Subgraph Counting in Subquadratic Time for Bounded Degeneracy Graphs

Daniel Paul-Pena
University of California, Santa Cruz, United States

C. Seshadhri
University of California, Santa Cruz, United States

Abstract

We study the classic problem of subgraph counting, where we wish to determine the number of occurrences of a fixed pattern graph H in an input graph G of n vertices. Our focus is on *bounded degeneracy* inputs, a rich family of graph classes that also characterizes real-world massive networks. Building on the seminal techniques introduced by Chiba-Nishizeki (SICOMP 1985), a recent line of work has built subgraph counting algorithms for bounded degeneracy graphs. Assuming fine-grained complexity conjectures, there is a complete characterization of patterns H for which linear time subgraph counting is possible. For every $r \geq 6$, there exists an H with r vertices that cannot be counted in linear time.

In this paper, we initiate a study of subquadratic algorithms for subgraph counting on bounded degeneracy graphs. We prove that when H has at most 9 vertices, subgraph counting can be done in $\tilde{O}(n^{5/3})$ time. As a secondary result, we give improved algorithms for counting cycles of length at most 10. Previously, no subquadratic algorithms were known for the above problems on bounded degeneracy graphs.

Our main conceptual contribution is a framework that reduces subgraph counting in bounded degeneracy graphs to counting smaller hypergraphs in arbitrary graphs. We believe that our results will help build a general theory of subgraph counting for bounded degeneracy graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Homomorphism counting, Bounded degeneracy graphs, Fine-grained complexity, Subgraph counting.

Funding Both authors are supported by NSF CCF-1740850, CCF-1839317, CCF-2402572, and DMS-2023495.

1 Introduction

The fundamental algorithmic problem of subgraph counting in a large input graph has a long and rich history [37, 24, 28, 22, 38, 2, 20, 42, 49]. There are applications in logic, properties of graph products, partition functions in statistical physics, database theory, machine learning, and network science [18, 16, 26, 11, 42, 23, 41]. We are given a *pattern* graph $H = (V(H), E(H))$, and an *input* graph $G = (V(G), E(G))$. All graphs are assumed to be simple. We use $\mathrm{Sub}_H(G)$ to denote the problem of computing the number of (not necessarily induced) subgraphs of H in G, that is, the number of subgraphs of G isomorphic to H.

If the pattern is part of the input, this problem becomes $\mathbb{NP}$-hard, as it subsumes subgraph isomorphism. Often, one thinks of the pattern H as fixed, and running times are parameterized in terms of the properties (like the size) of H. Let us set $n = |V(G)|$ and $k = |V(H)|$. There is a trivial brute-force $O(n^k)$ algorithm, but we do not expect $O(n^{k-\varepsilon})$ algorithms for general H for some constant $\varepsilon > 0$ [22].

The rich field of subgraph counting focuses on restrictions on the pattern or the input, under which non-trivial algorithms and running times are possible [33, 4, 16, 25, 24, 22, 11, 20, 12, 47]. Given the practical importance of subgraph counting, there is a special focus on linear time (or small polynomial) running times.

Inspired by seminal work of Chiba-Nishizeki [19], a recent line of work has focused on building a theory of subgraph counting for *bounded degeneracy graphs* [12, 7, 8, 15, 6]. These are classes of graphs where all subgraphs have a constant average degree. This work culminated in results of Bera-Pashanasangi-Seshadhri and Bera-Gishboliner-Levanzov-Seshadhri-Shapira [8, 6]. We now have precise dichotomy theorems characterizing linear time subgraph counting in bounded degeneracy graph. When H has at most 5 vertices, then $\mathrm{Sub}_H(G)$ can be determined in linear time (if G has bounded degeneracy). For all $k \geq 6$, there is a pattern H on k vertices that cannot be counted in linear time, assuming fine-grained complexity conjectures. The following question is the next step from this line of work.

When can we get subquadratic algorithms for subgraph counting (when G has bounded degeneracy)? Are there non-trivial algorithms that work for all H with 6 *(or more) vertices?*

Before stating our main results, we offer some justification for this problem.

Bounded degeneracy graphs: This is an extremely rich family of graph classes, containing all minor-closed families, bounded expansion families, and preferential attachment graphs [48]. Most massive real-world graphs, like social networks, the Internet, communication networks, etc., have low degeneracy ([30, 34, 51, 5, 9], also Table 2 in [5]). The degeneracy has a special significance in the analysis of real-world graphs, since it is intimately tied to the technique of "core decompositions" [48]. Most of the state-of-the-art practical subgraph counting algorithms use algorithmic techniques for bounded degeneracy graphs [2, 35, 42, 40, 34, 41].

Subquadratic time: From a theoretical perspective, the orientation techniques of Chiba-Nishizeki and further results [19, 7, 8, 6] are designed for linear time algorithms. The primary technical tool is the use of DAG-Tree decompositions, introduced in a landmark result of Bressan [12, 13]. Bressan's algorithm yields running times of the form n^r, where $r \in \mathbb{N}$ and is the *DAG-treewidth* of H. It is known that linear time algorithms are possible iff the DAG-treewidth is one [6]. It is natural to ask if the DAG-treewidth being two is a natural barrier. *Subquadratic* algorithms would necessarily require a new technique, other than DAG-treewidth. It would also show situations where the DAG-treewidth can be beaten.

From a practical standpoint, the best exact subgraph counting codes (the ESCAPE package [1]) use methods similar to the above results. Algorithms for bounded degeneracy graphs have been remarkably successful in dealing with modern large networks. Subquadratic algorithms could provide new practical tools for subgraph counting.

The focus on 6 vertices and the importance of cycle patterns: The problem of counting all patterns of a fixed size is called *graphlet analysis* in bioinformatics and machine learning [43, 50]. Current scalable exact counting codes go to 5 vertex patterns [1], which is precisely the theoretical barrier seen in [7]. Part of our motivation is to understand the complexity of counting all 6 vertex patterns (in bounded degeneracy graphs).

The seminal cycle detection work of Alon-Yuster-Zwick also gave algorithms parameterized by the graph degeneracy [4]. But most of their results are for cycle *detection*, whereas counting is arguably the more relevant problem. It is natural to ask if counting is also feasible with similar running times.

1.1 Main Results

Our main result shows that patterns with at most 9 vertices can be counted in subquadratic time. Let n be the number of vertices and κ be the degeneracy of the input graph G. The degeneracy κ is the maximum value, over all subgraphs of G, of the minimum degree of the subgraph. In what follows, $f : \mathbb{N} \to \mathbb{N}$ denote some explicit function.

▶ **Theorem 1.1.** *(Main Theorem) There is an algorithm that computes*[1] $\mathrm{Sub}_H(G)$ *for all patterns H with at most 9 vertices in time $f(\kappa)\tilde{O}(n^{5/3})$.*[2]

Recall that previous works gave (near) linear time algorithms when H had at most 5 vertices [7]. The best subgraph counting algorithm for bounded degeneracy graphs is Bressan's algorithm, which runs in at least quadratic (if not cubic) time for many patterns with 6 to 9 vertices.

Additionally, we construct an explicit 10-vertex pattern that we conjecture cannot be counted in subquadratic time in the bounded degeneracy setting. We are able to relate the complexity of counting that pattern to counting a specific hypergraph in general graphs, which we believe can not be done in subquadratic time. (More in §1.2.6 and §F.)

1.1.1 Counting cycles

As a secondary result, we are able to show that cycle counting in bounded degeneracy graphs can be done even faster. These results are tight, in the sense that any improvement will imply beating long standing cycle detection algorithms for sparse graphs. Let $\mathcal{C}_k$ denote the k-cycle and d_k be the exponent (in terms of edges) of the fastest algorithm for k-cycle detection. Gishboliner-Levanzov-Shapira-Yuster (henceforth GLSY) recently showed that *homomorphism* counting of $\mathcal{C}_{2k}$ in bounded degeneracy graphs can be done in time $O(n^{d_k})$ [29]. We prove that this complexity can be matched for the problem of *subgraph* counting. Cycle counting is more challenging, since it involves compute linear combinations of homomorphisms of non-cyclic patterns (which requires other techniques).

▶ **Theorem 1.2.**
- *There is an algorithm that computes* $\mathrm{Sub}_{\mathcal{C}_6}(G)$ *and* $\mathrm{Sub}_{\mathcal{C}_7}(G)$ *in time $f(\kappa)\tilde{O}(n^{d_3}) \approx f(\kappa)\tilde{O}(n^{1.41})$. Additionally, no $f(\kappa)o(n^{d_3})$ algorithm exists unless there exists a $o(m^{d_3})$ algorithm for counting triangles.*
- *There is an algorithm that computes* $\mathrm{Sub}_{\mathcal{C}_8}(G)$ *and* $\mathrm{Sub}_{\mathcal{C}_9}(G)$ *in time $f(\kappa)\tilde{O}(n^{d_4}) \approx f(\kappa)\tilde{O}(n^{1.48})$. Additionally, no $f(\kappa)O(n^{d_4})$ algorithm exists unless there exists a $o(m^{d_4})$ algorithm for counting 4-cycles.*
- *There is an algorithm that computes* $\mathrm{Sub}_{\mathcal{C}_{10}}(G)$ *in time $f(\kappa)\tilde{O}(n^{d_5}) \approx f(\kappa)\tilde{O}(n^{1.63})$. Additionally, no $f(\kappa)o(n^{d_5})$ algorithm exists unless there exists a $o(m^{d_5})$ algorithm for counting 5-cycles.*

Previously, for bounded degeneracy graphs, subquadratic results were only known for detecting cycles, by a result of Alon, Yuster and Zwick [4]. Theorem 1.2 improves or matches their bounds in all cases, despite solving the harder problem of counting. The lower bounds relating to cycle counting in general graphs follow directly from the techniques of GLSY [29]. We note that the exponents d_k have not been improved for twenty years [4, 53]. As a side

[1] We can obtain a similar result for the problem of counting only induced subgraphs $\mathrm{IndSub}_H(G)$, as it can be expressed as a linear combination of $\mathrm{Sub}_{H'}(G)$ for some patterns H' with $V(H') = V(H)$.
[2] We express our results parameterizing by the degeneracy of the input graph G. Note that if G is from a class with bounded degeneracy, then $f(\kappa)$ is constant and we can ignore that term.

corollary of our methods, we also get a better algorithm for counting 5-cycles in arbitrary graphs (Corollary 1.7).

