly on H and G (see Fig. 5).

The constructions of these graphs are defined in §6. We can construct H^L in $O(k)$ time and F in $O(n)$ time (Claim 6.1). Additionally, we will show that this transformation preserves both the LICL of the pattern graph and the bounded grad conditions on the pattern and input graphs respectively:

[2] All the definitions in the Sparsity book use indegree instead of outdegree, which is more commonly used for subgraph counting using the degeneracy orientation. We will use the outdegree orientation instead and hence some of the definitions have been altered to reflect this.

- $LICL(H^L) = LICL(H)$
- For any constant i, $\nabla_i(G)$ is bounded if and only if $\nabla_i(F)$ is bounded. (Lemma 6.1).

We will also prove that the number of homomorphisms from H to G is equal to the number of homomorphisms from H^L to F (Claim 6.2).

Then, we will compute both the optimal acyclic t-Fraternal extension of F, $\vec{F}^{(t)}$, and the collection of t-fraternal extensions of H^L. We will formally define these concepts in §7. We will also show how to efficiently construct $\vec{F}^{(t)}$:

LEMMA 5.1. *Let F be a graph with $O(n)$ vertices and bounded $\nabla_{(t-1)/2}$, we can construct $\vec{F}^{(t)}$ in $O(t \cdot n)$ time.*

Additionally, in the same section we prove that $\vec{F}^{(t)}$ will have bounded maximum outdegree:

LEMMA 5.2. *Let F be a graph with $O(n)$ vertices and bounded $\nabla_{(t-1)/2}$, then the directed graph $\vec{F}^{(t)}$ has bounded max outdegree.*

$\Sigma(H,t)$ is independent on the input graph and will only depend on the pattern graph H. Because H is assumed to be constant sized we will be able to compute $\Sigma(H,t)$ in $O(1)$ time.

We can now reduce our problem from computing $\mathrm{Hom}_{H^L}(F)$ to computing $\mathrm{Hom}_{\vec{H}'}(\vec{F}^{(t)})$ for all $\vec{H}' \in \Sigma(H,t)$. In Subsection 7.4 we will show a direct equivalence between both quantities, given by the following lemma:

LEMMA 5.3.

$$\mathrm{Hom}_{H^L}(F) = \sum_{\vec{H}' \in \Sigma(H,t)} \mathrm{Hom}_{H'}(\vec{F}^{(t)})$$

In §8 we introduce the concepts of hubset, hub-tree decomposition and hub-treewidth. These concepts are a generalization of the source set, DAG-tree decomposition and DAG-treewidth respectively for directed graphs that are not acyclical. Using these new concepts, in §9 we show that there is a relation between the hub-treewidth of the t-fraternal extensions of H and the $LICL$ of H, as given by the following lemma:

LEMMA 5.4. *Let H be a pattern graph with $LICL(H) < 3(t+1)$, then for any t-fraternal extension $\vec{H}' \in \Sigma(H,t,)$ we have that $\tau(\vec{H}') = 1$.*

Hence, because H has an $LICL < 3(t+1)$ we will have that the hub-treewidth of all the graphs in $\Sigma(H,t)$ is 1.

Finally, in §10 we show how to compute $\mathrm{Hom}_{\vec{H}'}(\vec{F}^{(t)})$ for each $\vec{H}' \in \Sigma(H,t)$. By replacing the DAG-tree decomposition with the hub-tree decomposition we are able to adapt Bressan's algorithm [15] to work with labeled, weighted and directed graphs, giving the following lemma:

LEMMA 5.5. *Let $\vec{G}$ be a directed weighted and labeled graph with n vertices and bounded outdegree and let $\vec{H}$ be a directed weighted and labeled graph with $\tau(H) = 1$. There exists an algorithm that computes $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in $O(n \log n)$ time.*

Hence, we can compute $\mathrm{Hom}_{\vec{H}'}(\vec{G}^{(t)})$ in linear time for each graph $\vec{H}' \in \Sigma(H,t)$, as from Lemma 5.4 we have that $\tau(\vec{H}') = 1$ and from Lemma 5.2 we have that $\Delta^+(\vec{F}^{(t)})$ is bounded. We can then aggregate the counts using Lemma 5.3 to obtain the final homomorphism count. The whole process will take $O(n \log n)$ time.
□

6 Labeled Graphs

As we mentioned in the introduction, we can not use the fraternal extensions directly on G and H. Fraternal extensions do not preserve the number of homomorphisms (see Fig. 5). We define a pair of labeled graphs that can be obtained directly from H and G in linear time. Every homomorphism of the original graphs will translate into an injective homomorphism in the labeled graphs. We then will be able to do fraternal extensions on the labeled graphs while preserving the homomorphism count.

First, we define the labeled version of H, H^L which basically is H but with every vertex labeled to itself:

DEFINITION 6.1. (H^L) *Given a pattern graph $H = (V_H, E_H)$ we define the labeled graph $H^L = (V_{H^L}, E_{H^L}, L_{H^L})$, where:*

- $V_{H^L} = V_H$
- $E_{H^L} = E_H$
- $L_{H^L} : V_{H^L} \to V_{H^L}$ *is a labeling function such that* $\forall v \in V_{H^L}, L_{H^L}(v) = v$.

Now we define the graph F, this graph is obtained using the categorical product (Definition 4.5) of H^L and G:

DEFINITION 6.2. (F) *Given a pattern graph $H = (V_H, E_H)$ and an input graph $G = (V_G, E_G)$, we define the labeled graph $F = (V_F, E_F, L_F)$ as follows:*

- $V_F = V_{H \times G}$
- $E_F = E_{H \times G}$
- $L_F : V_F \to V_{H^L}$, *where* $\forall \langle u, v \rangle \in V_F \; L(\langle u, v \rangle) = u$

Constructing H^L is trivial and takes constant time $O(k)$. We can construct F efficiently as in the following claim:

CLAIM 6.1. *F has $O(n \cdot \kappa)$ vertices and $O(n \cdot \kappa \cdot k^2)$, and can be constructed in $O(n \cdot \kappa \cdot k^2)$ time.*

Proof. We can construct the vertices and assign the labels in $O(n \cdot k)$ time. We will have at most $n \cdot \kappa$ edges in E_G, and k^2 edges in E_H, hence we will take at most $O(n \cdot \kappa \cdot k^2)$ to generate E_F. The total complexity will be $O(n \cdot \kappa \cdot k^2)$. ∎

We can show that the number of homomorphisms from H to G is equivalent to the number of homomorphisms from H^L to F:

CLAIM 6.2.

$$\mathrm{Hom}_H(G) = \mathrm{Hom}_{H^L}(F)$$

Proof. We show that there is a bijection between the homomorphisms from H to G and from H^L to F:

- Consider a homomorphism ϕ from H to G, it will map the vertex u to $\phi(u)$, we can create a homomorphism ϕ' from H^L to F by mapping u to the vertex $\langle u, \phi(u) \rangle$ in V_F for all $u \in V_H$. If there is an edge in H between u_i, u_j we will have that there is also an edge between $\phi(u_i)$ and $\phi(u_j)$ in G, that implies by construction the existence of the arc $(\langle u_i, \phi(u_i) \rangle, \langle u_j, \phi(u_j) \rangle)$ in F, and hence ϕ' will be a valid homomorphism.
- Similarly, given a homomorphism ϕ' from H^L to F we can obtain a homomorphism ϕ from H to G by setting $\phi(u) = v : \phi'(u) = \langle u, v \rangle$. Again we need to show that this is a valid homomorphism: let u_i, u_j be two vertices in H, we have $\phi(u_i) = v_i = v : \phi'(u_i) = \langle u_i, v \rangle$ and $\phi(u_j) = v_j = v : \phi'(u_j) = \langle u_j, v \rangle$. If there is an edge in H between u_i and u_j we will have that there is also an edge between $\langle u_i, v_i \rangle$ and $\langle u_j, v_j \rangle$ in F, by construction that is only possible if there was also an edge between v_i and v_j in G and hence ϕ preserves the edge.

∎

Therefore, we have proven that we can create these labeled graphs H^L and F in linear time, and use them to count the homomorphisms instead. If we look at H^L we will have that $LICL(H^L) = LICL(H)$ as the only difference between these graphs are the labels, which do not influence the structure of the graph. However, in order to be able to use these graphs we need to show a similar property for the input graph G and its labeled version F, in this case, we need to be able to preserve the grad ∇, at least by a polynomial factor. In order to prove this we use the following proposition from [50]:

PROPOSITION 6.1. (PROP. 4.6 [50]) *Let G be a graph, let $p \geq 2$ be a positive integer and let r be a half-integer. Then*

$$\tilde{\nabla}_r(G \bullet K_p) \leq \max(2r(p-1)+1, p^2)\tilde{\nabla}_r(G) + p - 1$$

We can now prove the following lemma:

LEMMA 6.1. *Let G be an input graph, and H be a pattern graph with constant size k. For any constant i, $\nabla_i(G)$ is bounded if and only if $\nabla_i(F)$ is bounded.*

Proof. First, we have that $G \subseteq F$, hence $\nabla_i(G) < \nabla_i(F)$ and if $\nabla_i(F)$ is bounded so will $\nabla_i(G)$. We show the other direction: note that F will be a subgraph of $G \bullet K_k$:

$$F = G \times H \subseteq G \times K_k = G \bullet \bar{K}_k \subseteq G \bullet K_k$$

Where the second equality comes from Fact 4.4.

Lastly, using Prop. 6.1 we have that if $\tilde{\nabla}_i(G)$ is bounded so will $\tilde{\nabla}_i(G \bullet K_k)$, and hence so will $\tilde{\nabla}_i(F)$. Combining this with Fact 4.3 we get that $\nabla_i(F)$ will be bounded if $\nabla_i(G)$ is bounded. ∎

COROLLARY 6.1. *G has bounded degeneracy if and only if F has bounded degeneracy.*

7 Fraternal Extensions

In this section we formally introduce our augmentation procedure, we define the concept of fraternal extensions and show how to construct them efficiently. We will also prove some properties of the fraternal extensions and the relation between the homomorphisms from H to G and the ones of their fraternal extensions.

7.1 The Fraternal Extension Procedure

We start by formally defining the fraternal extension of a graph:

DEFINITION 7.1. (FRATERNAL EXTENSION) *Given a directed graph $\vec{G} = (V_{\vec{G}}, E_{\vec{G}})$ we say that the directed weighted graph $\vec{G}' = (V_{\vec{G}'}, E_{\vec{G}'}, W_{\vec{G}'})$ is a t-fraternal extension of $\vec{G}$ if $\vec{G}'$ forms a t-fraternity function and $E_{\vec{G}} = \{e \in E_{\vec{G}'} : W_{\vec{G}'}(e) = 1\}$.*

If G is an undirected graph, we say that $\vec{G}'$ is a t-fraternal extension of G if it is a t-fraternal extensions of some orientation $\vec{G}$ of G. The orientations of G with unit weights are 1-fraternal extensions of G.

Abusing notation, given a directed weighted graph $\vec{G}'$ that forms t-fraternity function we say that the directed weighted graph $\vec{G}''$ is a t'-fraternal extension of $\vec{G}'$ if $\vec{G}''$ forms a t'-fraternity function and $\forall i \in (1,t), \vec{G}'_i = \vec{G}''_i$.

Note that all these definitions can be extended to labeled graphs.

We can construct fraternal extensions of a graph efficiently using a recursive procedure. The initial step is to give unit weights to every edge in the graph, each orientation of the resultant graph will be a different fraternal extension. Then, for $i \in [2,t]$, for every out-out wedge with combined weight is i we add an undirected arc connecting its endpoints with weight i. Each orientation of the undirected edges will generate a distinct fraternal extension. The orientation of the undirected edges in each step will be different if we are performing this procedure on the pattern or on the input graph, as we will see in following subsections. We summarize the extension procedure in Alg. 1:

We can prove that the fraternal extensions will connect the endpoints of out-out wedges.

CLAIM 7.1. *If a graph $\vec{G}'$ is a t-fraternal extension of a graph $\vec{G}$, then for every out-out wedge (u,v,w) in $\vec{G}$ with $W_{\vec{G}}((v,u)) + W_{\vec{G}}(v,w) \leq t$ we have that there is an edge connecting u and w in $\vec{G}'$ of weight at most t.*

Proof. If $\vec{G}'$ is a t-fraternal extension of a graph $\vec{G}$ then by the definition of fraternal extension we have that $\vec{G}'$ must form a t-fraternity function, thus there is a t-fraternity function $\omega : \mathcal{V} \times \mathcal{V} \to \mathbb{N} \cup \{\infty\}$ such that $\vec{G}' = G^{\omega}$. Let (u,v,w) be an out-out wedge in $\vec{G}$ with $W_{\vec{G}}((v,u)) + W_{\vec{G}}(v,w) \leq t$.

Abusing notation we will refer with u,v,w to the correspondent elements in $\mathcal{V}$. Because ω is a t-fraternity function we have by the definition of fraternity function (Definition 4.7)that for u and w, either:

- $\min(\omega(u,w), \omega(w,u)) = 1$, in which case there is an edge of weight 1 connecting the two vertices.

Algorithm 1 Extension($\vec{G}$,t)

Input:
-Directed graph $\vec{G}$ (should be a $(t-1)$-fraternal extension)
-Integer t
Output:
-Set of edges E^t

Let $E^t = \emptyset$.
for each out-out wedge (u, v, w) in $\vec{G}$ **do**
  if $W_{\vec{G}}(v, u) + W_{\vec{G}}(v, w) = t$ and $(u, w) \notin E_{\vec{G}}$ and $(w, u) \notin E_{\vec{G}}$ **then**
    Add (u, w) to E^t.
  end if
end for
Return E^t

- $\min(\omega(u, w), \omega(w, u)) = \min_{z \in \mathcal{V} \setminus \{u,w\}} \omega(z, u) + \omega(z, w)$, for $z = v$ we have $\omega(z, u) + \omega(z, w) = W_{\vec{G}}((v, u)) + W_{\vec{G}}(v, w) \leq t$, and hence we have that there will be an edge of weight at most t connecting u to w or w to u.

- $\min(\omega(x, y), \omega(y, x)) > t$ and $\min_{z \in \mathcal{V} \setminus \{x,y\}} \omega(z, x) + \omega(z, y) > t$, which is not possible as for $z = v$ we have $\omega(z, u) + \omega(z, w) = W_{\vec{G}}((v, u)) + W_{\vec{G}}(v, w) \leq t$.

In all the cases we either reach to a contradiction or we prove the existence of the edge, completing the proof. □

7.2 The Fraternal Extensions of H In the case of the pattern graph H, we are interested in generating all the possible t-fraternal extensions of its labeled version H^L at an specific depth t. We will use $\Sigma(H, t)$ to denote such collection of graphs. We define it as follows:

DEFINITION 7.2. ($\Sigma(H, t)$) *Given a pattern graph H We call $\Sigma(H, t)$ to the collection of directed weighted and labeled graphs $\vec{H}'$ such that $\vec{H}'$ is a t-fraternal extension of any acyclical orientation of the labeled graph H^L.*

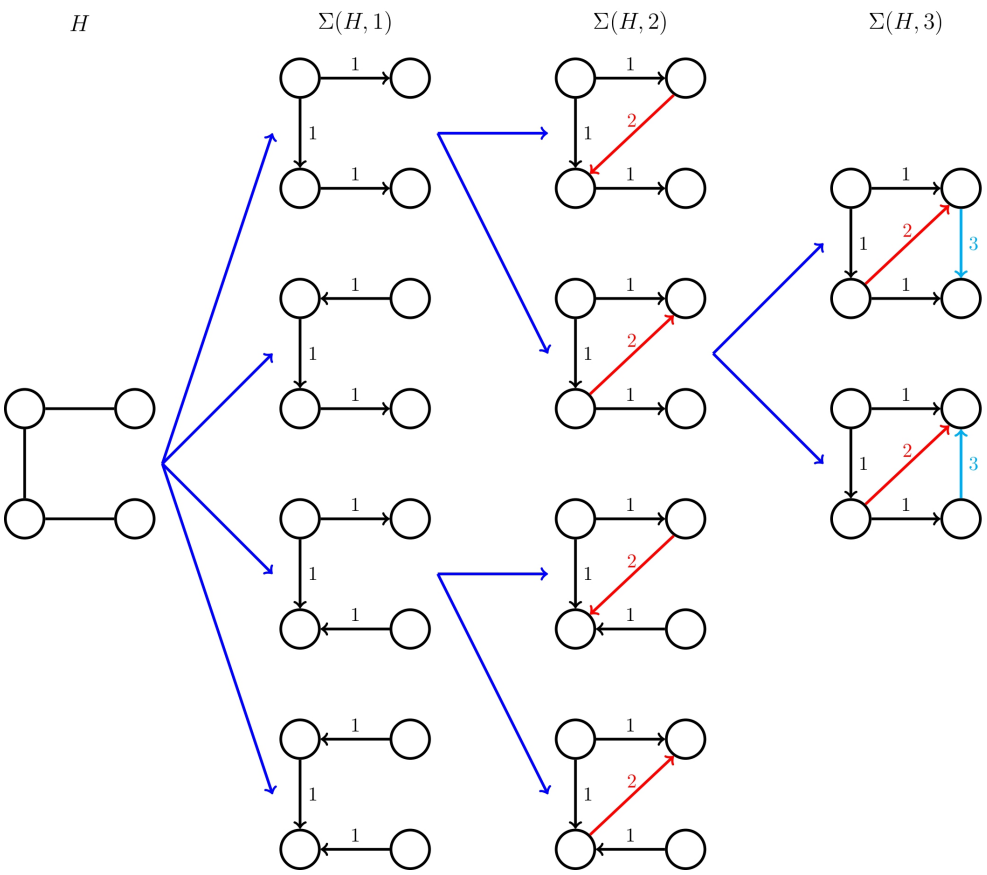

Figure 7: An example of how the fraternal extensions of a graph H are generated, the figure is not showing all the possible fraternal extensions. $\Sigma(H,1)$ corresponds to all the possible acyclical orientations of H. $\Sigma(H,2)$ adds the graphs obtained by adding edges between out-out wedges in the graphs of $\Sigma(H,1)$, this may generate some new out-out wedges that will be connected by edges in the graphs in $\Sigma(H,3)$. Note that a graph in $\Sigma(H,1)$ with no out-out wedges will also be part of $\Sigma(H,2)$ and $\Sigma(H,3)$.

Note that $\Sigma(H,1)$ corresponds exactly with the collection of acyclical orientations of H (adding unit weights to the edges and trivial labels to the vertices).

Let $\vec{H}_i$ be a i-fraternal extension of H. Abusing notation, we will use $\Sigma(\vec{H}_i, t)$ for $t > i$ to denote the set of t-fraternal extensions $\vec{H}'$ of H such that $\vec{H}'$ is also a t-fraternal extension of $\vec{H}_i$.

Obtaining the collection of fraternal extensions of a graph can be seen as an iterative process where we apply the extension procedure defined in Alg. 1 to all the fraternal extensions of the previous layer and then consider all the possible orientations of these new edges, this process is summarized in Alg. 2. Fig. 7 shows an example of this process applied to a simple pattern graph.

7.3 The Fraternal Extensions of F

We now formally define the optimal acyclic t-fraternal extension of F:

Definition 7.3. (Optimal Acyclic t-Fraternal Extension $\vec{F}^{(t)}$) *Given a graph F we use $\vec{F}^{(t)}$ to denote an optimal acyclic t-fraternal extension of F. It is defined as follows:*

- *$\vec{F}^{(t)}$ is a t-fraternal extension of F.*
- *For every $i \in [1,t]$ we have that the edges $\vec{F}_i^{(t)}$ are oriented acyclically following the degeneracy orientation of the edges with weight i.*

Note that for a graph F there are multiple graphs that will follows those properties, as the degeneracy orientation

Algorithm 2 ComputeExtensions(H^L, t)

Input:
-Labeled Graph H^L
-Integer t
Output:
-$\Sigma(H,t)$

Set $\Sigma(H,1)=\emptyset$
for each acyclical orientation $\vec{H}$ of H^L **do**
  Add $\vec{H}$ to $\Sigma(H,1)$ with unit weights.
end for
for $i \in [2,t]$ **do**
  Set $\Sigma(H,i)=\emptyset$
  for $\vec{H} \in \Sigma(H,i-1)$ **do**
    Let E^i = Extension$(\vec{H},i)$
    for each orientation $\vec{E}^i$ of the edges E^i with weights i **do**
      Add $\vec{H} \cup \vec{E}^i$ to $\Sigma(H,i)$
    end for
  end for
end for
Return $\Sigma(H,t)$

of a set of undirected edges might not be unique. We can select any arbitrary degeneracy orientation whenever it is not unique.

We can show that there is a strong relation between the $\nabla_{(t-1)/2}$ of a graph F and the maximum outdegree of any of its optimal acyclic t-fraternal extensions:

LEMMA 5.2. *Let F be a graph with $O(n)$ vertices and bounded $\nabla_{(t-1)/2}$, then the directed graph $\vec{F}^{(t)}$ has bounded max outdegree.*

Proof. We prove by induction on t:

When $t=1$, $\vec{F}^{(1)}$ corresponds with the degeneracy orientation of F. Because F has bounded $\nabla_{(t-1)/2}$, we get that it also has bounded ∇_0 and hence, bounded degeneracy. Therefore, the degeneracy orientation of F will have bounded outdegree $\Delta^+(\vec{F}^{(1)})$.

Now, for the inductive step, assume that for any $k<t$, $\Delta^+(\vec{F}^{(k)})$ is bounded. We prove that $\Delta^+(\vec{F}^{(k+1)})$ will also be bounded. First, we bring the following lemma from [50]:

LEMMA 7.1. (LEMMA 7.6 IN [50]) *Let $\mathcal{V}$ be a finite set, let $k \geq 1$ be an integer, let ω be a k-fraternity function and let $F=G_1^\omega$.*

There exists a $(k+1)$-fraternity function ω' such that:

$$\forall (x,y) \in \mathcal{V}^2, \omega(x,y) \leq k \Longrightarrow \omega'(x,y)=\omega(x,y)$$

$$\Delta^+_{k+1}(\omega') \leq \tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N_\omega(k+1)})$$

Here $N_\omega(k+1)$ is a real valued function defined in section 7.4 of [50] and its value is polynomial in $\Delta_1^+(\omega)+,\ldots,\Delta_k^+(\omega)$.

Let ω_k be the fraternity function of $\vec{F}^{(k)}$. By our assumption $\Delta^+(\vec{F}^{(k)})$ is bounded, hence $\Delta_1^+(\omega)+,\ldots,\Delta_k^+(\omega)$ will also be bounded and so will $N=N_\omega(k+1)$. Applying Lemma 7.1 we know that there exists a function ω' such that $\forall (x,y) \in \mathcal{V}^2, \omega_k(x,y) \leq k \Longrightarrow \omega'(x,y)=\omega_k(x,y)$ and $\Delta^+_{k+1}(\omega') \leq \tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N}$.

In other words, there exists a $k+1$-fraternal extension $\vec{F}'=G^{\omega'}$ of F such that its outdegree is at most $\tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N_\omega(k+1)})$. We must show now that $\tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N_\omega(k+1)})$ is bounded:

Recall that $N = N_\omega(k+1)$ is bounded. We have that $k < t$ hence $k/2 \leq (t-1)/2$ and $\nabla_{k/2}(F) \leq \nabla_{(t-1)/2}(F)$. Therefore, because F has bounded $\nabla_{(t-1)/2}$, it will also have bounded $\nabla_{k/2}$. Now, applying Prop. 6.1 we get that $\tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N})$ will also be bounded as N is a constant, and we get that:

$$\Delta^+(\vec{F}')_{k+1} = \Delta^+(G^{\omega'})_{k+1} = \Delta^+_{k+1} \leq \tilde{\nabla}_{k/2}(F \bullet \overline{K}_{1+N})$$

Therefore $\Delta^+(\vec{F}')_{k+1}$ will be bounded.

However is possible that ω' is orienting the edges of the $k+1$ layer in an orientation that is not acyclical, we can show that the max outdegree of the degeneracy orientation is as most a factor of 2 with respect to the minimum max outdegree of all cyclical orientations:

CLAIM 7.2. *Given an undirected graph G, let $\vec{G}^*$ be the orientation of G with minimal Δ^+ and let $\vec{G}^\kappa$ be the degeneracy orientation of G:*

$$\Delta^+(G^\kappa) \leq 2\Delta^+(G^*)$$

Proof. $\Delta^+(\vec{G}^\kappa)$ is equal to the degeneracy κ of G. There exists a subgraph G' in G such that has minimum degree $\kappa = \Delta^+(\vec{G}^\kappa)$. Let n' be the number of vertices in G' and m' the number of edges. We will have that $m' \geq \frac{n' \cdot \Delta^+(\vec{G}^\kappa)}{2}$.

If we divide all the m' edges equally so that the maximum outdegree in G' is minimized we will have that every vertex has an outdegree of at least $\frac{\Delta^+(\vec{G}^\kappa)}{2}$. Hence any orientation of the edges of G' will have an outdegree of at least $\frac{\Delta^+(\vec{G}^\kappa)}{2}$. Therefore, $\Delta^+(\vec{G}^*) \geq \frac{\Delta^+(\vec{G}^\kappa)}{2}$. ∎

Therefore $\Delta^+_{k+1}(\vec{F}^{(k+1)}) \leq 2\Delta^+(\vec{F}')_{k+1}$, and hence, it is bounded. Because for all the other layers $\vec{F}^{(k)}$ and $\vec{F}^{(k+1)}$ are identical and $\vec{F}^{(k)}$ has bounded outdegree, we get that $\Delta^+(\vec{F}^{(k+1)})$ is bounded.
∎

Additionally, if F belongs to the class of graphs with bounded rank $(t-1)/2$ grad, we can then compute $\vec{F}^{(t)}$ efficiently. The process is summarized in Alg. 3, and basically consist of perform multiple iterations of the extension procedure, orienting the edges by the degeneracy orientation each time. We get Lemma 5.1:

LEMMA 5.1. *Let F be a graph with $O(n)$ vertices and bounded $\nabla_{(t-1)/2}$, we can construct $\vec{F}^{(t)}$ in $O(t \cdot n)$ time.*

Proof. First, because F has bounded $\nabla_{(t-1)/2}$ it will also have bounded degeneracy κ. We can then construct $\vec{F}^{(1)}$ by orienting F acyclically using the degeneracy orientation, by Fact 4.1 this will take linear time on the number of vertices and edges of F, which is still $O(n)$.

By Lemma 7.1 we will have that for all $k \leq t$ the graph $\vec{F}^{(k)}$ has bounded Δ^+. We show an inductive process where given $\vec{F}^{(k)}$ we can compute $\vec{F}^{(k+1)}$ in linear time:

- First, compute all the out-out wedges of $\vec{F}^{(k)}$. This can be done in linear time as the outdegree of every vertex is bounded. The number of out-out wedges will also be linear in n.
- For every out-out wedge (u, v, w), if the sum of the weights of the two edges (v, u) and (v, w) is $k+1$ then create an edge connecting (u, w) with weight $k+1$. This process takes linear time in the number of out-out wedges, which is again, linear in n. And there will be at most $O(n)$ edges in the $k+1$ layer.
- Finally, orient the newly created edges using the degeneracy orientation when considering only the new edges. This again can be done in $O(n)$ time.

We complete the proof by showing that the resultant graph $\vec{F}'$ is an optimal acyclic $k+1$-fraternal extension of F, $\vec{F}^{(k+1)}$. First, we can see that every layer in the graph is oriented by the degeneracy orientation. That was true for the layer 1 to k as we started our construction with $\vec{F}^{(k)}$ and we haven't added any additional edge with a weight $\leq k$. It is also true for the $k+1$ layer as the last step of our construction orients the layer using the degeneracy orientation.