1.2 Main Ideas

The theorems above are obtained from a new reduction technique that converts homomorphism counting in bounded degeneracy graphs to subgraph counting in arbitrary graphs for some specific patterns.

The starting point for most subgraph counting algorithms for bounded degeneracy graphs is to use *graph orientations* [48]. A graph G has bounded degeneracy iff there exists an acyclic orientation $\vec{G}$ such that all vertices have bounded *outdegree*. (An acyclic orientation is obtained by directing the edges of G into a DAG.) Moreover, this orientation can be found in linear time [39]. To count H-subgraphs in G, we consider all possible orientations $\vec{H}$ of H and compute (the sum of) all $\mathrm{Sub}_{\vec{H}}(\vec{G})$.

The approach formalized by Bressan [13] and Bera-Pashanasangi-Seshadhri [7] is to break $\vec{H}$ into a collection of (out)directed trees rooted at the sources of $\vec{H}$. The copies of each tree in $\vec{G}$ can be enumerated in linear time, since outdegrees are bounded. We need to figure out how to "assemble" these trees into copies of $\vec{H}$.

Bressan's DAG-tree decomposition gives a systematic method to perform this assembly, and the running time is $O(n^\tau)$, where τ is the "DAG-treewidth" of $\vec{H}$. The definition is technical, so we do not give details here. Also, this method only gives the homomorphism count (edge-preserving maps from H to G), and we require further techniques to get subgraph counts [20]. The main contribution of the linear time dichotomy theorems is to completely characterize patterns H such that all orientations $\vec{H}$ have DAG-treewidth one [8, 6].

Since τ is a natural number, to get subquadratic time algorithms, we need new ideas.

1.2.1 The 6-cycle

It is well-known (from [4]) that linear-time orientation based methods hit an obstruction at 6-cycles. Consider the oriented 6-cycle in the left of Fig. 1. It is basically a "triangle" of out-out wedges (as given by the red lines); indeed, one can show that 6-cycle counting in bounded degeneracy graphs is essentially equivalent to triangle counting in arbitrary graphs [7, 8, 6]. This observation can be converted into an algorithm, as was shown by GLSY [29]. Starting with $\vec{G}$, we create a new graph with edges (the red lines) corresponding to the endpoints of out-out wedges. Since $\vec{G}$ has bounded outdegree, the number of edges in the new graph is $O(n)$. Every triangle in the new graph corresponds to a (directed) 6-cycle homomorphism in $\vec{G}$. Triangle counting in the new graph can be done in $O(n^{1.41})$ time (or $n^{3/2}$ time using a combinatorial algorithm) [4]. Getting the subgraph count is more involved, but we can use existing methods that reduce to homomorphism counting [20].

One can extend this idea more generally as follows. Let $\vec{H}$ be a directed pattern, a source is any vertex with no incoming arcs, and we define an intersection vertex as any vertex that can be "reached" by two different sources.

Suppose there are k sources and k intersection vertices. Suppose further that there is an ordering of the sources $\{s_0, ..., s_{k-1}\}$ and the intersection vertices $\{i_0, ..., i_{k-1}\}$ of $\vec{H}$ such that, for all j, only the sources s_j and s_{j+1} (taking modulo k in the indexes) can both reach the vertex i_j. This is the case of the oriented $\mathcal{C}_6$ and $\mathcal{C}_8$ in Fig. 1 or of any acyclic orientation of any cycle. One can then construct a new graph G' such that $\mathrm{Sub}_{\mathcal{C}_k}(G')$ is the same as $\mathrm{Hom}_{\vec{H}}(\vec{G})$ (where $\mathrm{Hom}_{\vec{H}}(\vec{G})$ denotes the homomorphism count).

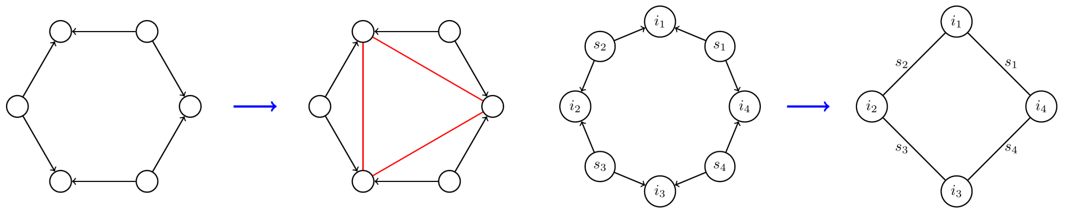

Figure 1 (a) The 6-cycle obstruction, this orientation has three sources intersecting with each other, this oriented pattern can not be counted in linear time in bounded degeneracy graphs. Adding an edge connecting the end-points of every out-out wedge gives a triangle. (b) An example of how the oriented $\mathcal{C}_8$ reduces to a $\mathcal{C}_4$, the four sources become edges connecting the intersection vertices.

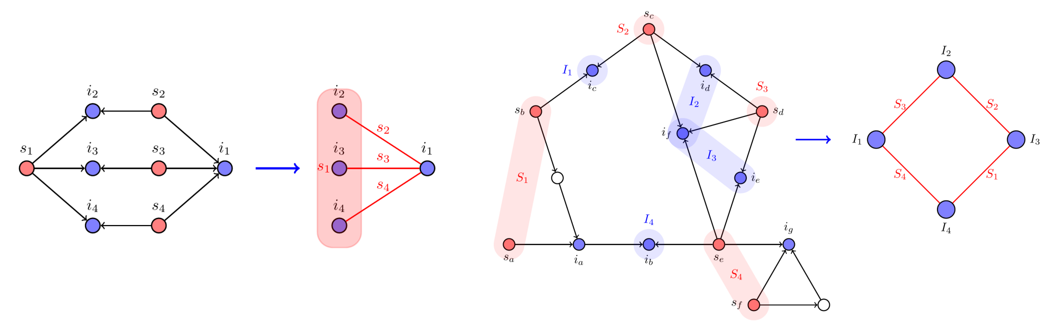

Figure 2 Two more complex examples P-reducibility.

1.2.2 Generalizing to non-cyclic patterns

So far, the algorithmic approach only makes sense for cycle patterns. Our main contribution is a framework that generalizes this approach to count homomorphisms and subgraphs of more complex patterns.

The first part of our framework is the concept of P-reducible patterns. A directed pattern is P-reducible if we can reduce counting homomorphisms of it to counting P subgraphs in a sparse hypergraph. The formal definition is technical and can be seen in §3. We provide a simplified exposition in this section.

Consider a directed pattern $\vec{H}$, for example the left image in Fig. 2. We can replace every source with a hyperedge connecting the intersection vertices reachable by the source. The result is a graph (or hypergraph) P such that $\vec{H}$ is P-reducible.

However, our framework allows for even more freedom: instead of looking at individual sources, we partition sources into sets of sources S_e. See the rightmost figure in Fig. 2 for an example, where the 6 sources are divided into 4 sets of sources. The sub-patterns induced by the vertices reachable by every set of sources are "easy" to count using existing techniques. We can also arrange the intersection vertices into sets I_v such that every set of sources will reach some sets of intersection vertices. We can obtain P by replacing every set of intersection vertices with a vertex and every set of sources S_e with a hyperedge e that contains the vertices corresponding to the sets of intersection vertices that can be reached by S_e.

In the example of Fig. 2, the intersection of the vertices reachable from source sets forms a "cyclic arrangement". The intersection vertices reachable from S_1 are also reachable from S_2 and S_4, and similarly for the other sets of sources, giving that the pattern will be $\mathcal{C}_4$-reducible.

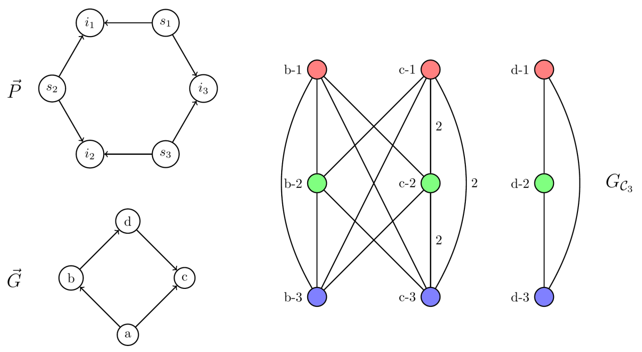

Figure 3 An example of the construction of $G_{\mathcal{C}_3}$, for pattern $\vec{H}$ and input graph $\vec{G}$. The red vertices correspond with i_1 in $\vec{H}$, the green ones with i_2 and the blue ones with i_3. The weight of the edges is 1 except when indicated. For example there are two homomorphisms $\phi : \vec{H}(s_2) \to \vec{G}$ that map i_1 and i_2 to c, hence the edge (c-1, c-2) has weight 2. One can verify that the number of homomorphisms from $\vec{H}$ to $\vec{G}$ is equal to the sum of products of (colorful) triangles in $G_{\mathcal{C}_3}$.

1.2.3 The reduced graph

The second part of our framework is the *reduced graph.* If a pattern is P-reducible, then for any directed input graph $\vec{G}$, we can construct a colored weighted graph G_P with the following property. The number of homomorphisms of the original pattern relates to the number of colorful copies of P in G_P.