Now we just need to show that $\vec{F}'$ is a $k+1$-fraternal extension of F. Let $\mathcal{V} = V_{\vec{F}'}$ and $\omega : \mathcal{V} \times \mathcal{V} \to \mathbb{N}$ be a function such that $G^\omega = \vec{F}'$. We must show that ω forms a $k+1$-fraternity function. We prove by contradiction, if ω is not a $k+1$ fraternity function, then by Definition 4.7 we have that there must exist a pair of vertices in $V_{\vec{F}'}$ such that the equivalent nodes u, v in $\mathcal{V}$ do not meet any of the following conditions:

1. $\min(\omega(u,v), \omega(v,u)) = 1$
2. $\min(\omega(u,v), \omega(v,u)) = \min_{w \in \mathcal{V} \setminus \{u,v\}} \omega(w,u) + \omega(w,v)$
3. $\min(\omega(u,v), \omega(v,u)) > k+1$ and $\min_{w \in \mathcal{V} \setminus \{u,v\}} \omega(w,u) + \omega(w,v) > k+1$.

If u and v do not meet the first condition then we have that $\min(\omega(u,v), \omega(v,u)) > 1$ (recall that in a fraternity function a missing edge is considered as ∞). If $\min(\omega(u,v), \omega(v,u)) < k+1$ then u, v was an edge in $\vec{F}^{(k)}$ which implies that it will meet the second condition, as $\vec{F}^{(k)}$ was a valid k-fraternity extension. Hence, we have that $\min(\omega(u,v), \omega(v,u))$ is either $k+1$ or $> k+1$:

- In the first case $\min(\omega(u,v), \omega(v,u)) = k+1$: because the second condition is not true we will have that $\min_{w \in \mathcal{V} \setminus \{u,v\}} \omega(w,u) + \omega(w,v) \neq k+1$. If it is greater than $k+1$ then our procedure would not have generated an edge connecting u, v with weight $k+1$. Otherwise we have that is lower than $k+1$ we have that $\min_{w \in \mathcal{V} \setminus \{u,v\}} \leq k$, but then in $\vec{F}^{(t)}$ we will have an edge connecting u and v with weight k or it would not be a valid fraternal extension. Both cases reach to a contradiction.
- In the second case we have that $\min(\omega(u,v), \omega(v,u)) > k+1$: Then because the third condition is false, we will have that $\min_{w \in \mathcal{V} \setminus \{u,v\}} \omega(w,u) + \omega(w,v) \leq k+1$. But in that case an out-out wedge with weight at most $k+1$ would connect u and v, and hence our procedure would have generated an edge connecting u and v with weight at most $k+1$, again reaching a contradiction.

□

Algorithm 3 CompOptimalExtension(F,t)

Input:
-Labeled Graph F
-Integer t
Output:
-$\vec{F}^{(t)}$

Let $\vec{F}^{(1)}$ be the degeneracy orientation of F with unit weights.
for $k \in [2, t]$ **do**
  Let $E^i = \text{Extension}(\vec{F}^{(i-1)}, i)$
  Let $\vec{E}^i$ be the degeneracy orientation of the edges in E^i with weight i.
  Set $\vec{F}^{(i)} = \vec{F}^{(i-1)} \cup \vec{E}^i$
end for
Return $\vec{F}^{(t)}$

7.4 Equivalence of Homomorphisms In this section we prove the equivalence between the homomorphisms of the original graphs and the fraternal extensions. This is given by Lemma 5.3, that we restate:

LEMMA 5.3.

$$\text{Hom}_{H^L}(F) = \sum_{\vec{H}' \in \Sigma(H,t)} \text{Hom}_{H'}(\vec{F}^{(t)})$$

Proof. Let $\Phi(H^L, F)$ be the set of homomorphisms from H^L to F. For every $\vec{H}' \in \Sigma(H,t)$, let $\Phi(\vec{H}', \vec{F}^{(t)})$ be the set of homomorphisms from $\vec{H}'$ to $\vec{F}^{(t)}$. We can see that each of these sets are disjoint:

CLAIM 7.3. *Let $\vec{H}', \vec{H}'' \in \Sigma(H,t)$ be two distinct t-fraternal extensions of H:*

$$\Phi(\vec{H}', \vec{F}^{(t)}) \cap \Phi(\vec{H}'', \vec{F}^{(t)}) = \emptyset.$$

Proof. First, we show that if $\vec{H}'$ and $\vec{H}''$ are distinct t-fraternal extensions of H then there must exist an edge $e \in E_{\vec{H}'}$ such that it is reversed in $\vec{H}''$. We prove it by contradiction: Assume there is no such edge, we show that then $\vec{H}' = \vec{H}''$. We do induction in the depth of the fraternal extensions:

- For the base case, $\vec{H}'$ and $\vec{H}''$ are both 1-fraternal extensions and therefore they will correspond to different orientations of the edges in H, if they don't differ in any edge then $\vec{H}'$ and $\vec{H}''$ will correspond to the exact same orientation.
- For the inductive step, assume for some $k < t$ that if there is no reversed edge in two k-fraternal extensions $\vec{H}'$,$\vec{H}''$ of H, then $\vec{H}' = \vec{H}''$. We show that same holds for $k+1$-fraternal extensions. By the assumption we know that two k-fraternal extensions that do not differ in any edge will correspond to the same graph. Hence, two $k+1$-fraternal extension $\vec{H}', \vec{H}'' \in \Sigma(H, k+1)$ will have the same edges (with the exact same orientations) up to the layer k, and therefore the edges in the layer $k+1$ must be the same. Because we are assuming that there are not reversed edges, they will also have the same orientation. Therefore $\vec{H}' = \vec{H}''$.

Hence, there exists an edge e that have different orientations in $\vec{H}'$ and $\vec{H}''$. Let u, v be the endpoints of the edge e, the arc (u,v) belongs to $\vec{H}'$ and the arc (v,u) to $\vec{H}''$. We show that no homomorphism ϕ can be both in $\Phi(\vec{H}', \vec{F}^{(t)})$ and in $\Phi(\vec{H}'', \vec{F}^{(t)})$. Consider the vertices $\phi(u)$ and $\phi(v)$ of $\vec{F}^{(t)}$, we can have either an arc from $\phi(u)$ to $\phi(v)$, from $\phi(v)$ to $\phi(u)$, or none. In order for ϕ to be a valid homomorphism of $\vec{H}'$, we will need the directed arc $(\phi(u), \phi(v))$ to be in $\vec{F}^{(t)}$, similarly for $\vec{H}''$ we will need the directed arc $(\phi(v), \phi(u))$ to be in $\vec{F}^{(t)}$. A fraternal extension can not contain two opposite edges connecting the same two vertices. Therefore ϕ can not be a homomorphism of $\vec{H}'$ and $\vec{H}''$ at the same time. ∎

Now we just need to show that:

$$\Phi(H^L, F) = \bigcup_{\vec{H}' \in \Sigma(H,t)} \Phi(\vec{H}', \vec{F}^{(t)})$$

We start by proving that for every $\vec{H}' \in \Sigma(H,t)$, if ϕ is a homomorphism from $\vec{H}'$ to $\vec{F}^{(t)}$ then it will also be a valid homomorphism from H^L to F: Let $\vec{H}' \in \Sigma(H,t)$ be a t-fraternal extension of H and ϕ a homomorphism from $\vec{H}'$ to $\vec{F}^{(t)}$, ϕ must be an injective mapping, hence every vertex in $\vec{H}'$ is mapped to a distinct vertex in $\vec{F}^{(t)}$.

We show that ϕ is also a valid homomorphism from H^L to F, consider the edge $(u,v) \in H^L$, we must show that the edge $(\phi(u), \phi(v))$ is present in F. Because $\vec{H}'$ is a fraternal extension of H^L, we will have that they share the edges of weight 1, hence either the arc (u,v) or the arc (v,u) will be present in $\vec{H}'$ (we can assume without loss of generality that it is oriented from u to v) with unit weight $W_{\vec{H}'}((u,v)) = 1$. Because ϕ is a homomorphism from $\vec{H}'$ to $\vec{F}^{(t)}$ we will have that the edge $(\phi(u), \phi(v))$ must be present in $\vec{F}^{(t)}$ with weight $W_{\vec{F}^{(t)}}((u,v)) = W_{\vec{H}'}((u,v)) = 1$. For $\vec{F}^{(t)}$ to have an edge with weight 1, such edge must be also in F, and hence $(\phi(u), \phi(v))$ is present in E_F.

Now we prove that if ϕ is a homomorphism from H^L to F then there exists a fraternal extension $\vec{H}' \in \Sigma(H,t)$ such that ϕ is a valid homomorphism from $\vec{H}'$ to $\vec{F}^{(t)}$: Let ϕ be a homomorphism from H^L to F, we show that we can construct a t-fraternal extension of H^L such that ϕ is a valid homomorphism from it to $\vec{F}^{(t)}$. We use induction on t:

- For the base case $k = 1$, we can orient every edge (u,v) in H^L so it matches the orientation of the edge $(\phi(u), \phi(v))$ in $\vec{F}^{(1)}$. This orientation will be acyclic and therefore the resultant graph will be in $\Sigma(H, 1,)$.
- For the inductive step, we assume that for $k < t$ exists a k-fraternal extension $\vec{H}^k$ of H where ϕ is a valid homomorphism from $\vec{H}^k$ to $\vec{F}^{(k)}$. We prove that it also holds for $k+1$: First, let $\vec{H}'$ be a $k+1$-fraternal

extension of $\vec{H}^k$, any edge (u,v) in $\vec{H}'$ with $W_{\vec{H}'}((u,v)) < t$ must have a correspondent edge $(\phi(u),\phi(v))$ in $\vec{F}^{(k)}$ and hence in $\vec{F}^{(k+1)}$ with $W_{\vec{F}^{(k+1)}}((\phi(u),\phi(v))) < W_{\vec{H}'}((u,v))$.

Hence we only need to verify that the edges e in $\vec{H}'$ with $W_{\vec{H}'}(e) = k+1$ are also present in $\vec{F}^{(t)}$. Consider the edge $e = (u,v)$ in $\vec{H}'$ with $W_{\vec{H}'}(e) = k+1$, we need to show that $\vec{F}^{(t)}$ contains either $(\phi(u),\phi(v))$ or $(\phi(v),\phi(u))$ as the edges in the last layer of $\vec{H}'$ can be oriented arbitrarily and still will be a valid $k+1$-fraternal extension of $\vec{H}^k$. If such edge e exists, then there must exist a vertex w in $\vec{H}'$ such that there is an out-out wedge (u,w,v) in $\vec{H}^k$ with total weight $k+1$, by the assumption, we will have that there is an out-out wedge $(\phi(u),\phi(w),\phi(v))$ in $\vec{F}^{(k)}$ with total weight $\leq k+1$. Hence $\vec{F}^{(t)}$ must include either $(\phi(u),\phi(v))$ or $(\phi(v),\phi(u))$ with weight $\leq k+1$.

□

8 The Hub-Set

As we mentioned before, the fraternal extensions of an acyclic graph might no longer be acyclic. Bressan's algorithm for counting homomorphism requires of directed acyclic graphs, as it relays on the definitions of DAG-tree decomposition and DAG-treewidth. We will generalize these definitions and extend Bressan's algorithm to directed graphs that are not necessarily acyclic. For that purpose we introduce the concept of hubset of a directed graph:

DEFINITION 8.1. (HUBSET) *Let $\vec{H}$ be a directed graph, a hubset of $\vec{H}$ is any subset of vertices $\mathcal{S} \subseteq V_{\vec{H}}$ such that:*

- *For each pair $s, s' \in \mathcal{S}$ with $s \neq s'$ there is no directed path connecting s to s' (or vice versa).*
- *For each vertex $v \in V_{\vec{H}} \setminus \mathcal{S}$, there exists a vertex $s \in \mathcal{S}$ such that there is a directed path connecting s to v.*

We will use $\mathcal{S}(\vec{H})$ to denote any hubset of $\vec{H}$. Note that when $\vec{H}$ is acyclic the hubset of $\vec{H}$ is unique and corresponds exactly with the source set. Furthermore, this applies to any fraternal extension of a DAG as we can see in the following claim:

CLAIM 8.1. *Let $\vec{H}'$ be a fraternal extension of the DAG $\vec{H}$. $\vec{H}$ has an unique hubset and $\mathcal{S}(\vec{H}') = S(\vec{H})$.*

Proof. Note that the edges of $\vec{H}$ are a subset of the edges of $\vec{H}'$. Hence, the second condition for the hubset is automatically satisfied as every vertex in $\vec{H}$ is reachable from at least one source in $S(\vec{H})$. For the first condition suffices to observe that the indegree of any of the sources will be 0 in all the fraternal extensions of $\vec{H}$. We can prove it by contradiction, assume that there is a fraternal extension that adds an edge incident to the source $s \in S(\vec{H})$, then we will have that s was one of the endpoints of an out-out wedge. That is not possible as s is a source and hence its initial indegree in $\vec{H}$ was 0.

Now we prove that the hubset is unique, note that by the previous argument every source of $\vec{H}$ will still be source in $\vec{H}'$. Hence all the sources must be included in the hubset in order to satisfy the second condition. Adding any additional vertex in the hubset is also not possible, as all the vertices are reachable from at least one source and we would violate the first condition. □

Note that every graph $\Sigma(H,1)$ is a DAG, therefore the previous claim will apply to every graph in $\Sigma(H,t)$ for every $t > 0$.

We can now define a new type of decomposition of a graph based on the hubset. This new decomposition is just a generalization of Bressan's DAG-tree decomposition for directed graphs:

DEFINITION 8.2. (HUB-TREE DECOMPOSITION) *Let $\vec{H}$ be a directed graph with hubset $\mathcal{S} = \mathcal{S}(\vec{H})$. A hub-tree decomposition of $\vec{H}$ is a rooted tree $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ with the following properties:*

1. *Each node $B \in \mathcal{B}$ is a subset of $\mathcal{S}$ of $\vec{H}$, $B \subseteq \mathcal{S}$.*
2. *$\bigcup_{B \in \mathcal{B}} B = \mathcal{S}$.*

3. *For all $B, B_1, B_2 \in \mathcal{B}$, if B is on the unique path between B_1 and B_2 in $\mathcal{T}$, then we have $Reach_{\vec{H}}(B_1) \cap Reach_{\vec{H}}(B_2) \subseteq Reach_{\vec{H}}(B)$.*

We similarly define the hub-treewidth of a graph:

DEFINITION 8.3. (HUB-TREEWIDTH(τ)) *The hub-treewidth of a hub-tree decomposition $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ is defined as the maximum size over all the nodes of $\mathcal{T}$:*

$$\tau(\mathcal{T}) = \max_{B \in \mathcal{B}} |B|$$

We also use $\tau(\vec{H})$ to refer to the hub-treewidth of the directed graph $\vec{H}$, which is the minimum $\tau(\mathcal{T})$ over all possible hub-tree decomposition of $\vec{H}$.

Note that when $\vec{H}$ is acyclic the definitions for DAG-tree decomposition and hub-tree decomposition are equivalent. Similarly we will have that the DAG-treewidth and hub-treewidth are the same, that is why we will refer to both of them using τ.

9 The hub-treewidth of fraternal extensions and the LICL

As we can see in Fig.6, the fraternal extensions do not necessarily reduce the $LICL$ of the pattern graphs, as new cycles can be formed with the new edges in the extensions. However, there is a clear relation between the $LICL$ of the original graph and the hub-treewidth of the fraternal extensions. We will be proving such relation in this section, given by the following lemma:

LEMMA 5.4. *Let H be a pattern graph with $LICL(H) < 3(t+1)$, then for any t-fraternal extension $\vec{H}' \in \Sigma(H, t,)$ we have that $\tau(\vec{H}') = 1$.*

9.1 Main Technical Lemma

In this subsection we prove the main technical lemma of this paper, which will allow us to prove the relation between fraternal extensions and dag-treewidth. First we will define a long out-out wedge:

DEFINITION 9.1. (LONG OUT-OUT WEDGE) *A long out-out wedge is a graph form by the union of two directed paths of any length as the result of combining their sources. Fig.8 shows an example of a long out-out wedge.*

We now prove the following claims that will be used in the main lemma of this subsection:

CLAIM 9.1. *Let u, v be the endpoints of a long out-out wedge with total weight w and l edges in some fraternal extension $\vec{H}_i$ of H, then for all $t \geq w$, for all $\vec{H}' \in \Sigma(\vec{H}_i, t)$ there is a direct path connecting either u to v or v to u using only the vertices in the long out-out wedge.*

Proof. We can prove by induction on the number of edges of the long out-out wedge:

The base case is when $l = 2$, this is simply a standard out-out wedge with where the two edges have a combined weight of w. By Claim 7.1 we have that any t-fraternal extension of $\vec{H}_i$ at depth $t \geq w$ will add an edge connecting u, v if it was not already present. Hence, we will either have a path from u to v or v to u.

Now we show the inductive step: Assume that the claim holds for $l = k$, we will prove that it also holds for $l = k + 1$.

Let s be the source of the long out-out wedge with length $k + 1$, s will be forming a normal out-out wedge with a vertex u' in the $s - u$ path and a vertex v' in the $s - v$ path, both (s, u') and (s, v') edges will have a combined weight strictly less than w, thus for some $t' < w$ by Claim 7.1 any fraternal extension of H_i at level t' will have an edge (u', v') (we can assume without loss of generality that it will go from u' to v').

If $u' = u$ then we actually a directed path from u to v, otherwise, we have a long out-out wedge where the total weight is still at most w, but with the source at u', with one less edge, hence $l = k$. Using the assumption of the inductive step we know that there will be a path either from u to v or from v to u in the t-fraternal extension. □

In Fig.8 we can see an example of the long out-out wedge construction.

We now prove the following:

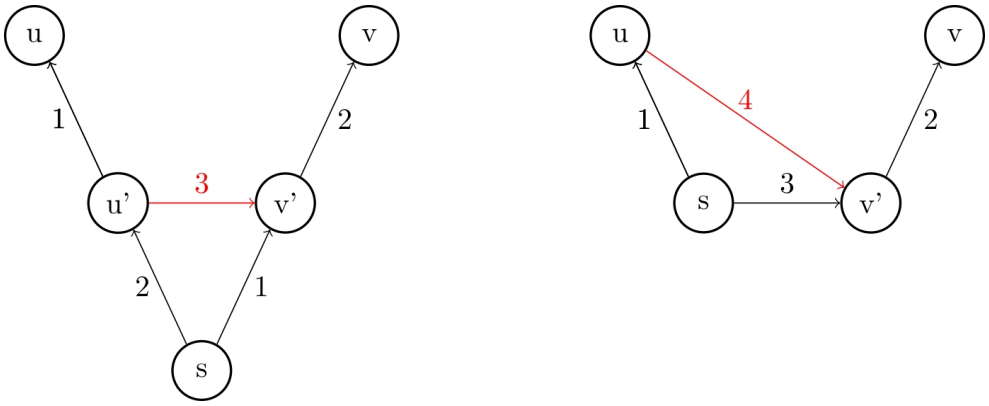

Figure 8: An example of connectivity between the end-points of a long out-out wedge, we can see how u' becomes the source of a new long out-out wedge after adding the edge of weight 3. When adding the edge of weight 4 in the right picture we end up with a direct path between u and v.

CLAIM 9.2. *Let P be an induced undirected path of length t from u to v in H, then for any t-fraternal extension $\vec{H} \in \Sigma(H,t)$: There is an undirected path P' connecting u and v (ignoring edge directions) using only vertices in P with total weight at most t, such that P' either:*

- *Case 1: Forms a direct path from u to v.*
- *Case 2: Forms a direct path from v to u.*
- *Case 3: Contains a vertex s such that there is a direct path from both u and v to s in P' (a long in-in wedge).*

Proof. We can prove by induction on t:

Our base case is when $t = 1$, we have that u and v are connected by one edge in H, in every orientation of H we will have that either the edge goes from u to v (Case 1) or from v to u (Case 2).

For the inductive step we assume that the claim holds for all $t \leq k$, we will show that it also holds for $t = k+1$:

Let u and v be two vertices in H inducing the path P of length $k+1$. Let u' be the vertex adjacent to u in P, and let e be the edge connecting u and u'. Then u' and v form an induced path of length k using the vertices of P.

Using the assumption of the inductive step, we have that for all the k-fraternal extensions $\vec{H}_k \in \Sigma(H,k)$ we will have that there is a path P'' connecting u' and v (ignoring directions) using only vertices in P with total weight at most k, following one of the three cases. We will prove that all the cases lead to the construction of a path P' from u to v for all $\vec{H}_{k+1} \in \Sigma(\vec{H}_k, k+1)$ following one of the three conditions:

- Case 1: We have that P'' is a direct path from u' to v with weight at most k. We have two possibilities depending on the orientation of e:
  - (a) e is oriented from u to u': then $e \cup P''$ forms a direct path from u to v with total weight at most $k+1$.
  - (b) e is oriented from u' to u: then, in $\vec{H}_k$, $e \cup P''$ forms a long out-out wedge with the source at u' with at most $k+1$ edges and at most $k+1$ total weight. Hence by Claim 9.1 we have that there will be a direct path in all $\vec{H}_{k+1} \in \Sigma(\vec{H}_k, k+1)$ from u to v or from v to u.
- Case 2: We have that P'' is a direct path from v to u' with weight at most k. We have two possibilities depending on the orientation of e:
  - (a) e is oriented from u to u': then we have that u' is reachable from both u and v in $\vec{H}_k$.
  - (b) e is oriented from u' to u: then $e \cup P''$ forms a direct path from v to u with total weight at most $k+1$.

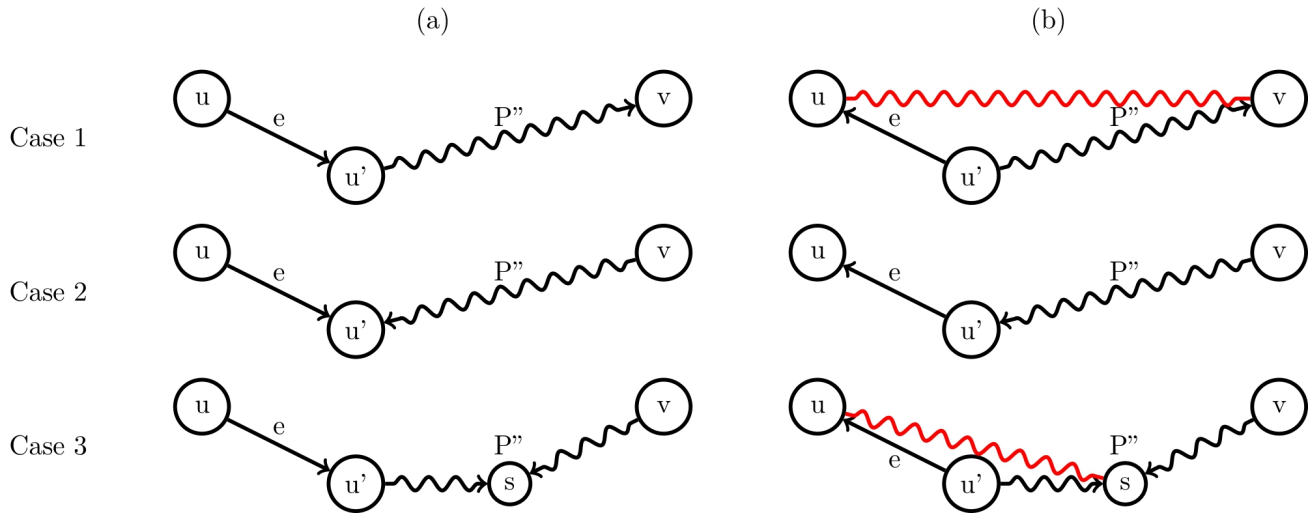

Figure 9: The 6 possible scenarios in the proof of Claim 9.2 depending on the orientations of e and P''. The red edges represent the cases where a long out-out wedge appears, we know the endpoints of the long out-out wedges will connect because of Claim 9.1

- Case 3: We have that P'' has a vertex s such that there is a direct path from both u' and v to s in P''. We have two possibilities depending on the orientation of e:
  (a) e is oriented from u to u': then we have a direct path from u to s, and $e \cup P''$ form a long in-in wedge ending in s.
  (b) e is oriented from u' to u: then we can see how in $\vec{H}_k$, u and s form a long out-out wedge in $e \cup P''$ centered in u' with less than k edges and less than k weight. Hence by Claim 9.1 we have that there will be a direct path in all $\vec{H}_{k+1} \in \Sigma(\vec{H}_k, k+1)$ from u to s or from s to u. In the first case, s will be reachable from both u and v, and in the second case we have a direct path from v to u.

As we can see, every possibility lead to one of the three cases in the claim. In Fig. 9 there is a depiction of all the cases.
□

Before presenting the main technical lemma, we bring the definition of Unique reachability graph from [11]. However this definition was created for directed acyclical graphs as it uses the sources of the graph. We adapt it to use the hubset instead:

DEFINITION 9.2. (UNIQUE REACHABILITY GRAPH) *Let $\vec{H}$ be a directed graph with hubset $\mathcal{S} = \mathcal{S}(\vec{H})$ and $\mathcal{S}_p \subseteq \mathcal{S}$ be a subset of the hubset. We define a unique reachability graph $UR_{\mathcal{S}_p}(\mathcal{S}_p, E_{\mathcal{S}_p})$ on the vertex set $\mathcal{S}_p$, and the edge set $E_{\mathcal{S}_p}$ such that there exists an edge $e = \{s_1, s_2\} \in E_{\mathcal{S}_p}$, for $s_1, s_2 \in \mathcal{S}_p$ if and only if the set $(Reach_{\vec{H}}(s_1) \cap Reach_{\vec{H}}(s_2)) \setminus Reach_{\vec{H}}(\mathcal{S}_p \setminus \{s_1, s_2\})$ is non-empty.*

We can now finally introduce the main technical lemma:

LEMMA 9.1. *Given a pattern graph H, if $LICL(H) < 3(t+1)$ then for any graph $\vec{H}' \in \Sigma(H, t)$ we have that $UR_{\vec{H}'}$ is acyclical.*

Proof. We prove a stronger statement, let the graph $\vec{H}' \in \Sigma(H, t)$ be a t-fraternal extension of H. If $UR_{\vec{H}'}$ contains a cycle of length l, then H will contain an induced cycle of length at least $(t+1) \cdot l$.

Consider the subset $\mathcal{S}'(\vec{H}')$ of l vertices from the hubset $\mathcal{S}(\vec{H})$ forming the cycle in $UR_{\vec{H}'}$, we have $\mathcal{S}'(\vec{H}') \subseteq \mathcal{S}(\vec{H}')$. We can enumerate them as $s_1, s_2, \ldots, s_l$, where for $i \in [1, l]$ we have that s_i and s_{i+1} (with $s_{l+1} = s_1$) share a edge in $UR_{\vec{H}'}$.