The reduced graph consists of $|V(P)|$ layers of vertices, where each layer is related to one intersection set of $\vec{H}$. Specifically, there is a vertex in the j-th layer for every possible image of the corresponding intersection set I_j in $\vec{G}$, that is, every set of vertices in $\vec{G}$ such that I_j can be mapped to it. Moreover, the vertices of every layer will have the same color. For example, for every intersection set I_j in $\vec{H}$ and any map $\phi : I_j \to \vec{G}$ there is a vertex ($\phi(I_j)$-j) in G_P with color j.[3]

The edges have weights that represent the number of homomorphisms mapping the intersection sets to the corresponding images. For example, let S be a source set reaching two intersection sets $I_j = \{i_j\}$ and $I_{j'} = \{i_{j'}\}$, and let $u, v \in \vec{G}$. An edge e connecting (u-j) and (v-j') with weight $w = w(e)$ indicates that there are w different homomorphisms mapping $\vec{H}(S)$ (the subgraph of $\vec{H}$ induced by the vertices reachable by S) to $\vec{G}$ that map j to u and j' to v. Additionally we can show that if $\vec{G}$ has bounded outdegree, then the new reduced graph will have $O(n)$ edges.

We give an example in Fig. 3. For simplicity of exposition, the graph $\vec{G}$ is smaller than the pattern. Each vertex of $\vec{G}$ has three copies (each with a different color) in the reduced graph, denoted $G_{\mathcal{C}_3}$. Since the vertex a cannot be the image of any intersection vertex (it has zero indegree), copies of this vertex do not appear in $G_{\mathcal{C}_3}$. Observe that there is no mapping of an out-out wedge where b is one endpoint and d is the other endpoint. Hence, there is no edge from a copy of b to a copy of d.

Every colorful copy of P in G_P correspond to fixing the positions of the intersection sets in $\vec{G}$, and the weight of each hyperedge will correspond to the number of homomorphisms mapping that portion of the graph. Hence the product of the weights of all hyperedges in each copy will give the total number of homomorphisms mapping all the intersection sets

[3] There could be $O(n^{|I_j|})$ such vertices, however, as we will note later, there will be at most $O(n)$ edges, so we can ignore vertices with degree 0 and we do not need to even create them.

to the corresponding vertices in $\vec{G}$. Therefore, the total number of homomorphisms will be equal to the quantity Col-WSub$_P(G_P)$, which corresponds to the sum of products of weights of colorful copies of P. That is:

$$\text{Col-WSub}_P(G_P) = \sum_{P \in \text{Col-}\mathcal{S}(P,G_P)} \prod_{e \in E(P)} w(e) \tag{1}$$

Here, Col-$\mathcal{S}(P, G_P)$ denotes the set of distinct colorful copies of P in G_P. We are able to show that solving Col-WSub$_P(G_P)$ is equivalent to counting the number of homomorphisms of $\vec{H}$ in $\vec{G}$.

Therefore, we can count homomorphisms of P-reducible patterns in the same time as counting colorful copies of P in the reduced graph.

▶ **Lemma 1.3.** *Let $c > 1$, if there exists a $\tilde{O}(m^c)$ algorithm that for any graph G' computes Col-WSub$_P(G')$. Then, for any P-reducible pattern $\vec{H}$ and directed input graph $\vec{G}$ we can compute* $\text{Hom}_{\vec{H}}(\vec{G})$ *in time $f(d)\tilde{O}(n^c)$, where d is the maximum outdegree of $\vec{G}$.*

1.2.4 From directed to undirected: getting homomorphisms counts

To extend our reduction framework to undirected graphs we introduce the concept of $\mathcal{P}$-computable patterns. Note that different acyclic orientations of the same pattern might reduce to different graphs. We say that a pattern H is $\mathcal{P}$-computable if all the acyclic orientations of H can be computed in linear time or are P-reducible for some $P \in \mathcal{P}$. If for all the patterns in $\mathcal{P}$ we can compute Col-WSub$_P$ in time $O(m^c)$ then we can use Lemma 1.3 to get a bound in the complexity of Hom$_H(G)$.

For each $k \geq 6$, we find a set of hypergraphs $\mathcal{P}_k$ such that all patterns with k vertices are $\mathcal{P}_k$-computable. These sets grow with k and we have $\mathcal{P}_k \subseteq \mathcal{P}_{k+1}$. We give a definition of these sets in Section §5 and prove the following lemma.

▶ **Lemma 1.4.** *Let $k \geq 6$, every connected pattern H with k vertices is $\mathcal{P}_k$-computable.*

For $k = 9$ we are able to show that there exists a subset $\mathcal{P}_9^*$ of $\mathcal{P}_9$ such that every 9-vertex pattern is also $\mathcal{P}_9^*$-computable and for every pattern $P \in \mathcal{P}_9^*$ we can compute Col-WSub$_P$ in $\tilde{O}(m^{5/3})$ time. Allowing us to show that for all patterns with 9 vertices or less we can count the number of homomorphisms in subquadratic time.

We can also show similar results for cycles. All orientations of cycle patterns can be reduced to cycles of half their length. This means that any $2k$-cycle and $2k+1$-cycle are $\{\mathcal{C}_k \cup ... \cup \mathcal{C}_3\}$-computable (in these cases we will simply write $\mathcal{C}_k$-computable).

We can also show that for all cycles we can compute Col-WSub$_{\mathcal{C}_k}(G)$ in $\tilde{O}(m^{d_k})$ time, the fastest time for detecting k-cycles in general graphs.

▶ **Lemma 1.5.** *For all $k \geq 3$, there is an algorithm that computes Col-WSub$_{\mathcal{C}_k}(G)$ in time $\tilde{O}(m^{d_k})$.*

This means that we can compute the number of homomorphisms of $\mathcal{C}_{2k}$ and $\mathcal{C}_{2k+1}$ in time $\tilde{O}(m^{d_k})$, similar result to the one obtained in GLSY [29].

1.2.5 Getting subgraph counts

At this point, we have algorithms for computing various pattern *homomorphisms.* To get subgraph counts, we need the inclusion-exclusion techniques of [20]. One can express the H-subgraph count as a linear combination of H'-homomorphism counts, where H' is a pattern

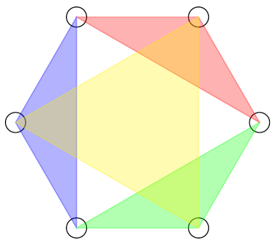

Figure 4 The hypergraph $\mathcal{H}_\triangle$.

in $Spasm(H)$. The $Spasm(H)$ consists of all patterns H' such that H has a surjective homomorphism to H'. Thus, every pattern in the spasm has at most as many vertices as H.

Hence for any pattern H with k vertices, all the patterns in $Spasm(H)$ will have at most k vertices and hence they will also be $\mathcal{P}_k$-computable, giving Theorem 1.1.

In the case of the cycles, to be able to extend the results from last section to the Sub problem, we analyze the spasm of the different cycles. For $k \leq 10$ we are able to show that the patterns in the spasms of $\mathcal{C}_k$ are also $\mathcal{C}_{\lfloor k/2 \rfloor}$-computable. That combined with Lemma 1.5 implies the upper bound of Theorem 1.2.

1.2.6 Inverting the reduction for conditional hardness

We show that in some cases our reduction procedure is optimal. For example, counting small cycles in general graphs can be reduced to counting cycles of twice the length in graphs of degeneracy $\kappa = 2$. The reduction is quite simple and just involves subdividing the edges. With a slight modification, the subdivision approach can be used to show lower bounds for odd cycles too, which gives the lower bound of Theorem 1.2. We note that an analogous result for counting homomorphisms was shown in GLSY [29].

▶ **Lemma 1.6.** *Let $6 \leq k \leq 10$, an $f(\kappa)o(n^{d_{\lfloor k/2 \rfloor}})$ algorithm for counting k-cycles implies the existence of a $o(m^{d_{\lfloor k/2 \rfloor}})$ algorithm for counting $\lfloor k/2 \rfloor$-cycles.*

This inversion of the reduction procedure also gives us an algorithm for counting undirected 5-cycles in general graphs which improves on the current state of the art, and matches the complexity for 5-cycle detection.

▶ **Corollary 1.7.** *There is an algorithm that, for any graph G, computes $\mathrm{Sub}_{\mathcal{C}_5}(G)$ in time $O\left(m^{d_5}\right) \approx O\left(m^{1.63}\right)$.*

The approach of subdividing edges can be used to prove a relation between other patterns too. In the case of hypergraphs, we can replace hyperedges of arity r by r-stars. We are able to show a strong relation between the hypergraph $\mathcal{H}_\triangle$ depicted in Fig. 4, and a 10-vertex pattern. We conjecture that for this pattern we can not count the number of subgraphs in subquadratic time.

▶ **Conjecture 1.8.** *There is no $o(m^2)$ algorithm for computing $\mathrm{Sub}_{\mathcal{H}_\triangle}(G)$.*

This conjecture implies that there are no subquadratic algorithms for computing the number of subgraphs of all 10-vertex patterns.

▶ **Lemma 1.9.** *If Conjecture 1.8 holds, there is no algorithm that computes $\mathrm{Sub}_H(G)$ in time $f(\kappa)o(n^2)$ for all patterns H with 10 vertices.*

We consider it an interesting open problem to relate our conjecture with existing fine-grained complexity assumptions. Some previous works prove barriers for subquadratic *listing* in general graphs [17], but no similar results exist for counting hypergraphs.

1.3 Related Work

Subgraph counting is closely tied to homomorphism counting; in some cases, it is more convenient to talk about the latter. Seminal work of Curticepean-Dell-Marx showed that the optimal algorithms for subgraph counting can be designed from homomorphism counting algorithms and vice versa [20].

Much of the study of subgraph/homomorphism counting comes from paramterized complexity theory. Díaz et al [24] gave a $O(2^k n^{tw(H)+1})$ algorithm for determining the H-homomorphism count, where $tw(H)$ is the treewidth of H. Dalmau and Jonsson [22] proved that $\mathrm{Hom}_H(G)$ is polynomial time solvable iff H has bounded treewidth. Otherwise it is $\#W[1]$-complete. Roth and Wellnitz [47] consider restrictions of both H and G, and focus of $\#W[1]$-completeness.

Tree decompositions have played an important role in subgraph counting. We give a brief review of the graph parameters treewidth and degeneracy. The notion of tree decomposition and treewidth were introduced in a seminal work by Robertson and Seymour [44, 45, 46], although the concept was known earlier [10, 31].

Degeneracy is a measure of sparsity and has been known since the early work of Szekeres-Wilf [52]. We refer the reader to the short survey of Seshadhri [48] about degeneracy and algorithms. The degeneracy has been exploited for subgraph counting problems in many algorithmic results [19, 27, 2, 35, 42, 40, 34, 41].