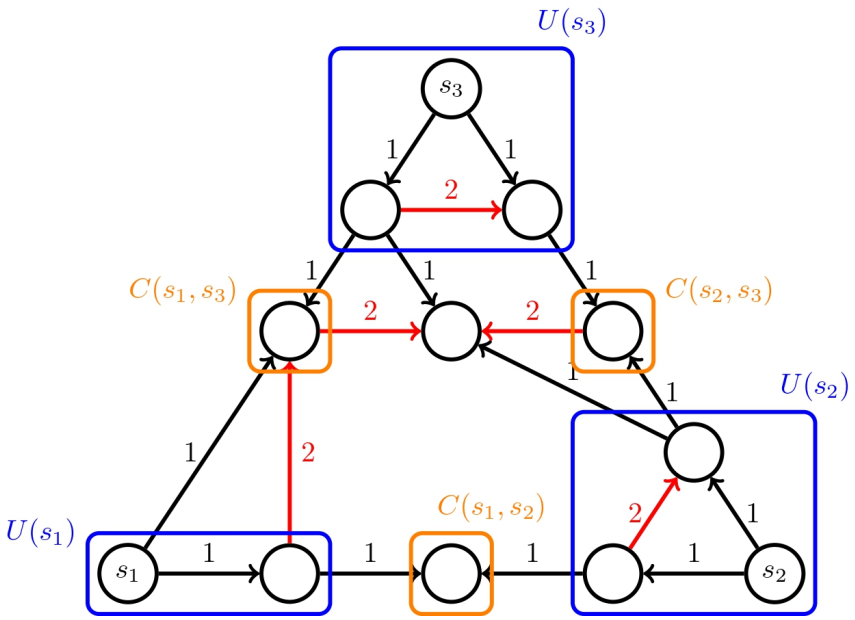

Figure 10: An example of a graph with three sources that will form the hubset. The U and C regions for each source or pair of sources are highlighted. This graph is 2-fraternal extension where the Unique Reachability Graph will have a cycle, we can see how the original graph (if we only consider edges with weight 1) contains an induced cycle of length at least 9.

For each vertex $s_i \in \mathcal{S}'(\vec{H}')$, we define $U(s_j)$ as the subset of vertices of $\vec{H}'$ that are reachable by s_i but are not reachable by any vertex in $\mathcal{S}'(\vec{H}') \setminus s_i$. Note that $U(s_i)$ is not empty as we will have $s_i \in U(s_i)$. Also for $s_i \neq s_j$ there can not be any edge in H connecting any vertex in $U(s_i)$ to any vertex in $U(s_j)$ or one of the vertices would be reachable by at least 2 vertices in $\mathcal{S}'(\vec{H}')$.

Similarly, for each pair of vertices $s_i, s_j \in \mathcal{S}'(\vec{H}')$, let $C(s_i, s_j)$ be the subset of vertices of $\vec{H}'$ that is reachable by both s_i and s_j but not by any other vertex in $\mathcal{S}'(\vec{H}') \setminus \{s_i, s_j\}$. Note that for $i \in [1, l]$ we will have that $C(s_i, s_{i+1})$ is not empty, as they share an edge in $UR_{\vec{H}'}$. Again for $i \neq j$ there can not be any edge in H connecting any vertex in $C(s_i, s_{i+1})$ with any vertex in $C(s_j, s_{j+1})$ or one of the vertices would be reachable by at least 3 vertices in $\mathcal{S}'(\vec{H}')$.

In Fig. 10 we show an example of the definitions of U and C.

Let $\mathcal{V}_{\mathcal{S}'} = \bigcup_{i=1}^{l} U(s_i) \cup \bigcup_{i=1}^{l} C(s_i, s_{i+1})$. We claim that there is an induced cycle of length $(t+1) \cdot l$ in the induced subgraph of H by $\mathcal{V}_{\mathcal{S}'}$.

We define $A(s_i) = C(s_{i-1}, s_i) \cup U(s_i) \cup C(s_i, s_{i+1})$, that is, the portion of $\mathcal{V}_{\mathcal{S}'}$ reachable by s_i.

Now we can show the following claim:

Claim 9.3. *For all $i \in [1, l]$, for all $u \in C(s_{i-1}, s_i)$ (with $s_0 = s_l$) and for all $w \in C(s_i, s_{i+1})$ (with $s_{l+1} = s_1$), exists an induced path in H of length at least $t+1$ connecting u and w that only uses vertices in $A(s_i)$.*

Proof. Let P be the shortest path in H from u to w that only uses vertices in $A(s_i)$. Because this is a shortest path it will also be an induced path. Such a path will always exists as there is a path from the vertex s_i to all the vertices in $A(s_i)$ that only uses such vertices. We prove that it will have length at least $t+1$:

Assume the opposite, then there is a path from u to w in P that only uses vertices in $A(s_i)$ with at most t edges. However by Claim 9.2 we would have that in $\vec{H}'$ there is a path $P' \subseteq P$ where P' either:

- Case 1: Is a direct path from u to w, which would mean that w is reachable by three vertices of the hubset and hence not in $C(s_i, s_{i+1})$.
- Case 2: Is a direct path from w to u, which would mean that u is reachable by three vertices of the hubset and hence not in $C(s_{i-1}, s_i)$.

- Case 3: Has a vertex s that is reachable from both u and w, but in that case s would be reachable from three vertices of the hubset and it would not be in $\mathcal{V}_{\mathcal{S}'}$.

Hence we reach a contradiction in all the cases and therefore the shortest path must have length at least $t+1$, concluding the proof of this claim.
□

We can now select one vertex $u_i \in C(s_i, s_{i+1})$ for each $i \in [1, l]$ such that the total length for all $i \in [1, l]$ of the paths connecting u_i and u_{i-1} using the vertices in $A(s_i)$ is minimized. We have a total of l paths.

If there is a common vertex between two of the paths (ignoring the ends of the paths) then setting that vertex as the end vertex would yield a shorter path, hence we have that the l paths only intersect in their ends, and thus, combining them we obtain a cycle of length $l \cdot (t+1)$ (as each of the individual paths have length $t+1$ by the previous claim).

Only left to show is that this cycle is actually an induced cycle: The paths forming the cycle are all induced, so suffices to show that there are no edges connecting two different paths. There are no edges connecting two different $U(s_i)$ or two different $C(s_i, s_{i+1})$, hence the only possibility would be to have two vertices in the same $C(s_i, s_{i+1})$ of two different paths connected by an edge but if that is the case, replacing the center of that C region by any of the two vertices would reach a shorter total length, which is not possible. Hence we have that the cycle will be an induced cycle. □

9.2 Rest of the proof In this section we complete the proof of Lemma 5.4, the proof is very similar to the proof for Lemma 4.4 in [11]. For completeness we will include the whole proof with the convenient modifications. We will start by defining a partial hub-tree decomposition which is a generalization of the partial DAG-tree decomposition introduced in [11]:

DEFINITION 9.3. (PARTIAL HUB-TREE DECOMPOSITION) *Let $\vec{H}$ be a directed graph with hubset set $\mathcal{S} = \mathcal{S}(\vec{H})$. For a subset $\mathcal{S}_p \subseteq \mathcal{S}$, a partial hub-tree decomposition of $\vec{H}$ with respect to $\mathcal{S}_p$ is a tree $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ with the following three properties.*

1. *Each node $B \in \mathcal{B}$ is a subset of $\mathcal{S}_p$: $B \subseteq S_p$.*
2. *The union of the nodes in $\mathcal{T}$ is the entire set $\mathcal{S}_p$: $\bigcup_{B \in \mathcal{B}} B = S_p$.*
3. *For all $B, B_1, B_2 \in \mathcal{B}$, if B is on the unique path between B_1 and B_2 in $\mathcal{T}$, then we have $Reach_{\vec{H}}(B_1) \cap Reach_{\vec{H}}(B_2) \subseteq Reach_{\vec{H}}(B)$.*

In the case that $\mathcal{S}_p = \mathcal{S}$ this definition corresponds exactly with the hub-tree decomposition.

Now we bring a few more definitions from [11], again generalized to directed graphs:

DEFINITION 9.4. (INTERSECTION-COVER AND $\mathcal{S}_p$-COVER) *Let $\vec{H}$ be a directed graph with hubset $\mathcal{S} = \mathcal{S}(\vec{H})$. Let s_1 and s_2 be a pair of vertices in $\mathcal{S}$. We call a vertex $s \in \mathcal{S}$ an intersection-cover of s_1 and s_2 if $Reach(s_1) \cap Reach(s_2) \subseteq Reach(s)$. Assume $\mathcal{S}_p \subseteq \mathcal{S}$ is a subset of the hubset $\mathcal{S}$. We call a vertex $s \in \mathcal{S}$, a $\mathcal{S}_p$-cover of $s_1 \in \mathcal{S}$ if for each vertex $s_2 \in \mathcal{S}_p$, s is an intersection-cover for s_1 and s_2.*

DEFINITION 9.5. (GOOD-PAIR) *Let $\vec{H}$ be a directed graph with hubset $\mathcal{S} = \mathcal{S}(\vec{H})$. Let $x \in \mathcal{S}$ be a vertex of the hubset and $\mathcal{T}_{\mathcal{S}_p}$ be a partial hub-tree decomposition of width one for $\mathcal{S}_p \subset \mathcal{S}$ where $x \notin \mathcal{S}_p$. We call the pair $(x, \mathcal{T}_{\mathcal{S}_p})$ a good-pair if there exists a leaf node $l \in \mathcal{T}_{\mathcal{S}_p}$ connected to the node $d \in \mathcal{T}_{\mathcal{S}_p}$ such that d is an intersection-cover for x and l.*

We also restate the following Lemma, presented in [11] as Lemma 4.8, but extending it to non-acyclical directed graphs:

LEMMA 9.2. *[Equivalent to Lemma 4.8 of [11]] Let $\vec{H}$ be a directed graph with hubset $\mathcal{S} = \mathcal{S}(\vec{H})$ and let $\mathcal{S}_p \subset \mathcal{S}$ be a subset of the hubset. Assume $\mathcal{T}$ is a partial hub-tree decomposition for $\mathcal{S}_p$ with $\tau(\mathcal{T}) = 1$. Consider a vertex $s \in \mathcal{S}$ such that $s \notin \mathcal{S}_p$. If d is a $\mathcal{S}_p$-cover of s, then connecting s to d in $\mathcal{T}$ as a leaf results in a tree $\mathcal{T}'$ that is a partial hub-tree decomposition for $\mathcal{S}_p \cup \{s\}$. Furthermore, $\tau(\mathcal{T}') = 1$*

Proof. Because we had $\tau(\mathcal{T}) = 1$ and we are just adding a leaf with a single vertex to $\mathcal{T}$, we will have that $\tau(\mathcal{T}') = 1$. Therefore, suffices to show that $\mathcal{T}'$ is a valid partial hub-tree decomposition for $\mathcal{S}_p \cup \{s\}$. We prove by contradiction: Assume it is not, then there must exist three nodes $s_1, s_2, s_3 \in \mathcal{T}'$, with s_3 being in the path between s_1 and s_2, such that $Reach(s_1) \cap Reach(s_2) \not\subseteq Reach(s_3)$. If $s \notin \{s_1, s_2, s_3\}$, then this is not possible as all the nodes were already in $\mathcal{T}$ and it was a valid hub-tree decomposition. Hence s must be one of the three vertices, it can not be s_2 as s is a leaf, we can assume without loss of generalization that $s = s_3$. Now s_2 lies on the unique path between s_1 and s.

If $s_2 = d$, then because d is a $\mathcal{S}_p$-cover of s we will have that $Reach(s_1) \cap Reach(s) \subseteq Reach(d)$, reaching a contradiction. Otherwise, $s_2 \neq d$ but s_2 must lie in the path between s_1 and d, and hence $Reach(s_1) \cap Reach(d) \subseteq Reach(s_2)$ and we also had that $Reach(s_1) \cap Reach(s) \subseteq Reach(d)$ hence $Reach(s_1) \cap Reach(s) \subseteq Reach(s_2)$. Reaching a contradiction. ∎

Now we can prove the main lemma of this section, again following closely the proof of Lemma 4.4 of [11]. We restate the lemma:

LEMMA 5.4. *Let H be a pattern graph with $LICL(H) < 3(t+1)$, then for any t-fraternal extension $\vec{H}' \in \Sigma(H, t,)$ we have that $\tau(\vec{H}') = 1$.*

Proof. Let $\vec{H}'$ be a t-fraternal extension of H and $\mathcal{S} = \mathcal{S}(\vec{H}')$ be the hubset of $\vec{H}'$. Let $\mathcal{S}_p \subseteq \mathcal{S}$ denote a subset of $\mathcal{S}$. We prove by induction on the size of $\mathcal{S}_p$, that there exists a partial hub-tree decomposition of width 1 for each $\mathcal{S}_p \subseteq \mathcal{S}$. If $\mathcal{S}_p = \mathcal{S}$ then we have that there is a hub-tree decomposition for $\vec{H}'$ of width $\tau = 1$.

The base cases for $|\mathcal{S}_p| = 1$ and $|\mathcal{S}_p| = 2$ are both trivial: for $|\mathcal{S}_p| = 1$ we can put the only vertex of $\mathcal{S}_p$ in its own bag and it will be a valid partial hub-tree decomposition. For $|\mathcal{S}_p| = 2$ we can put both vertices in separate bags and connect them by an edge, obtaining again a valid partial hub-tree decomposition.

For the inductive step we assume that it is possible to build a partial hub-tree decomposition with hub-treewidth one for any subset $\mathcal{S}_p \subset \mathcal{S}$ where $|\mathcal{S}_p| \leq r$, and $1 \leq |r| < |\mathcal{S}|$. We show how to construct a partial hub-tree decomposition with $\tau = 1$ for any subset of $\mathcal{S}$ of size $r + 1$:

Let $\mathcal{S}_{r+1} \subseteq \mathcal{S}$ be any subset of size $r + 1$. Let $x \in \mathcal{S}_{r+1}$ be an arbitrary vertex of $\mathcal{S}_{r+1}$. By the induction hypothesis we can construct a partial hub-tree decomposition with $\tau = 1$ for $\mathcal{S}_{-x} = \mathcal{S}_{r+1} \setminus \{x\}$. We denote such hub-tree decomposition with $\mathcal{T}_{-x}$. We can then show that $(x, \mathcal{T}_{-x})$ forms a good-pair, this is given by the following claim which is equivalent to Claim 4.10 of [11]:

CLAIM 9.4. *There exists a vertex $x \in \mathcal{S}_{r+1}$ and a width one partial hub-tree decomposition $\mathcal{T}_{-x}$ for $\mathcal{S}_{-x} = \mathcal{S}_{r+1} \setminus \{x\}$ such that $(x, \mathcal{T}_{-x})$ is a good-pair.*

Proof. We prove by contradiction. Assume that the claim is false, consider the unique reachability graph on the vertex set $\mathcal{S}_{r+1}$, $UR_{\mathcal{S}_{r+1}}$. Let $x \in \mathcal{S}_{r+1}$ be an arbitrary vertex from $\mathcal{S}_{r+1}$. By the assumption we have that $(x, \mathcal{T}_{-x})$ is not a good-pair. Hence, for each leaf node $l \in \mathcal{T}_{-x}$ connected to the vertex d we get that d is not an intersection-cover for x and l. Hence there exist a vertex v that is reachable by x and l but not d. But also, because d is the only vertex connected to l in $\mathcal{T}_{-x}$ we have that d is a $\mathcal{S}_{-x}$-cover for l and the only vertex that can reach v in $\mathcal{S}_{-x}$ is l. Thus, the edge $\{x, l\}$ will be in $UR_{\mathcal{S}_{r+1}}$.

Because $\mathcal{T}_{-x}$ has at least two leaves we will have that the degree of x in $UR_{\mathcal{S}_{r+1}}$ must be at least 2. This is true for every vertex in $x \in \mathcal{S}_{r+1}$. This implies that there is a cycle in $UR_{S_{r+1}}$ of length at least 3, using Lemma 9.1 this means that $LICL(H) \geq 3(t + 1)$, but we had that $LICL(H) < 3(t + 1)$, hence reaching a contradiction. ∎

Now, we show that if we have a good-pair $(x, \mathcal{T}_{-x})$ we can construct a hub-tree decomposition of hub-tree decomposition one, again [11] proved a more restrictive statement that we will generalize:

CLAIM 9.5. (EQUIVALENT TO CLAIM 4.9 FROM [11]) *Let $x \in \mathcal{S}_{r+1}$ and $\mathcal{T}_{-x}$ be a width one partial hub-tree decomposition for $\mathcal{S}_{-x} = \mathcal{S}_{r+1} \setminus \{x\}$ such that $(x, \mathcal{T}_{-x})$ is a good-pair. Then, there exists a partial hub-tree decomposition $\mathcal{T}$ for $\mathcal{S}_{r+1}$ with $\tau(\mathcal{T}) = 1$.*

Proof. For $(x, \mathcal{T}_{-x})$ to form a good-pair we must have that there is a leaf l in $\mathcal{T}_{-x}$ connected to a node $d \in \mathcal{T}_{-x}$ such that d is an intersection-cover for x and l. From the assumption of the inductive step we can construct a

hub-tree decomposition of width one for $\mathcal{S}_{r+1} \setminus l$ and connect l as a leaf to the node d. Let $\mathcal{T}$ be the resultant tree. We can show that $\mathcal{T}$ is a valid hub-tree decomposition of $\mathcal{S}_{r+1}$ of width 1.

We had that d is intersection-cover of l and x, also because l only connects to d in $\mathcal{T}_{-x}$ we have that d is a $\mathcal{S}_{-x}$-cover of l. Hence d is a $\mathcal{S}_{r+1}$-cover of l.

By Lemma 9.2 we have that $\mathcal{T}$ is a valid partial hub-tree decomposition of $\mathcal{S}_{r+1}$ with hub-treewidth one. □

Hence, combining both claims we get that we can construct a hub-tree decomposition with hub-treewidth of 1 for $\mathcal{S}_{r+1}$. This proves the induction and subsequently the lemma. □

10 Generalizing Bressan's algorithm

In this section we prove Lemma 5.5. This will complete the proof of the upper bound of our Main Theorem, as shown in §5. We will show how to adapt Bressan's Algorithm to compute the homomorphisms of the fraternal extensions. This requires working with non-acyclical graphs using the hubset and the hub-tree decomposition instead of the DAG-tree decomposition, and using graphs that are weighted and labeled. We start by restating the main lemma of this section:

LEMMA 5.5. *Let $\vec{G}$ be a directed weighted and labeled graph with n vertices and bounded outdegree and let $\vec{H}$ be a directed weighted and labeled graph with $\tau(H) = 1$. There exists an algorithm that computes $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in $O(n \log n)$ time.*

Given directed graphs $\vec{H}$ and $\vec{G}$. Lemma 4 in [15] shows a way of computing homomorphisms for the subgraphs induced by $Reach_{\vec{H}}(s)$ for every source $s \in S(\vec{H})$. We can generalize this result to directed weighted and labeled graphs:

LEMMA 10.1. *Let $\vec{H}$ be a directed weighted and labeled graphs with k vertices and hubset $\mathcal{S} = \mathcal{S}(\vec{H})$. Let $\vec{G}$ be a directed weighted graph with max outdegree $d = \Delta^+(\vec{G})$. For any vertex $s \in \mathcal{S}$, the set of homomorphisms from $\vec{H}(s)$ to $\vec{G}$ has size $O(d^{k-1}n)$ and can be enumerated in time $O(k^2 d^{k-1} n)$.*

Proof. Let $\vec{T}$ be a directed spanning tree of $\vec{H}(s)$ rooted at s. Let O be any arbitrary ordering of the vertices of $\vec{H}(s)$ such that all the edges of T are not inverted. For every vertex $u \in \vec{H}(s)$ following that ordering, we can enumerate all the possible candidates of $\vec{G}$ for the mapping $\phi(u)$. The first vertex s will have n candidates, as it can be assigned to every vertex in $\vec{G}$. However for all the other vertices, because they have at least one incoming edge from a vertex already assigned, we just need to look at the out-neighbors of the corresponding mapped vertex in $\vec{G}$, there will be then at most d candidates, as that is the maximum outdegree in $\vec{G}$. Hence the total number of possible homomorphisms is bounded by $O(nd^{k-1})$. We can list all these candidate homomorphisms in a similar amount of time. Only left is to verify if each candidate homomorphisms ϕ is valid:

- For each vertex $u \in V_{\vec{H}(s)}$, verify that they are mapped to a vertex with the same label $L_{\vec{H}}(u) = L_{\vec{G}}(\phi(u))$.
- For each edge $(u, v) \in E_{\vec{H}(s)}$, verify that $(\phi(u), \phi(v)) \in E_{\vec{G}}$ and $W_{\vec{H}}((u, v)) \geq W_{\vec{G}}((u, v))$.

This can be done in $O(k^2)$ time as we will have at most k vertices and k^2 edges, and every check can be done in constant time. Hence the total time required will be $O(k^2 d^{k-1} n)$. □

Note that in the case that the graph $\vec{G}$ has bounded outdegree and $\vec{H}$ is constant sized we will be able to compute $\mathrm{Hom}_{\vec{H}(s)}(\vec{G})$ in $O(n)$ time for all the vertices $s \in \mathcal{S}(\vec{H})$.

Given a hub-tree decomposition $\mathcal{T}$ of $\vec{H}$, we will use $down(B)$ to denote the down-closure of B in $\mathcal{T}$, that is, the union of all the bags $B \in \mathcal{B}$ that are descendants of B. We will then use $\vec{H}(down(B))$ to refer to the union of all the graphs $\vec{H}(B)$ for $B \in down(B)$.

If $\vec{H}$ has $\tau(H) = 1$ we can used a modification of the algorithm presented by Bressan in [15] to compute $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in linear time. Given a hub-tree decomposition decomposition $\mathcal{T}$ of $\vec{H}$, this algorithm uses dynamic programming to compute $\mathrm{Hom}_{\vec{H}(down(s))}(\vec{G})$ for any vertex $s \in \mathcal{S}$ aggregating the values of $\mathrm{Hom}_{\vec{H}(down(s'))}(\vec{G})$ of all the descendants s' of s and $\mathrm{Hom}_{\vec{H}(s)}(\vec{G})$.

Given a homomorphism ϕ we say that ϕ' respects ϕ if for every value u that ϕ takes $\phi(u) = \phi'(u)$. Given a homomorphism ϕ that maps the vertices in the set V, we call the restriction of ϕ to $V' \subseteq V$ to the map ϕ' that maps the vertices of V' with $\phi'(v) = \phi(v) \forall v \in V'$. Additionally we denote with $ext(\vec{H}, \vec{G}, \phi)$ to the number of homomorphisms ϕ' from $\vec{H}$ to $\vec{G}$ that respects ϕ. We can show the following lemma which is a generalization of Lemma 3 in [15]:

LEMMA 10.2. *Let $\mathcal{T}$ be a hub-tree decomposition of a graph $\vec{H}$ and let $B_1, \ldots, B_l$ be the children of B in $\mathcal{T}$. Fix $\phi_B : \vec{H}(B) \to \vec{G}$. Let $\Phi(\phi_B) = \{\phi : \vec{H}(down(B)) \to \vec{G} | \phi \text{ respects } \phi_B)\}$, and for $i = 1, \ldots, l$ let $\Phi_i(\phi_B) = \{\phi : \vec{H}(down(B_i)) \to \vec{G} | \phi \text{ respects } \phi_B)\}$. Then there exists a bijection between $\Phi(\phi_B)$ and $\Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$, and therefore:*

$$ext(\vec{H}(down(B)), \vec{G}, \phi_B) = \prod_{i=1}^{l} ext(\vec{H}(down(B_i)), \vec{G}, \phi_B)$$

Proof. The proof is similar to the proof of Lemma 3 in [15]:

First, we show that there exists an injection from $\Phi(\phi_B)$ to $\Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$: Fix any $\phi \in \Phi(\phi_B)$, and let ϕ_i be the restriction of ϕ to $\vec{H}(down(B_i))$, because ϕ respects ϕ_B so will ϕ_i, hence $\phi_i \in \Phi_i(\phi_B)$, therefore the tuple $(\phi_1, \ldots, phi_l) \in \Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$.

Now we show the opposite, that there exists an injection from $\Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$ to $\Phi(\phi_B)$: Fix any tuple $(\phi_1, \ldots, \phi_l) \in \Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$, note that the different ϕ_i of the tuple only intersect in $\vec{H}(B)$, and they all respect ϕ_B, hence we can combine $\phi_B, \phi_1, \ldots, \phi_l$ and obtain a homomorphism ϕ from $\vec{H}(down(B_i))$ to $\vec{G}$ such that ϕ respects ϕ_B, hence we will have that $\phi \in \Phi(\phi_B)$.

∎

We now adapt Bressan's algorithm [15] to our setting, the full algorithm can be seen in Alg. 4. We prove its correctness and runtime in the following Lemma:

LEMMA 10.3. *Let $\vec{H}$ be a labeled weighted and directed graph with a hub-tree decomposition $\mathcal{T}$ such that $\tau(\mathcal{T}) = 1$, let B be any node of $\mathcal{T}$ and let $\vec{H}$ be a labeled weighted and directed graph with bounded outdegree. Alg. 4 returns a dictionary C_B such that for every homomorphism $\phi \in \Phi(\vec{H}(B), \vec{G})$ we have $C_B(\phi) = ext(\vec{H}(down(B)), \vec{G}, \phi)$, and runs in $O(n \cdot \log n)$ time.*

Proof. First we prove the correctness of the algorithm. We can see that in the base case, when B is a leaf of $\mathcal{T}$, C_B will contain 1 for every $\phi_B \in \Phi(H(B), G)$.

If B is not a leaf, we assume that the algorithm returns the desired value for every child B_i of B. In this case the value of AGG_{B_i} after the first for loop will be $|\phi \in \Phi(\vec{H}(B_i), \vec{G}) : \phi \text{ respects } \phi_r| = ext(\vec{H}(down(B_i)), \vec{G}, \phi_r)$. Hence we will have that:

$$C_B(\phi) = \prod_{i=1}^{l} AGG_{B_i}(\phi_i) = \prod_{i=1}^{l} ext(\vec{H}(down(B_i)), \vec{G}, \phi_i)$$

$$= \prod_{i=1}^{l} ext(\vec{H}(down(B_i)), \vec{G}, \phi) = ext(\vec{H}(down(B)), \vec{G}, \phi)$$

Where the last inequality comes from Lemma 10.2.

For the runtime, we have that B has at most $O(k)$ children, from Lemma 10.1 we have that every dictionary C_{B_i} will have at most $O(n)$ keys, and we can enumerate them in $O(n)$ time. We will need $O(n \log n)$ time to access the dictionary, so the total complexity is $O(n \cdot \log n)$. ∎

Algorithm 4 Generalized Bressan's Algorithm: Homomorphisms($\vec{H}, \vec{G}, B$)

Input:
-Directed weighted labeled graph $\vec{H}$ with hub-tree decomposition $\mathcal{T}$
-Directed weighted labeled graph $\vec{G}$
-A node $B \in \mathcal{T}$
Output:
-Dictionary C_B

Let C_B be an empty dictionary with default value 0.
if B is a leaf **then**
  for every homomorphism $\phi_B : \vec{H}(B) \to \vec{G}$ **do**
    $C_B(\phi_B) = 1$
  end for
else
  let $B_1, \ldots, B_l$ be the children of B in $\mathcal{T}$
  for $i = 1, \ldots, l$ **do**
    C_{B_i} =Generalized Bressan's Algorithm($\vec{H}, \vec{G}, B_i$)
    Let AGG_{B_i} be an empty dictionary with default value 0.
    for every key ϕ in C_{B_i} **do**
      let ϕ_r be the restriction of ϕ to $Reach_{\vec{H}}(B) \cap Reach_{\vec{H}}(down(B_i))$
      $AGG_{B_i}(\phi_r) += C_{B_i}(\phi)$
    end for
  end for
  for every homomorphism $\phi : \vec{H}(B) \to \vec{G}$ **do**
    Let ϕ_i be the restriction of ϕ to $Reach_{\vec{H}}(B) \cap Reach_{\vec{H}}(down(B_i))$, for $i = 1, \ldots, l$.
    $C_B(\phi) = \prod_{i=1}^{l} AGG_{B_i}(\phi_i)$
  end for
end if
return C_B

We can finally prove the main lemma of this section:

Proof. (Lemma 5.5) We can compute a hub-tree decomposition $\mathcal{T}$ for $\vec{H}$ in $f(k)$ time for some function f and then run Alg. 4 in the root s of $\mathcal{T}$ to obtain C_s. From Lemma 10.3 we have that this takes $O(n \cdot \log n)$. We can then sum all the values of C_s to obtain $\mathrm{Hom}_{\vec{G}}(\vec{H})$, this takes additional $O(n)$ time.