Bressan connected degeneracy to treewidth-like notation and introduced the concept of DAG treewidth [12, 13]. The main result is the following. 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 [32], the subgraph counting problem does not admit any $f(\kappa, k)m^{o(\tau(H)/\ln \tau(H))}$ algorithm, for any positive function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Bera-Pashanasangi-Seshadhri introduced the first theory of linear time homomorphism counting [7], showing that all patterns with at most 5 vertices could be counted in linear time. It was later shown that for every pattern with no induced cycles of length 6 or more, the number of homomorphisms could also be counted in linear time [8, 6].

A recent work of Komarath et al. [36] gave quadratic and cubic algorithms for counting cycles in sparse graphs. Gishboliner et al. gave subquadratic algorithms for homomorphism counting of cycles in bounded degeneracy graphs [29]. Bressan, Lanziger and Roth have also studied counting algorithms for directed patterns [14].

1.4 Paper Organization

In Section 3, we define our reduction framework formally and prove the equivalence between the bounded degeneracy and the general setting. In Section 4, we prove that many patterns are cycle-reducible. In Section 5, we define the sets $\mathcal{P}_k$ and complete the proof of the main theorem. Finally, in Section 6, we show how to compute Col-WSub for cycle patterns. Due to space limitations most proofs have been deferred to the appendix, including the proofs of Lemma 1.6 and Lemma 1.9.

2 Preliminaries

Graphs, subgraphs and homomorphisms

We use $H = (V(H), E(H))$ to denote the pattern graph and $G = (V(G), E(G))$ to denote the input graph. We will use $n = |V(G)|$ and $m = |E(G)|$ for the number of vertices and edges of G respectively.

A homomorphism from H to G is a mapping $\phi : V(H) \to V(G)$ such that $\forall (u, v) \in E(H)$ we have $(\phi(u), \phi(v)) \in E(G)$. We use $\Phi(H, G)$ to denote the set of all homomorphisms from H to G. We use $\mathrm{Hom}_H(G)$ to denote the problem of counting the number of homomorphisms from H to G, that is, computing $|\phi(H, G)|$. Similarly we use $\mathrm{Sub}_H(G)$ to denote the problem of counting the number of subgraphs (not necessarily induced) of G isomorphic to H.

We say that two homomorphisms $\phi : V \to G$ and $\phi' : V' \to G$ agree if for any vertex $v \in V \cap V'$ we have $\phi(v) = \phi'(v)$.

The spasm of a graph is the set of all possible graphs obtained by recursively combining two vertices that are not connected by an edge, removing any duplicated edge. Using inclusion-exclusion arguments one can express the value of $\mathrm{Sub}_H(G)$ as a weighted sum of homomorphism counts of the graphs in the spasm of H. Hence, computing $\mathrm{Hom}_{H'}(G)$ for all $H' \in \mathit{Spasm}(H)$, allows to compute $\mathrm{Sub}_H(G)$, as given by the following equation: $\mathrm{Sub}_H(G) = \sum_{H' \in \mathit{Spasm}(H)} f(H')\mathrm{Hom}_{H'}(G)$.

Here $f(H')$ are a series of non-zero coefficients that can be computed for each H. See [11] for more details. Curticapean, Dell and Marx showed that this process is optimal [20], that is, the complexity of $\mathrm{Sub}_H(G)$ is exactly the hardest complexity of $\mathrm{Hom}_{H'}(G)$ for the graphs in $\mathit{Spasm}(H)$.

Degeneracy and directed graphs

A graph G is κ-degenerate if every subgraph has a minimum degree of at most κ. The degeneracy $\kappa(G)$ of a graph is the minimum value such that G is κ-degenerate. There exists an acyclic orientation of a graph, called the degeneracy orientation, which has the property that the maximum outdegree of the graph is at most κ [39]. Additionally, this orientation can be computed in $O(n + m)$ time. We will use $\vec{G}$ to denote the directed input graph. For a pattern H, we use $\Sigma(H)$ to denote the set of acyclic orientations of H. When orienting the input graph, every occurrence of H will now appear as exactly one of its acyclic orientations, that is, $\mathrm{Hom}_H(G) = \sum_{\vec{H} \in \Sigma(H)} \mathrm{Hom}_{\vec{H}}(\vec{G})$.

We say that $v \in V(\vec{H})$ is a source if its in-degree is 0. We use $S(\vec{H})$ to denote the set of sources of $\vec{H}$. We say a vertex u is reachable from v if there is a directed path connecting v to u and use $\mathit{Reach}_{\vec{H}}(s)$ to denote the set of vertices reachable by the source s. Abusing notation, for a set $S' \subseteq S(\vec{H})$ we use $\mathit{Reach}_{\vec{H}}(S')$ (or $\mathit{Reach}(S')$ if $\vec{H}$ is clear from the context) to denote the set of vertices reachable by any vertex in S'. We use $\vec{H}(S)$ and $\vec{H}(s)$ for the subgraphs of $\vec{H}$ induced by $\mathit{Reach}_{\vec{H}}(S)$ and $\mathit{Reach}_{\vec{H}}(s)$.

We say that a vertex v of $\vec{H}$ is an intersection vertex if there are at least two distinct sources $s, s' \in S(\vec{H})$ such that v is reachable by both of them, that is, $v \in \vec{H}(s) \cap \vec{H}(s')$. We use $I(\vec{H})$ to denote the set of intersection vertices. Note that $S(\vec{H}) \cap I(\vec{H}) = \emptyset$.

The DAG-treewidth

Bressan introduced the concept of DAG-tree decomposition of a directed acyclic graph [12]:

▶ **Definition 2.1** (DAG-tree decomposition [12]). *For a given directed acyclic graph $\vec{H} = (V(\vec{H}), E(\vec{H}))$, a DAG-tree decomposition of $\vec{H}$ is a rooted tree $T = (\mathcal{B}, \mathcal{E})$ such that:*

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

The DAG-treewidth of a DAG-tree decomposition T, $\tau(T)$, is the maximum size of all the bags in T. The DAG-treewidth of a directed acyclic graph $\vec{H}$ is the minimum value of $\tau(T)$ across all valid DAG-tree decomposition T of $\vec{H}$. For an undirected graph H, we will have that $\tau(H) = \max_{\vec{H}\in\Sigma(H)} \tau(\vec{H})$.

Using DAG-tree decomposition to compute homomorphisms

Bressan gave an algorithm that computes $\mathrm{Hom}_H(G)$ making use of the DAG-tree decomposition of a directed pattern [12, 13]. The algorithm decomposes the pattern into smaller subgraphs, computes the number of homomorphisms of every subgraph and then combines the counts using dynamic programming.

▶ **Theorem 2.2.** *[12] For any pattern H with k vertices there is an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(k,\kappa)\tilde{O}(n^{\tau(H)})$.*

For the patterns with $\tau(H) = 1$ we obtain an algorithm that runs in linear time when parameterized by the degeneracy of the input graph. Bera et al. showed an exact characterization of which patterns have $\tau(H) = 1$.

▶ **Lemma 2.3.** *[8] $LICL(H) < 6 \Leftrightarrow \tau(H) = 1$.*

If we analyze the algorithm from Bressan in more detail, we can see that it can be used as a black box to obtain some more fine-grained counts. In order to understand this we need to introduce the following definition:

A homomorphism ϕ' extends ϕ if for every vertex $u \in \phi$ we have $\phi(u) = \phi'(u)$. Let ϕ be a homomorphism from a subgraph $\vec{H}'$ of $\vec{H}$ to $\vec{G}$. We define $ext\left(\vec{H}, \vec{G}; \phi\right)$ as the number of homomorphisms ϕ' from $\vec{H}$ to $\vec{G}$ that extend ϕ.

▶ **Lemma 2.4** (Lemma 5 in [13] (restated)). *There exists an algorithm that, given a directed pattern $\vec{H}$ with k vertices and a DAG-tree decomposition T rooted in s with $\tau(T) = 1$, and a directed graph $\vec{G}$ with n vertices and maximum outdegree d; returns, for every homomorphism $\phi : \vec{H}[s] \to \vec{G}$, the quantity $ext\left(\vec{H}, \vec{G}; \phi\right)$. The algorithm runs in time $f(k,d)\tilde{O}(n)$.*

Hypergraphs

A hypergraph is a graph where the each edge (or hyperedge) is a subset of the vertices. The arity of a hyperedge is the number of vertices that it contains. We will only consider hypergraphs where every hyperedge has arity at least 2. We use $E(G)$ for the set of hyperedges of G, where for each $e \in E(G)$ we have $e \subseteq V(G)$.We will also consider weighted hypergraphs, for a hypergraph G, the function $w : E(G) \to \mathbb{N}$ gives the weight of each hyperedge e in G. We use $\mathcal{S}_k$ to denote the simplex hypergraph of arity k.

3 The reduction procedure

In this section we explain our reduction procedure. We start with the concept of P-reducibility. The main idea is to divide the set of sources of the directed pattern $\vec{H}$ into $|E(P)|$ different subsets of sources S_e, each corresponding to one hyperedge e in $E(P)$, such that every source in $S(\vec{H})$ belong to exactly one subset S_e. We also require that the subgraphs reachable by every set of source can be counted efficiently using Bressan's algorithm.

We also set $|V(P)|$ distinct subsets of intersections vertices $I_v \subseteq I(\vec{H})$, with each subset corresponding to each vertex v in $V(P)$. Not all the intersection vertices of $I(\vec{H})$ must belong necessarily to one of the subsets, and we use I^* to denote the set of intersection vertices that appear in at least one of the set. Moreover, different intersection sets can contain the same intersection vertex, however we require that the vertices corresponding to sets that contain the same intersection vertex in $V(P)$ induce a connected subgraph.

We use $I(e)$ to denote the vertices in I^* reachable by the sources in S_e. For every hyperedge e of P, we require that the corresponding set of sources can reach all the vertices in the intersection sets corresponding to the vertices of e. Also the set of sources related to every hyperedge containing the vertex v must reach all the vertices in I_v. These conditions ensure that we will be able to combine the homomorphism counts for each of the subgraphs $\vec{H}(S_e)$. We now present the full definition.