We prove the correctness of this approach: Because s is the root of $\mathcal{T}$ we will have that $\vec{H}(down(s)) = \vec{H}$, hence $C_s(\phi) = ext(\vec{H}, \vec{G}, \phi)$ for all $\phi \in \Phi(\vec{H}(s), \vec{G})$. Summing over all ϕ we have that:

$$\sum_{\phi \in \Phi(\vec{H}(s), \vec{G})} C_s(\phi) = \sum_{\phi \in \Phi(\vec{H}(s), \vec{G})} ext(\vec{H}, \vec{G}, \phi) = \mathrm{Hom}_{\vec{H}}(\vec{G})$$

□

11 Lower Bound

In this section we prove the lower bound of the main theorem, given by the following theorem:

THEOREM 11.1. *For all $t > 0 \in \mathbb{N}$, let G be an input graph with n vertices, m edges and bounded $\nabla_{(t-1)/2}(G)$ and let H be a pattern graph on k vertices with $LICL(H) \geq 3(t+1)$. Assuming the Triangle Detection Conjecture, for any $\epsilon < \gamma$ there is no (expected) $O(m^{1+\epsilon})$ algorithm for the $\mathrm{Hom}_H(G)$ problem, where γ is the constant of the Triangle Detection Conjecture.*

For $t = 1$ the former theorem is proved to be true [11] as bounded ∇_0 is equivalent to bounded degeneracy. We will prove that the theorem holds for all the $t > 1$.

In order to do so we first show that counting a pattern in a bounded grad class is as hard as counting any subgraph of such pattern, this was showed to be true by [9] in the case of bounded degeneracy graphs. The proof uses some techniques introduced by [21]. We will extend such proof for all bounded grad classes.

Then we will show a simple reduction inspired by [10] that allows to relate counting triangles in a general graph (a problem which can not be done in linear time if the Triangle Detection Conjecture is true) with counting non-induced cycles in bounded grad classes. We then finalize the proof by extending te result to homomorphism counts of cycles.

11.1 Reducing to Cycle Homomorphisms In this subsection we show that computing homomorphisms of a pattern is as easy as computing all homomorphisms of all the induced subgraphs of that pattern. The proof follows closely the proof from Lemma 1.7 in [9], but generalizing to graphs with bounded ∇_k.

LEMMA 11.1. *Let $\mathcal{G}_i$ be the class of bounded ∇_i graphs, for some i. Let H be a pattern graph, if computing $\mathrm{Hom}_H(G)$ for any $G \in \mathcal{G}_i$ is easy, then so is computing $\mathrm{Hom}_{H'}(G)$ for every induced subgraph H' of H.*

We prove this lemma by proving the more general lemma that follows, which is a generalization of Lemma 4.1 from [9]:

LEMMA 11.2. *For every graph H there is $k = k(H)$ such that the following hols. For every graph G there are graphs $G_1, \ldots, G_k$, computable in time $O(|V_G| + |E_G|)$, such that $|V_{G_i}| = O(|V_G|)$ and $|E_{G_i}| = O(|E_G|)$ for every $i = 1, \ldots, k$ such that knowing $\mathrm{Hom}_H(G_1), \ldots, \mathrm{Hom}_H(G_k)$ allows one to compute $\mathrm{Hom}_{H'}(G)$ for all induced subgraphs H' of H in constant time. Furthermore, if G has bounded ∇_i, then so do $G_1, \ldots, G_k$.*

In order to prove this lemma, we first need to introduce another additional lemma, which again is a generalization of a lemma from [9], in this case Lemma 4.2:

LEMMA 11.3. *Let $H_1, \ldots, H_k$ be pairwise non-isomorphic graphs and let $c_1, \ldots, c_k$ be non-zero constants. For every graph G there are graphs $G_1, \ldots, G_k$, computable in time $O(|V_G| + |E_G|)$ such that $|V_{G_i}| = O(|V_G|)$ and $|E_{G_i}| = O(|E_G|)$ for every $i = 1, \ldots, k$, and such that knowing $b_j := c_1 \cdot \mathrm{Hom}_{H_1}(G_j) + \ldots + c_k \cdot \mathrm{Hom}_{H_k}(G_j)$ for every $j = 1, \ldots, k$ allows one to compute $\mathrm{Hom}_{H_1}(G), \ldots, \mathrm{Hom}_{H_k}(G)$ in constant time. Furthermore, if G has bounded ∇_i, then so do $G_1, \ldots, G_k$.*

Proof. We start the proof by stating the following lemma from [33] and [44], which was stated in [9] as Lemma A.2:

LEMMA 11.4. *Let $H_1, \ldots, H_k$ be pairwise non-isomorphic graphs, and let $c_1, \ldots, c_k \neq 0$ be non-zero constants. Then there exist graphs $F_1, \ldots, F_k$ such that the $k \times k$ matrix $M_{i,j} = c_j \cdot \mathrm{Hom}_{H_j}(F_i), 1 \leq i, j \leq k$, is invertible.*

Now, let $G_i = F_i \times G$, we first show that if G has bounded ∇_i, then so do $G_1, \ldots, G_k$: The proof is very similar to Lemma 6.1. Note that the size of F_i does not depends on the input graph G, only on the graphs $H_1, \ldots, H_k$. Because we assume such graphs to be constant-sized so will be $F_1, \ldots, F_k$. Let f be the number of vertices in F_i, we have:

$$G_i = F_i \times G \subseteq G \times K_f = G \bullet \bar{K}_f \subseteq G \bullet K_f$$

Where the second equality comes from Fact 4.4. Then using Prop. 6.1 we will have that if $\nabla_i(G)$ is bounded so will $\nabla_i(G \bullet K_f)$, and therefore by the previous equation so will $\nabla_i(G_i)$.

Now, let $b_i = \sum_{j=1}^{k} c_1 \cdot \mathrm{Hom}_{H_1}(G_j) + \ldots + c_k \cdot \mathrm{Hom}_{H_k}(G_j)$, we can rewrite it as:

$$b_i = \sum_{j=1}^{k} c_j \cdot \mathrm{Hom}_{H_j}(F_i \times G) = \sum_{j=1}^{k} c_j \cdot \mathrm{Hom}_{H_j}(F_i) \cdot \mathrm{Hom}_{H_j}(G) = \sum_{j=1}^{k} M_{i,j} \cdot \mathrm{Hom}_{H_j}(G)$$

Hence for $1 \leq i \leq k$ we obtain a system of linear equations with $\mathrm{Hom}_{H_1}(G), \ldots, \mathrm{Hom}_{H_k}(G)$ as variables and M as the matrix of the system. By Lemma 11.4 M is invertible, then given $b_1, \ldots, b_k$ we can compute $\mathrm{Hom}_{H_1}(G), \ldots, \mathrm{Hom}_{H_k}(G)$ in constant time. ∎

We can now complete the proof by proving Lemma 11.2:

Proof. (Lemma 11.2)

The proof of this lemma comes directly from Lemma 4.1 in [9]. The only difference is showing that the graphs $G_1, \ldots, G_k$ have bounded grad, instead of bounded degeneracy, when G does. We can see that this comes directly from Lemma 11.3, which generalizes Lemma 4.2 in [9]. ∎

11.2 From counting triangles to counting cycles In this subsection we prove a hardness result for non-induced copies in the case that the pattern is the cycle graph. We use a reduction very similar to the one found in [10]: we can take any graph G and replace every edge by some combination of paths. The resultant graphs will actually have bounded grad for certain depth, depending on the length of the path.

We will prove the following:

LEMMA 11.5. *For all $t > 1 \in \mathbb{N}$, let G be any input graph with n vertices, m edges and bounded $\nabla_{(t-1)/2}(G)$ and let $H = \mathcal{C}_k$ be the cycle graph on k vertices for $k \in \{3(t+1), 3(t+1)+1\}$. Assuming the Triangle Detection Conjecture, for any $\epsilon < \gamma$ there is no (expected) $O(m^{1+\epsilon})$ algorithm for the $\mathrm{Sub}_H(G)$ problem, where γ is the constant of the Triangle Detection Conjecture.*

Proof. Fix any $t > 1$. We will first show a reduction for cycles of length $k = 3(t+1)$.

Let G be a graph. We define the graph G_t by replacing every edge in G by a path of $t+1$ edges, formally:

DEFINITION 11.1. (G_t) *Let $G = (V, E)$ be an arbitrary input graph. We define the reduced graph $G_t = (V_t, E_t)$ as follows:*

- *For each vertex $v \in V$ we create a vertex $v \in V^o$.*
- *For each edge $e = (u, w) \in E$ we create t extra vertices $v_{e,1}, \ldots, v_{e,t}$ in V^*.*
- *We define $V_t = V^o \cup V^*$.*
- *We create the edge set E_t by adding the edges $(u, v_{e,1}), (v_{e,1}, v_{e_2}), \ldots, (v_{e,t-1}, v_{e,t}), (v_{e,t}, w)$ for every edge $(u, v) \in E$.*

Fig. 11 shows how the reduction replaces an edge by the path in G_t. We can show that there exists a relation between the number of triangles in G and the number of $\mathcal{C}_k$ cycles in G_t:

CLAIM 11.1. *Let $t > 1$ and G any graph, set $k = 3(t+1)$, there is a triangle in G if and only if there is a $\mathcal{C}_k$ cycle in G_t.*

Proof. Consider any triangle v_1, v_2, v_3 in G, in the graph G_t each pair of those vertices will be separated by a path of length $t+1$, hence combining those paths we obtain a cycle of length $3(t+1) = k$ in G_t.

Conversely, let $\mathcal{C} = \mathcal{C}_k$ be a k-cycle in G_t, we can show that $\mathcal{C}$ must contain exactly 3 vertices in V^o: If it contained 1 or 2 then $\mathcal{C}$ would not be able to be a cycle, while if it contains 4 or more then it will form a cycle of at least $4(t+1)$ vertices, which is greater than k. Take the three vertices in V^o, they must be connected to each other by a path of $t+1$ edges in G_t and hence by edges in G, therefore, they will form a triangle. ∎

Similarly we define a reduction for the cycles of length $k = 3(t+1)+1$. In this case we will replace every edge of G by two different paths:

DEFINITION 11.2. $(G_{t'})$ *Let $G = (V, E)$ be an arbitrary input graph. We define the reduced graph $G_{t'} = (V_{t'}, E_{t'})$ as follows:*

- *For each vertex $v \in V$ we create a vertex $v \in V^o$.*
- *For each edge $e = (u, w) \in E$ we create t extra vertices $v_{e,1}, \ldots, v_{e,t}$ in V^* and another additional $t+1$ vertices $v'_{e,1}, \ldots, v'_{e,t+1}$ in V^**
- *We define $V_t = V^o \cup V^*$.*
- *We create the edge set E_t by adding the edges $(u, v_{e,1}), (v_{e,1}, v_{e_2}), \ldots, (v_{e,t-1}, v_{e,t}), (v_{e,t}, w)$ and $(u, v'_{e,1}), (v'_{e,1}, v'_{e_2}), \ldots, (v'_{e,t}, v'_{e,t+1}), (v'_{e,t+1}, w)$ for every edge $(u, v) \in E$.*

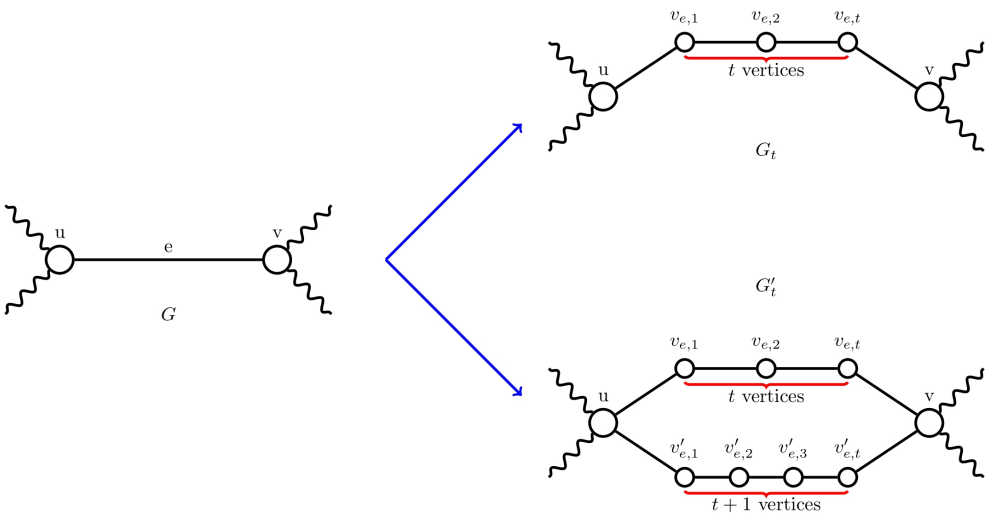

Figure 11: An example of how an edge $e = (u, v)$ in G is replaced in the reduced graphs G_t and $G_{t'}$. In G_t we will add t vertices between u and v forming a path. While in $G_{t'}$ we will add two path, one with t vertices and another with $t + 1$ vertices. This process will be applied to every edge in G.

In Fig. 11 we show how each edge of G is replaced in $G_{t'}$. Again we can show that there is a relation between the number of triangles in G and the number of $\mathcal{C}_k$ cycles in $G_{t'}$:

CLAIM 11.2. *Let $t > 1$ and G any graph, set $k = 3(t+1)+1$, there is a triangle in G if and only if there is a $\mathcal{C}_k$ cycle in $G_{t'}$.*

Proof. Consider any triangle v_1, v_2, v_3 in G, in the graph $G_{t'}$ each pair of those vertices will be separated by a path of length $t+1$ and a path of length $t+2$, hence combining those paths we can obtain three different cycles of length $3(t+1)+1 = k$ in $G_{t'}$.