▶ **Definition 3.1** (P-reducible)**.** *A connected DAG $\vec{H}$ is P-reducible if we can set:*

- *For every vertex $v \in V(P)$, a subset of intersection vertices $I_v \subseteq I(\vec{H})$. With $I^* = \bigcup_i I_i$. Such that for every intersection vertex $i \in I^*$, we have that the vertices $\{v : i \in I_v\} \subseteq V(P)$ induce a connected hypergraph in P.*
- *For every hyperedge $e \in E(P)$, a subset of sources $S_e \subseteq S(\vec{H})$. With $S_e \cap S_{e'} = \emptyset \; \forall e \neq e'$ and $\bigcup_{e \in E(P)} S_e = S(\vec{H})$. Such that every subset of sources S_e contains a source s_e with $\vec{H}(s_e) \cap I^* = \vec{H}(S_e) \cap I^* = I(e)$, and the subgraph $\vec{H}(S_e)$ admits a $\tau = 1$ DAG-tree decomposition rooted at the source s_e.*

Satisfying:

1. *For every vertex $v \in V(P)$: $I_v \subseteq I(e) \forall e \ni v$*
2. *For every hyperedge $e \in E(P)$: $\bigcup_{v \in e} I_v = I(e)$*

We now define the reduced graph G_P.

▶ **Definition 3.2** (Reduced graph G_P)**.** *Given a P-reducible directed pattern on source sets $\{S_e : e \in E(P)\}$ and intersection sets $\{I_v : v \in V(P)\}$, we define the reduced graph G_P of the directed input graph $\vec{G}$ as follows:*

- *For every vertex $v \in V(P)$ and every homomorphism $\phi : I_v \to \vec{G}$ we have the vertex ($\phi(I_v)$-v) with color v. The vertices with the same color form the "layers" of G_P.*
- *For every hyperedge $e \in E(P)$ and for every homomorphism $\phi : I(e) \to \vec{G}$, let ϕ_v be the restriction of ϕ to I_v for each vertex $v \in e$, we will have a hyperedge connecting the vertices $\{(\phi_v(I_v)\text{-}v) : v \in e\}$ with weight ext $\left(\vec{H}(S_e), \vec{G}; \phi\right)$.*

We use $V^{(v)}(G_P)$ to refer to the vertices of G_P in the v-th layer. The number of vertices in every layer v can be up to $O(n^{|I_v|})$, however we will only consider vertices that are not isolated, that is, have degree at least 1. We can show that we can construct G_P efficiently when only considering such vertices.

▶ **Lemma 3.3.** *Given a P-reducible pattern $\vec{H}$ and a directed graph $\vec{G}$ with maximum outdegree d, we can construct G_P in $f(d)\tilde{O}(n)$ time. Additionally, the number of non-isolated vertices and the total number of hyperedges are bounded by $f(d)O(n)$.*

We can now prove the equivalence between homomorphisms of the original pattern and weighted colorful copies of the reduced hypergraph. This lemma relates the counts between the original and the reduced graph.

▶ **Lemma 3.4.** $\mathrm{Hom}_{\vec{H}}(\vec{G}) = \mathit{Col\text{-}WSub}_P(G_P)$

Finally, we have all the tools to complete our reduction framework, giving us Lemma 1.3.

▶ **Lemma 1.3.** *Let $c > 1$, if there exists a $\tilde{O}(m^c)$ algorithm that for any graph G' computes $\mathit{Col\text{-}WSub}_P(G')$. Then, for any P-reducible pattern $\vec{H}$ and directed input graph $\vec{G}$ we can compute $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in time $f(d)\tilde{O}(n^c)$, where d is the maximum outdegree of $\vec{G}$.*

Proof. For any P-reducible pattern we can use Lemma 3.3 to construct the reduced G_P graph in time $f(d)\tilde{O}(n)$ for any input graph $\vec{G}$. This graph will have $f(d)O(n)$ edges. We can then use the $\tilde{O}(m^c)$ algorithm to compute $\mathrm{Col\text{-}WSub}_P(G_P)$ in time $f(d)\tilde{O}(n^c)$. From Lemma 3.4 we have that $\mathrm{Col\text{-}WSub}_P(G_P)$ will be equal to $\mathrm{Hom}_{\vec{H}}(\vec{G})$. ◀

3.1 From directed to undirected

We introduce the concept of $\mathcal{P}$-computable, which will help us give upper bounds in the complexity of undirected patterns.

▶ **Definition 3.5** ($\mathcal{P}$-computable)**.** *Let $\mathcal{P}$ be a set of hypergraphs. We say that a pattern H is $\mathcal{P}$-computable if every acyclic orientation $\vec{H} \in \Sigma(H)$ has either DAG-treewidth of 1 or there exists a hypergraph $P \in \mathcal{P}$ such that $\vec{H}$ is P-reducible.*

If a pattern H is $\mathcal{P}$-computable and $\mathcal{P}$ is only formed by cyclic patterns we will instead write $\mathcal{C}_l$-computable, where l is the length of the largest cycle in $\mathcal{P}$. The complexity of computing homomorphisms of $\mathcal{P}$-computable patterns will be dominated by the hardest complexity for computing $\mathrm{Col\text{-}WSub}_P$.

▶ **Lemma 3.6.** *Let $\mathcal{P}$ be a set of hypergraphs, if for every hypergraph $P \in \mathcal{P}$ there is an algorithm that computes $\mathit{Col\text{-}WSub}_P$ in time $\tilde{O}(m^c)$, then for any input graph G with degeneracy κ and any $\mathcal{P}$-computable pattern H, there is an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa)\tilde{O}(n^c)$.*

4 Reducing to cycles

We first focus on patterns that can be reduced to counting cycles. We start by introducing the following two lemmas, showing that directed patterns with either few sources or few intersection vertices are $\mathcal{C}_3$-reducible.

▶ **Lemma 4.1.** *Every directed acyclic pattern $\vec{H}$ with at most 3 sources is either $\mathcal{C}_3$-reducible or $\tau(\vec{H}) = 1$.*

▶ **Lemma 4.2.** *Every directed acyclic pattern $\vec{H}$ with at most 3 intersection vertices is either $\mathcal{C}_3$-reducible or $\tau(\vec{H}) = 1$.*

Using this two lemmas we can prove that all 6 and 7-vertex patterns are $\mathcal{C}_3$-computable.

▶ **Lemma 4.3.** *Every 6 and 7-vertex undirected pattern H is $\mathcal{C}_3$-computable.*

Additionally, the patterns in the spasm will always have less vertices, and hence all patterns in the spasms of all 6 and 7-vertex patterns will also be $\mathcal{C}_3$-computable. This fact, together with Lemma 1.5 gives the following.

▶ **Corollary 4.4.** *Let H be a pattern with 6 or 7 vertices. For any input graph G, we can compute $\mathrm{Hom}_H(G)$ and $\mathrm{Sub}_H(G)$ in time $f(\kappa)\tilde{O}(n^{d_3}) \approx f(\kappa)O(n^{1.41})$.*

We can also show that the acyclic orientations of cycle patterns are always $\mathcal{C}_k$-reducible for some k at most half of the length of the cycle. This is equivalent to the result in [29], but expressed using our reducibility framework.

▶ **Lemma 4.5.** *For all $k \geq 3$, $\mathcal{C}_{2k}$ and $\mathcal{C}_{2k+1}$ are $\mathcal{C}_k$-computable.*

Moreover, we can also show that all patterns in the spasms of cycles up to length 10 are also cycle-computable.

▶ **Lemma 4.6.**
- *All the patterns in Spasm($\mathcal{C}_6$) and Spasm($\mathcal{C}_7$) are $\mathcal{C}_3$-computable.*
- *All the patterns in Spasm($\mathcal{C}_8$) and Spasm($\mathcal{C}_9$) are $\mathcal{C}_4$-computable.*
- *All the patterns in Spasm($\mathcal{C}_{10}$) are $\mathcal{C}_5$-computable.*

This lemma allows us to prove the upper bound of Theorem 1.2.

▶ **Lemma 4.7.** *For all $6 \leq k \leq 10$, there is an algorithm that computes $\mathrm{Sub}_{\mathcal{C}_k}(G)$ in time $f(\kappa)O(n^{d_{\lfloor k/2 \rfloor}})$.*

5 Reducing to other patterns

Consider the set of directed patterns with 8 vertices. Using the results of the previous section we can show that most of the orientations will either admit a DAG-tree decomposition with $\tau = 1$ or will be $\mathcal{C}_3$-reducible. However, if a pattern $\vec{H}$ has 4 sources and 4 intersection vertices then it might not be cycle-reducible. Instead we might need to reduce to some hypergraphs, like in Fig. 2. The following definitions will help us determining which patterns we will need to reduce to for patterns with at least 8 vertices.

▶ **Definition 5.1.** *[$\mathcal{P}_{i,s}$, $\mathcal{P}_k$] We define $\mathcal{P}_{i,s}$ as the set of hypergraphs P with i vertices and s hyperedges such that:*

1. *Every vertex has degree at least 2.*
2. *Every hyperedge contains at least 2 vertices.*
3. *No hyperedge is a subset of any other hyperedge.*
4. *For every pair of distinct vertices $u, v \in V(P)$ the set of hyperedges containing u can not be equal or a subset of the set of hyperedges containing v.*

For any $k \geq 7$, we define $\mathcal{P}_k$ recursively as the union of $\mathcal{P}_{k-1}$ and all sets $\mathcal{P}_{i,s}$, with $i+s=k$ and $i, s \geq 4$, with $\mathcal{P}_6 = \{\mathcal{C}_3\}$.

We can prove that for any k, patterns with k vertices will reduce to some pattern in $\mathcal{P}_k$. This was stated earlier as Lemma 1.4.

▶ **Lemma 1.4.** *Let $k \geq 6$, every connected pattern H with k vertices is $\mathcal{P}_k$-computable.*

In order to prove Theorem 1.1, we show exactly which patterns form $\mathcal{P}_9$.

▶ **Lemma 5.2.** $\mathcal{P}_9 = \{\mathcal{C}_3, \mathcal{C}_4, \mathcal{D}, \mathcal{S}_3, \mathcal{H}_1, \mathcal{H}_2\}$

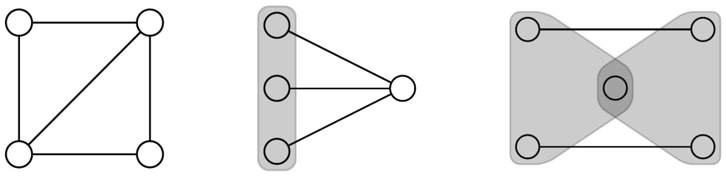

Figure 5 The diamond graph $\mathcal{D}$, the hypergraph $\mathcal{H}_1$ and the hypergraph $\mathcal{H}_2$.

Where, $\mathcal{D}$ is the diamond pattern, $\mathcal{S}_3$ the 3-simplex, $\mathcal{H}_1$ and $\mathcal{H}_2$ are the two hypergraphs shown in Fig. 5. It turns out that simplex-reducible patterns are also cycle-reducible. Hence we can set $\mathcal{P}_9^* = \mathcal{P}_9 \setminus \mathcal{S}_3$ and show that every $\mathcal{P}_9$-computable pattern is also $\mathcal{P}_9^*$-computable.

▶ **Lemma 5.3.** *If a pattern H is $\mathcal{P}_9$-computable, then it is also $\mathcal{P}_9^*$-computable*

We can show that all the hypergraphs in $\mathcal{P}_9^*$ can be counted in subquadratic time.

▶ **Lemma 5.4.** *For any weighted colored hypergraph G with m edges, there is an algorithm that computes Col-WSub$_P(G)$ for all patterns $P \in \mathcal{P}_9^*$ in time $\tilde{O}(m^{5/3})$.*

Finally we can prove the main theorem.

▶ **Theorem 1.1.** *(Main Theorem) There is an algorithm that computes $\mathrm{Sub}_H(G)$ for all patterns H with at most 9 vertices in time $f(\kappa)\tilde{O}(n^{5/3})$.*

Proof. Let H be a pattern with 9 or less vertices. From Lemma 1.4 we have that H is $\mathcal{P}_9$-computable, using Lemma 5.3 we will have that it is also $\mathcal{P}_9^*$-computable. Additionally, from Lemma 5.4 we have that for all hypergraphs $P \in \mathcal{P}_9^*$ we can compute $\mathrm{Col\text{-}WSub}_P(G)$ in $\tilde{O}(m^{5/3})$ time. This together with Lemma 3.6 gives that we compute $\mathrm{Hom}_H(G)$ in $f(\kappa)\tilde{O}(n^{5/3})$ time.

All the graphs H' in the Spasm of H have also at most 9 vertices, hence we can compute $\mathrm{Hom}_{H'}(G)$ for them and use inclusion-exclusion to obtain the value of $\mathrm{Sub}_H(G)$ in total time $f(\kappa)\tilde{O}(n^{5/3})$. ◀

6 Counting cycles

We adapt the two algorithms for counting weighted homomorphisms of cycles shown in [29] for computing $\mathrm{Col\text{-}WSub}_{\mathcal{C}_k}$. The first is a combinatorial algorithm that matches the complexity of detecting directed cycles combinatorially [4]. The second is a matrix multiplication based algorithm which adapts the algorithm from [53]. The complexity of this algorithm for counting $\mathcal{C}_k$ is given by the value c_k, the exact values of c_k for $k \geq 6$ are not known, but the following upper bound holds [21]:

$$c_k \leq \frac{\omega(k+1)}{2\omega + k - 1} \quad \text{if } k \text{ is odd} \qquad c_k \leq \frac{\omega k - 4/k}{2\omega + k - 2 - 4/k} \quad \text{if } k \text{ is even} \tag{2}$$

Where ω is the matrix multiplication exponent.

▶ **Lemma 6.1.** *Let G be a colored weighted graph with m edges. For all $k > 3$:*

- *There is a combinatorial algorithm that computes Col-WSub$_{\mathcal{C}_k}(G)$ in time $\tilde{O}(m^{2-1/\lceil k/2 \rceil})$.*
- *There is an algorithm that computes Col-WSub$_{\mathcal{C}_k}(G)$ in time $\tilde{O}(m^{c_k})$.*

For any k we use d_k for the fastest of the two algorithms, similar to [29]. Lemma 1.5 follows directly from Lemma 6.1 and the following equation:

$$d_k = min(2 - 1/\lceil k/2 \rceil, c_k) \tag{3}$$

Note that $d_k < 2$ for all k, hence we have subquadratic algorithms for all cycles. For $k < 6$ and using the fastest matrix multiplication exponent known to date $\omega = 2.371339$ [3], we have that the matrix multiplication algorithm is faster and we get the following approximate values of d_k: $d_3 \approx 1.41$, $d_4 \approx 1.48$ and $d_5 \approx 1.63$.

References

1 Escape. `https://bitbucket.org/seshadhri/escape`.

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. `doi:10.1109/ICDM.2015.141`.

3 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2039. `doi:10.1137/1.9781611978322.63`.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. `doi:10.1007/BF02523189`.

5 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. `doi:10.4230/LIPIcs.ICALP.2020.11`.

6 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. `doi:10.1145/3520240`.

7 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. `doi:10.4230/LIPIcs.ITCS.2020.38`.

8 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, page 2315–2332, 2021. `doi:10.1137/1.9781611976465.138`.

9 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. `doi:10.1145/3375395.3387665`.

10 Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973. `doi:10.1016/0097-3165(73)90016-2`.

11 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. `doi:10.1007/3-540-33700-8_18`.

12 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. `doi:10.4230/LIPIcs.IPEC.2019.6`.

13 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021. `doi:10.1007/s00453-021-00811-0`.

14 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. `doi:10.1145/3564246.3585204`.

15 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.

16 Graham R Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of combinatorial theory, series B*, 77(2):221–262, 1999. doi:10.1006/jctb.1999.1899.

17 Karl Bringmann and Egor Gorbachev. A fine-grained classification of subquadratic patterns for subgraph listing and friends, 2024. URL: https://arxiv.org/abs/2404.04369, arXiv:2404.04369.

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. doi:10.1145/800105.803397.

19 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing (SICOMP)*, 14(1):210–223, 1985. doi:10.1137/0214017.

20 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. doi:10.1145/3055399.3055502.

21 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: hardness for all induced patterns and faster non-induced cycles. STOC 2019, page 1167–1178, 2019. doi:10.1145/3313276.3316329.

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. doi:10.1016/j.tcs.2004.08.008.

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. doi:10.4230/LIPIcs.ICALP.2019.113.

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. doi:10.1016/S0304-3975(02)00017-8.

25 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000. doi:10.1002/1098-2418(200010/12)17:3/4\%3C260::AID-RSA5\%3E3.0.CO;2-W.

26 Martin E Dyer and David M Richerby. On the complexity of# csp. In *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, pages 725–734, 2010. doi:10.1145/1806689.1806789.

27 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.

28 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing (SICOMP)*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

29 Lior Gishboliner, Yevgeny Levanzov, Asaf Shapira, and Raphael Yuster. Counting homomorphic cycles in degenerate graphs. *ACM Trans. Algorithms*, 19(1), February 2023. doi:10.1145/3560820.

30 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. doi:10.1007/11917496_15.

31 Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8(1-2):171–186, 1976. doi:10.1007/BF01917434.

32 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. doi:10.1109/SFCS.1998.743516.

33 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. doi:10.1137/0207033.

34 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. doi:10.1145/3038912.3052636.

35 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. doi:10.1145/2736277.2741101.

36 Balagopal Komarath, Anant Kumar, Suchismita Mishra, and Aditi Sethia. Finding and Counting Patterns in Sparse Graphs. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, pages 40:1–40:20, 2023. doi:10.4230/LIPIcs.STACS.2023.40.

37 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967. doi:10.1007/BF02280291.

38 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

39 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. doi:10.1145/2402.322385.

40 Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science*, 2(1), 2017. doi:10.1007/s41109-017-0027-2.

41 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. doi:10.1145/3336191.3371773.

42 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. doi:10.1145/3038912.3052597.

43 Natasa Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007. doi:10.1093/bioinformatics/btl301.

44 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

45 Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

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

47 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. doi:10.1137/1.9781611975994.133.

48 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. doi:10.4230/LIPIcs.ICDT.2023.3.

49 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. doi:10.1145/3308560.3320092.

50 Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009. URL: http://proceedings.mlr.press/v5/shervashidze09a.html.

51 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. doi:10.1007/s10115-017-1077-6.

52 George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968. doi:10.1016/S0021-9800(68)80081-X.

53 Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 254–260, 2004. URL: `http://dl.acm.org/citation.cfm?id=982792.982828`.

A Proofs of Section 3

A.1 Proof of Lemma 3.3

First, we show how to compute homomorphism extensions of the different subgraphs $\vec{H}(S_e)$ of P-reducible patterns, this is a necessary step in order to construct the reduced graph.

▶ **Lemma A.1.** *Let $\vec{H}$ be a P-reducible directed pattern and $\vec{G}$ a directed input graph with n vertices and maximum outdegree d. Let $\{S_e : e \in E(P)\}$ and $\{I_v : v \in V(P)\}$ be the sets of sources and intersections that achieve the P-reducibility. For every set of sources S_e we can compute $ext\left(\vec{H}(S_e), \vec{G}; \phi\right)$ for every $\phi : I(e) \to \vec{G}$ in $f(d)\tilde{O}(n)$ time. Additionally, there are at most $f(d)O(n)$ homomorphisms ϕ with non-zero extension.*

Proof. Fix a set of sources S_e. From the definition of P-reducible we have that $\vec{H}(S_e)$ admits a DAG-tree decomposition with $\tau = 1$ rooted at s_e. Therefore we can use Bressan's results in Lemma 2.4 to compute the quantity $ext\left(\vec{H}(S_e), \vec{G}; \phi\right)$ for each homomorphism $\phi : \vec{H}(s_e) \to \vec{G}$. Note that from Definition 3.1, we have that $I(e) \subseteq \vec{H}(s_e)$, hence it is possible to iterate over the homomorphisms from $\vec{H}(s_e)$ and aggregate the counts in order to obtain, for each homomorphism $\phi' : I(e) \to \vec{G}$ the quantity $ext\left(\vec{H}(S_e), \vec{G}; \phi'\right)$, that is, the number of homomorphisms of $\vec{H}(S_e)$ in $\vec{G}$ that map the vertices in $I(e)$ to some specific set of vertices.