Conversely, let $\mathcal{C} = \mathcal{C}_k$ be a k-cycle in $G_{t'}$, we can show that $\mathcal{C}$ must contain exactly 3 vertices in V^o: If it contained 1 or 2 then $\mathcal{C}$ could only be a cycle of length $2t+3$ which is strictly less than $3(t+1)+1$ for $t > 1$, while if it contains 4 or more then it will form a cycle of at least $4(t+1)$ vertices, which is greater than $3(t+1)+1 = k$. Take the three vertices in V^o, they must be connected to each other by either a path of $t+1$ or $t+2$ edges in $G_{t'}$ and hence by edges in G, therefore, they will form a triangle. ∎

We also show that both G_t and $G_{t'}$ have bounded $\nabla_{(t-1)/2}$:

CLAIM 11.3. *Let $t > 1$ and G be an arbitrary graph, G_t and $G_{t'}$ have bounded $\nabla_{(t-1)/2}$.*

Proof. Let $G' = (V', E')$ be any shallow topological minor of G_t or $G_{t'}$ at depth $(t-1)/2$. That is, a graph where the vertices are a subset of the vertices of G_t or $G_{t'}$ and the edges correspond to disjoint paths in G_t or $G_{t'}$ of length at most t.

The vertices of G' can either be part of $V'^* = V^* \cap V'$ or $V'^o = V^o \cap V'$. The degree of the vertices in V'^* can not be greater than 2 as the original degree of such vertices in either G_t or $G_{t'}$ were 2. Additionally, any edge in G' can not have both of its endpoints in V'^o, as the minimum distance between such vertices is $t+1$. Therefore, every edge will have at least one end in V'^*, because the degree of such vertices is at most 2, we will have at most $2|V'^*|$ edges.

We can then bound the average edge density of G':

$$\frac{|E'|}{|V'|} \leq \frac{2|V'^*|}{|V'^*| + |V'^o|} \leq \frac{2|V'^*|}{|V'^*|} = 2$$

Hence every topological minor of G_t or $G_{t'}$ at depth $(t-1)/2$ has bounded average edge density and $\tilde{\nabla}_{(t-1)/2}(G_t)$ and $\tilde{\nabla}_{(t-1)/2}(G_{t'})$ are bounded. Which by Fact 4.3 implies that $\nabla_{(t-1)/2}(G_t)$ and $\nabla_{(t-1)/2}(G_{t'})$ are also bounded. ∎

Now, assume that we have an algorithm that can count the number of cycles of size $k = 3(t+1)$ in graphs of bounded $\nabla_{(t-1)/2}$ in time $O(m^{1+\varepsilon})$ for some $\varepsilon < \gamma$, then given a graph G we could construct G_t and obtain $\text{Sub}_{\mathcal{C}_k}(G')$ in time $O(m^{1+\varepsilon})$, as from Claim 11.3 we have that G_t has bounded $\nabla_{(t-1)/2}$. We can then use Claim 11.1 to determine if G contains a triangle. However, this directly contradicts the Triangle Detection Conjecture and hence not such algorithm can exist.

Similarly, assume that we have an algorithm that can count the number of cycles of size $k = 3(t+1)+1$ in graphs of bounded $\nabla_{(t-1)/2}$ in time $O(m^{1+\varepsilon})$ (again for some $\varepsilon < \gamma$), then given a graph G we could construct $G_{t'}$ and obtain $\text{Sub}_{\mathcal{C}_k}(G')$ in time $O(m^{1+\varepsilon})$, as from Claim 11.3 we have that G' has bounded $\nabla_{(t-1)/2}$. We can then use Claim 11.2 to determine if G contains a triangle. Again, this directly contradicts the Triangle Detection Conjecture and hence not such algorithm can exists. ∎

11.3 From Cycle Subgraphs to Homomorphisms Now we extend the hardness result from counting subgraphs to counting homomorphisms of cycle graphs. It is given by the following lemma:

LEMMA 11.6. *For all $t > 1 \in \mathbb{N}$, let G be any input graph with n vertices, m edges and bounded $\nabla_{(t-1)/2}(G)$ and let $H = \mathcal{C}_k$ be the cycle graph on k vertices for $k \in \{3(t+1), 3(t+1)+1, 3(t+1)+2\}$. Assuming the Triangle Detection Conjecture, for any $\epsilon < \gamma$ there is no (expected) $O(m^{1+\epsilon})$ algorithm for the $\text{Hom}_H(G)$ problem, where γ is the constant of the Triangle Detection Conjecture.*

Proof. Fix any $t > 1$. We prove each of the cases separately:

- Let $k = 3(t+1)$, let $H = \mathcal{C}_k$ be the cycle with k vertices and G any graph with bounded $\nabla_{(t-1)/2}(G)$. Let H' be any graph in the *Spasm* of H different than H. We have that $LICL(H') < 3(t+1)-1$, hence by Theorem 5.1 we can compute $\text{Hom}_{H'}(G)$ in $O(n)$ time. Now, assume that we can compute $\text{Hom}_H(G)$ in $O(m^{1+\epsilon})$ time for some $\varepsilon < \gamma$, then we could use Lemma 4.1 to obtain $\text{Sub}_H(G)$, but that contradicts Lemma 11.5, and hence no $O(m^{1+\epsilon})$ algorithm exists.

- Similarly, let $k = 3(t+1)+1$, let $H = \mathcal{C}_k$ be the cycle with k vertices and G any graph with bounded $\nabla_{(t-1)/2}(G)$. Let H' be any graph in the *Spasm* of H different than H. We have that $LICL(H') < 3(t+1)$, hence by Theorem 5.1 we can compute $\text{Hom}_{H'}(G)$ in $O(n)$ time. Now, assume that we can compute $\text{Hom}_H(G)$ in $O(m^{1+\epsilon})$ time for some $\varepsilon < \gamma$, then we could use Lemma 4.1 to obtain $\text{Sub}_H(G)$, but that contradicts Lemma 11.5, and hence no $O(m^{1+\epsilon})$ algorithm exists.

- Finally, let $k = 3(t+1)+2$, let $H = \mathcal{C}_k$ be the cycle with k vertices and G any graph. Consider the reduced graph G_t, remember that from Claim 11.3 we have that G_t has bounded $\nabla_{(t-1)/2}(G)$. G_t can not contain any cycle of length exactly k, as every cycle has a multiple of $(t+1)$ edges, hence $\text{Sub}_H(G_t) = 0$. Consider the Spasm of H, apart from $\mathcal{C}_k$ itself the only one other pattern in $Spasm(H)$ with $LICL \geq 3(t+1)$ will be the cycle $\mathcal{C}_{k-2}$ with a tail, let H^* be such pattern.

  For any other pattern $H' \in Spasm(H) \setminus \{H, H^*\}$ we have that $LICL(H') < 3(t+1)$ and hence by Theorem 5.1 we can compute $\text{Hom}_{H'}(G)$ in $O(n)$ time. Now assume there is an $O(m^{1+\varepsilon})$ algorithm that allows us to compute $\text{Hom}_H(G)$ for some $\varepsilon < \gamma$. Then we could use Lemma 4.1 to obtain the value of $\text{Hom}_{H^*}(G)$ as all the other terms in the equation will be known. However, we just showed that counting homomorphisms of the $\mathcal{C}_{3(t+1)}$ cycle is not possible in linear time, and by Lemma 11.1 we will have that there is no algorithm for counting H^* as it is a supergraph of $\mathcal{C}_{3(t+1)}$.

∎

We can now complete the proof of the lower bound:

Proof. (Theorem 11.1)

First, for $t = 1$ the theorem is true. As the statement becomes equivalent to show that there is no algorithm for counting cycles of length greater than 6 in bounded degeneracy graphs(assuming Triangle Detection Conjecture), this was proved in [11].

Hence we just need to prove for $t > 1$. Note that suffices to show that there is no $o(m)$ algorithm for computing $\mathrm{Hom}_H(G)$ for graphs G of bounded $\nabla_{(t-1)/2}$ and graphs H with $LICL(H) \in [3(t+1), 3(t+2))$.

Fix some $t > 1$, and let H bet any graph with $LICL(H) \in [3(t+1), 3(t+2))$, note that H must be a supergraph of either $\mathcal{C}_{3(t+1)}, \mathcal{C}_{3(t+1)+1}$, or $\mathcal{C}_{3(t+1)+2}$. Now assume that there is an algorithm that computes $\mathrm{Hom}_H(G)$ in $O(m^{1+\varepsilon})$ time for graphs G with bounded $\nabla_{(t-1)/2}$, then using Lemma 11.1 we have that we can compute $\mathrm{Hom}_{H'}(G)$ for $H' \in \{\mathcal{C}_{3(t+1)}, \mathcal{C}_{3(t+1)+1}, \mathcal{C}_{3(t+1)+2}\}$, but this directly contradicts Lemma 11.6, completing the proof. ∎

12 From Homomorphism to non-induced copies

Proof. (Theorem 1.2)

We first prove the upper bound: Let G be any input graph with bounded $\nabla_{r/2}$ and H a graph with constant size k and $LICL(Spasm(H)) < 3(r+2)$. Consider any graph $H' \in Spasm(H)$, we have that $LICL(H') < 3(r+2)$, and hence by Theorem 5.1 there is an algorithm that computes $\mathrm{Hom}_{H'}(G)$ in time $f(\nabla_{r/2})O(m)$, for some explicit function f. The size of $Spasm(H)$ only depends on k, thus we can compute $\mathrm{Hom}_{H'}(G)$ for all the graphs $H' \in Spasm(H)$ in $f(\nabla_{r/2})O(m)$ time. Using Lemma 4.1 we have that we can compute $\mathrm{Sub}_H(G)$ as a linear combination of $\mathrm{Hom}_{H'}(G)$ for all the H' in the spasm of H, this will take additional constant time giving the upper bound result.

Now we prove the lower bound: Again let G be any input graph with bounded $\nabla_{r/2}$ and H a graph with constant size k and $LICL(Spasm(H)) \geq 3(r+2)$. This means that there exists a graph $H' \in Spasm(H)$ with $LICL(H') \geq 3(r+2)$. Assume by contradiction that there is a $O(m^{1+\varepsilon})$ algorithm that computes $\mathrm{Sub}_H(G)$ for some $\varepsilon < \gamma$. We can then use Lemma 11.3 to construct a series of graphs $G_1, \ldots, G_k$ that are also bounded $\nabla_{r/2}$. We can then compute $\mathrm{Sub}_H(G_i)$ for each of the graphs using the $O(m^{1+\varepsilon})$ algorithm that we are assuming exists.

Then, because $\mathrm{Sub}_H(G_i)$ is a linear combination of $\mathrm{Hom}_{H'}(G_i)$ for all the $H' \in Spasm(H)$ we can apply again Lemma 11.3 to compute $\mathrm{Hom}_{H'}(G)$ for all the $H' \in Spasm(H)$ in additional constant time. However, recall that there is a $H' \in Spasm(H)$ for which $LICL(H') \geq 3(r+2)$. By Theorem 11.1 we have that there is no $O(m^{1+\varepsilon})$ algorithm to compute $\mathrm{Hom}_{H'}(G)$. Hence, we reach a contradiction. ∎

References

[1] 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.

[2] 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.

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. 31st ACM Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

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

[5] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, 2019.

[6] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[7] Suman K Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *International Symposium on Theoretical Aspects of Computer Science*, 2017.

[8] Suman K Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *International Colloquium on Automata, Languages and Programming*, 2020.

[9] 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.

[10] 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.

[11] 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 23152332, 2021.

[12] Suman K Bera and C Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Principles of Database Systems*, pages 457–467, 2020.
[13] 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.
[14] 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.
[15] Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021.
[16] Marco Bressan, Matthias Lanzinger, and Marc Roth. The complexity of pattern counting in directed graphs, parameterised by the outdegree. In *Annual ACM Symposium on the Theory of Computing*, pages 542–552, 2023.
[17] Graham R Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of combinatorial theory, series B*, 77(2):221–262, 1999.
[18] 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.
[19] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing (SICOMP)*, 14(1):210–223, 1985.
[20] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009.
[21] 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.
[22] Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004.
[23] 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.
[24] Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Counting h-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002.
[25] Reinhard Diestel. *Graph Theory, Fourth Edition*. Springer, 2010.
[26] Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.
[27] Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.
[28] Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proc. 50th Annual ACM Symposium on the Theory of Computing*, pages 722–734, 2018.
[29] Talya Eden, Dana Ron, and C Seshadhri. Faster sublinear approximations of k-cliques for low arboricity graphs. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, 2020.
[30] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994.
[31] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 632640, USA, 1995. Society for Industrial and Applied Mathematics.
[32] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27:275–291, 2000.
[33] Paul Erdős, László Lovász, and Joel Spencer. Strong independence of graphcopy functions. *Graph theory and related topics*, pages 165–172, 1978.
[34] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing (SICOMP)*, 33(4):892–922, 2004.
[35] G. Goel and J. Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006.
[36] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 653–662, 1998.
[37] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
[38] 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.
[39] Madhav Jha, C Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 589–597, 2013.
[40] 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.
[41] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In

Proc. 39th International Colloquium on Automata, Languages and Programming, pages 598–609, 2012.
[42] Tamara G Kolda, Ali Pinar, Todd Plantenga, C Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM Journal on Scientific Computing*, 36(5):S48–S77, 2014.
[43] László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967.
[44] László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.
[45] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In *Proc. 19th Annual European Symposium on Algorithms*, pages 677–688, 2011.
[46] 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.
[47] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016.
[48] Jaroslav Neetil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.
[49] Jaroslav Neetil and Patrice Ossona de Mendez. Grad and classes with bounded expansion ii. algorithmic aspects. *European Journal of Combinatorics*, 29(3):777–791, 2008.
[50] Jaroslav Neetil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012.
[51] Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science*, 2(1), 2017.
[52] 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.
[53] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.
[54] 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.
[55] Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2161–2180, 2020.
[56] 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.
[57] 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.
[58] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, 2018.
[59] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World Wide Web*, pages 607–614, 2011.
[60] George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.