The number of homomorphisms from $\vec{H}(s_e)$ to $\vec{G}$ is bounded by $O(nd^k)$. Hence the call to Bressan's and the aggregation will take a total of $f(d)\tilde{O}(n)$ time.

Additionally, each homomorphism from $\vec{H}(s_e)$ to $\vec{G}$ can contribute to exactly one homomorphism from $I(e)$ to $\vec{G}$, therefore the total number of such homomorphisms with non-zero extension will be at most $f(d)O(n)$. ◀

We can now prove Lemma 3.3.

▶ **Lemma 3.3.** *Given a P-reducible pattern $\vec{H}$ and a directed graph $\vec{G}$ with maximum outdegree d, we can construct G_P in $f(d)\tilde{O}(n)$ time. Additionally, the number of non-isolated vertices and the total number of hyperedges are bounded by $f(d)O(n)$.*

Proof of Lemma 3.3. Let $\{S_e : e \in E(P)\}$ and $\{I_v : v \in V(P)\}$ be the sets of sources and intersection vertices that achieve the P-reducibility. Using Lemma A.1 we can compute, for every homomorphism $\phi : I(e) \to \vec{G}$ the quantity $ext\left(\vec{H}(S_e), \vec{G}; \phi\right)$, in $f(d)\tilde{O}(n)$ time. Additionally we know that for each S_e there are at most $f(d)O(n)$ homomorphisms ϕ with non-zero extension.

Initialize G_P as the empty graph. Now, for every $e \in E(P)$, and for every $\phi : I(e) \to \vec{G}$ with non-zero extension, let ϕ_v be the restriction of ϕ to I_v for each $v \in e$. If not present in the graph, create the vertex $u_v = (\phi_v(I_v)\text{-}(v))$ for each $v \in e$. Then add a hyperedge connecting the vertices $\{u_v : v \in e\}$ with weight $ext\left(\vec{H}(S_e), \vec{G}; \phi\right)$. The resulting graph is exactly G_P.

The complexity of this process is dominated by computing the extensions as in Lemma A.1, hence the graph can be constructed in $f(d)\tilde{O}(n)$ time. Also, the number of hyperedges is equal to the number of homomorphisms with non-zero extension, which again using Lemma A.1 will be bounded by $f(d)O(n)$. Finally, the number of vertices added to the graph is at most the number hyperedges times the maximum arity of P, which is bounded by k, and hence it will be bounded also by $f(d)O(n)$. ◀

A.2 Proof of Lemma 3.4

First we show that each colorful copy of P in G_P can be related to $|V(P)|$ homomorphisms that do not disagree in any vertex.

▷ Claim A.2. Let $P' \in \text{Col-}\mathcal{S}(P, G_P)$ with vertex set $\{u_v : v \in V(P)\} = V(P')$. Consider the homomorphisms $\{\phi_v\}$ corresponding to each vertex $u_v \in V(P')$ such that $u_v = (\phi_v(I_v)\text{-}v)$. Every pair of homomorphisms will agree with each other.

Proof. We prove by contradiction. Consider the case that for some $P' \in \text{Col-}\mathcal{S}(P, G_P)$ we have two vertices u_v and $u_{v'}$ for which the corresponding homomorphisms ϕ_v and $\phi_{v'}$ disagree in the value of i, that is, $\exists i \in I_v \cap I_{v'}$ such that $\phi_v(i) \neq \phi_{v'}(i)$.

From Definition 3.1 we have that the vertices $v \in V(P)$ for which $I_v \ni i$ induce a connected hypergraph in P. Let P_i be that hypergraph. Note that $v, v' \in P_i$, hence there must exist a sequence of vertices $\{v, v_1, v_2 ... v'\}$ from v to v' in P_i such that there is a hyperedge in P containing every consecutive pair of the sequence, otherwise, P_i is not connected and we reach a contradiction.

Now, consider the vertices $u_v, u_{v_1}, ..., u_{v'} \in V(P')$ corresponding to each vertex of the sequence. For each consecutive pair $u_{v_l}, u_{v_{l+1}}$ in the sequence, we have that there is a hyperedge e in G_P that contains both u_{v_l} and $u_{v_{l+1}}$. But this implies that the homomorphisms ϕ_{v_l} and $\phi_{v_{l+1}}$ corresponding to those vertices can not disagree in the value of i, as they are both obtained by restricting the same homomorphism $\phi : I(e) \to \vec{G}$ to I_{v_l} and $I_{v_{l+1}}$ respectively.

This implies that for every pair of vertices in the sequence the corresponding homomorphisms must agree. However, this also implies that all the homomorphisms in the sequence must map i to the same vertex, reaching a contradiction. ◀

We can now prove Lemma 3.4. This lemma relates the counts between the original and the reduced graph.

▶ **Lemma 3.4.** $\text{Hom}_{\vec{H}}(\vec{G}) = \textit{Col-WSub}_P(G_P)$

Proof. Let $\text{Col-}\mathcal{S}(P, G_P)$ be the collection of appearances of colorful P in G_P. Because of how we set the colors in the reduced graph, every such appearance will have one vertex in each of the $|V(P)|$ layers of G_P.

Let $\Phi(I^*, \vec{G}) = \{\phi : I^* \to \vec{G}\}$. Every homomorphism from $\vec{H}$ to $\vec{G}$ will extend exactly one of the homomorphisms ϕ in $\Phi(I^*, \vec{G})$. Hence we can write:

$$\text{Hom}_{\vec{H}}(\vec{G}) = \sum_{\phi \in \Phi(I^*, \vec{G})} ext\left(\vec{H}, \vec{G}; \phi\right) \tag{4}$$

Every such homomorphism ϕ in $\Phi(I^*, \vec{G})$ corresponds exactly with a set of homomorphisms $\phi_v \in \phi(I_v, \vec{G})$ for each $v \in V(P)$, such that $\phi = \bigcup_{v \in V(P)} \phi_v$. Note that all such homomorphisms must agree which each other. Moreover, let $\Phi_\times = \bigtimes_{v \in V(P)} \Phi(I_v, \vec{G})$ be the collection of all such possible sets and $\Phi'_\times$ the restriction of $\Phi_\times$ to sets where all the homomorphisms agree with each other. Every set $\{\phi_v\} \in \Phi'_\times$ will correspond with exactly one homomorphism ϕ in $\Phi(I^*, \vec{G})$. Hence we can express the previous expression as follows:

$$\sum_{\phi \in \Phi(I^*, \vec{G})} ext\left(\vec{H}, \vec{G}; \phi\right) = \sum_{\{\phi_v\} \in \Phi'_\times} ext\left(\vec{H}, \vec{G}; \bigcup_{v \in V(P)} \phi_v\right) \tag{5}$$

Now, consider the subgraphs $\vec{H}(S_e)$ for every $e \in E(P)$. Note that these subgraphs only intersect with each other on the vertices of I^*, and $I(e) = I^* \cap \vec{H}(S_e)$. For a hyperedge $e \in E(P)$ and a homomorphism $\phi \in \Phi(I^*, \vec{G})$, let ϕ_e be the restriction of ϕ to the vertices of $I(e)$. We can then write $ext\left(\vec{H}, \vec{G}; \phi\right)$ as the product of extensions $ext\left(\vec{H}(S_e), \vec{G}; \phi_e\right)$. Moreover, when expressing ϕ as a set $\{\phi_v\} \in \Phi'_\times$, every ϕ_e will be equal to the union of ϕ_v for all $v \in e$. Hence for any such set we will have:

$$ext\left(\vec{H}, \vec{G}; \bigcup_{v \in V(P)} \phi_v\right) = \prod_{e \in E(P)} ext\left(\vec{H}(S_e), \vec{G}; \bigcup_{v \in e} \phi_v\right) \tag{6}$$

Combining (4), (5) and (6) we get:

$$\mathrm{Hom}_{\vec{H}}(\vec{G}) = \sum_{\{\phi_v\} \in \Phi'_\times} \prod_{e \in E(P)} ext\left(\vec{H}(S_e), \vec{G}; \bigcup_{v \in e} \phi_v\right) \tag{7}$$

Note that every homomorphisms ϕ_v in the sets of $\Phi'_\times$ corresponds with the vertex $(\phi_v(I_v)\text{-}v)$ of G_P. Additionally, by Definition 3.2 the value of $ext\left(\vec{H}(S_e), \vec{G}; \bigcup_{v \in e} \phi_v\right)$ corresponds with the weight of the hyperedge connecting the vertices $\{(\phi_v(I_v)\text{-}v) : v \in e\}$ of G_P. Thus, we can rewrite the previous expression, by iterating over vertices of the different layers, as long as their corresponding homomorphisms agree with each other. Let $V(\Phi'_\times)$ be the collection of sets of vertices $\{u_v\}$ corresponding to each $\{\phi_v\} \in \Phi'_\times$. We have:

$$\sum_{\{\phi_v\} \in \Phi'_\times} \prod_{e \in E(P)} ext\left(\vec{H}(S_e), \vec{G}; \bigcup_{v \in e} \phi_v\right) = \sum_{\{u_v\} \in V(\Phi'_\times)} \prod_{e \in E(P)} w(\{u_v : v \in e\}) \tag{8}$$

For a set of vertices $\{u_v\}$ to have non-zero total product, we will require that all the hyperedges $\{u_v : v \in e\}$ for $e \in E(P)$ have non-zero weight. Hence, the subgraph of G_P induced by $\{u_v\}$ must contain a copy of P. Additionally, every copy of P in G_P will correspond to some set of vertices $\{u_v\} \in V(\Phi'_\times)$, as by Claim A.2 the vertices of every copy must agree in their corresponding homomorphisms. Given this, together with (7) and (8) we get:

$$\mathrm{Hom}_{\vec{H}}(\vec{G}) = \sum_{\{u_v\} \in V(\Phi'_\times)} \prod_{e \in E(P)} w(\{u_v : v \in e\}) = \sum_{P' \in \text{Col-}\mathcal{S}(P, G_P)} \prod_{e' \in E(P')} w(e') \tag{9}$$

Which by the definition of Col-WSub$_P$ (Equation (1)) is precisely Col-WSub$_P(G_P)$. ◀

A.3 Proof of Lemma 3.6

▶ **Lemma 3.6.** *Let $\mathcal{P}$ be a set of hypergraphs, if for every hypergraph $P \in \mathcal{P}$ there is an algorithm that computes Col-WSub$_P$ in time $\tilde{O}(m^c)$, then for any input graph G with degeneracy κ and any $\mathcal{P}$-computable pattern H, there is an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa)\tilde{O}(n^c)$.*

Proof. First we compute the degeneracy orientation $\vec{G}$ of G, this will take $O(n+m)$ time. Note that the number of edges is bounded by $f(\kappa)O(n)$, the maximum outdegree of $\vec{G}$ will be $d = O(\kappa)$. Let H be a $\mathcal{P}$-computable pattern. Every acyclic orientation $\vec{H} \in \Sigma(H)$ either has a DAG-treewidth of 1 ot it is P-reducible for some hypergraph $P \in \mathcal{P}$. In the first case we can use Theorem 2.2 to compute $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in $f(\kappa)\tilde{O}(n)$ time. In the second case we can use Lemma 1.3 to compute $\mathrm{Hom}_{\vec{H}}(\vec{G})$ in time $f(d)\tilde{O}(n^c)$. We can then aggregate the counts for all the orientations, obtaining $\mathrm{Hom}_H(G)$ in time $f(\kappa)\tilde{O}(n^c)$. ◀

B Proofs of Section 4

B.1 Proof of Lemma 4.1

▶ **Lemma 4.1.** *Every directed acyclic pattern $\vec{H}$ with at most 3 sources is either $\mathcal{C}_3$-reducible or $\tau(\vec{H}) = 1$.*

Proof. First, if the pattern has only 1 or 2 sources, then one can construct a DAG-tree decomposition T of $\vec{H}$ with $\tau = 1$ by putting every source in its own node of T.

Now we consider the case where $\vec{H}$ has 3 sources. Let s_1, s_2 and s_3 be the three sources. Consider the case where there is a pair of sources that do not intersect with each other, let wlog s_1, s_3 be those sources, that is, $\mathit{Reach}(s_1) \cap \mathit{Reach}(s_3) = \emptyset$. We can construct a DAG-tree decomposition of $\vec{H}$ by putting every source in its own bag an putting s_2 in between s_1 and s_3. This will be a valid DAG-tree decomposition with $\tau = 1$.

Otherwise, every pair of sources intersect each other. Let $S_i = \{s_i\}$ and set $I_i = \vec{H}(S_i) \cap \vec{H}(S_{i+1})$. We show that these sets give a valid $\mathcal{C}_3$-reduction. First, the two sets are valid sets, the source sets are non-overlapping and have $\tau = 1$ (as there is only one source in each set). While the intersections are all adjacent to each other, so intersection sets containing the same vertex will form a continuous sequence. Also, we can check that the two conditions from Definition 3.1 are satisfied directly by how we constructed the sets. Hence the pattern will be 3-reducible[4]. ◀

B.2 Proof of Lemma 4.2

▶ **Lemma 4.2.** *Every directed acyclic pattern $\vec{H}$ with at most 3 intersection vertices is either $\mathcal{C}_3$-reducible or $\tau(\vec{H}) = 1$.*

Proof. If there are 1 or 2 intersection vertices then one can construct a valid DAG-tree decomposition T with $\tau(T) = 1$ by finding a source that reaches all the intersection vertices (one must exist or the pattern is not connected) and setting it as the root of T and then connecting all the other sources directly to the root as singleton bags. For every pair of sources, their intersection will be a subset of the vertices reachable by the root and hence we will have a valid DAG-tree decomposition.

If we have 3 intersection vertices we need to consider the following cases. If there is a source that can reach the 3 intersection vertices then the previous construction will give a valid DAG-tree decomposition with $\tau = 1$. Hence, we consider the case where every source at most reach 2 of the intersection vertices. Let $\{u, v, w\} = I(\vec{H})$ be the three intersection vertices, and consider the three pairs u-v, v-w, u-w. For every source $s \in S(\vec{H})$ we will have that s either reaches one of the intersection vertices or one of the three pairs. If only two

[4] Note that it is possible that some of these patterns also have $\tau = 1$.

of the pairs are reachable by the sources then we can construct a DAG-tree decomposition with $\tau = 1$: Let u-v, v-w be the two pairs that appear, and let s_1 and s_2 be two sources with $u, v \in \mathit{Reach}(s_1)$ and $v, w \in \mathit{Reach}(s_2)$, set s_1 as the root of the tree and s_2 as one of its children, connect every other source to either s_2 if they can reach w and to s_1 otherwise.

If the three pairs are reachable, then we can show a valid $\mathcal{C}_3$-reduction. Let s_1, s_2, s_3 be three sources reaching a different pair of intersection vertices, specifically, let $u, v \in \mathit{Reach}(s_1)$, $v, w \in \mathit{Reach}(s_2)$ and $u, w \in \mathit{Reach}(s_3)$. We put s_1 in S_1, s_2 in S_2 and $s_3 \in S_3$, for every other source, if they reach two intersection vertices then we put it in the set which contains a source reaching the same two vertices, if they reach only one intersection vertex then we arbitrarily add it to one of the two sets of sources that reach that vertex. We set $I_1 = \{v\}$, $I_2 = \{w\}$ and $I_3 = \{u\}$. We can verify that this is a valid $\mathcal{C}_3$-reduction, as it will satisfy all the properties from Definition 3.1. ◀

B.3 Proof of Lemma 4.3

▶ **Lemma 4.3.** *Every 6 and 7-vertex undirected pattern H is $\mathcal{C}_3$-computable.*

Proof. Fix a pattern H, any acyclic orientation $\vec{H} \in H$ will either have at most 3 sources or 3 intersection vertices. In the first case we can use Lemma 4.1 to show that $\vec{H}$ is $\mathcal{C}_3$-reducible or has $\tau = 1$, similarly in the second case we can use Lemma 4.2. Hence, all acyclic orientations of H are either $\mathcal{C}_3$-reducible or have $\tau = 1$, which implies that H is $\mathcal{C}_3$-computable. ◀

B.4 Proof of Lemma 4.5

▶ **Lemma 4.5.** *For all $k \geq 3$, $\mathcal{C}_{2k}$ and $\mathcal{C}_{2k+1}$ are $\mathcal{C}_k$-computable.*

Proof. Fix any k, we want to show that all acyclic orientations of $\mathcal{C}_{2k}$ and $\mathcal{C}_{2k+1}$ have $\tau = 1$ or are $\mathcal{C}_l$-reducible for some $l \leq k$. Note that any such orientation can have at most k sources. If the pattern has either 1 or 2 sources, then we can construct a DAG-tree decomposition of width one by putting the sources in their own bags. If an orientation $\vec{H}$ has $l \geq 3$ sources then we can show that it is $\mathcal{C}_l$-reducible:

Let $s_0, ..., s_{l-1}$ be the l sources of $\vec{H}$, ordered in either clockwise or anti-clockwise. For $i \in [l]$, set $S_i = \{s_i\}$ and $I_i = \vec{H}(S_i) \cap \vec{H}(S_{i+1})$, note that these sets are always non-empty, as consecutive sources in a cycle intersect in a vertex. Additionally a source s_i will only reach two intersection vertices, the one with s_{i-1} and the one with s_{i+1}. Therefore, these sets will satisfy all the conditions in Definition 3.1, giving that $\vec{H}$ is $\mathcal{C}_l$-reducible. ◀

B.5 Proof of Lemma 4.6

We start by proving some auxiliary lemmas.

▶ **Lemma B.1.** *Given two directed patterns $\vec{P}, \vec{P}'$ with $\tau(\vec{P}) = \tau(\vec{P}') = 1$, the graph $\vec{P}''$ obtained by either merging a vertex from $\vec{P}$ with a vertex from $\vec{P}'$ or two vertices u, v from $\vec{P}$ connected by an edge with two vertices u', v' from $\vec{P}'$ connected by an edge will also have $\tau(\vec{P}'') = 1$.*

Proof. Consider a DAG-tree decomposition T of $\vec{P}$ and a DAG-tree decomposition T' of $\vec{P}'$ with $\tau(T) = \tau(T') = 1$. Let $r \in T$ be any bag with $u \in \mathit{Reach}(r)$ (note that in the case of merging two vertices v will also be in $\mathit{Reach}(r)$), similarly let $r' \in T'$ be any bag with $u' \in \mathit{Reach}(r')$. Now we consider the following scenarios:

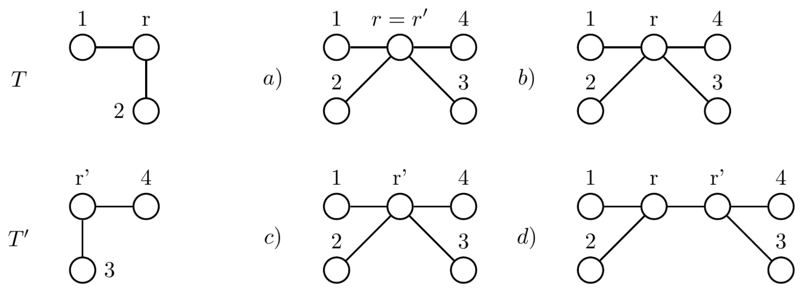

Figure 6 The possible cases in Lemma B.1. T and T' represent DAG-tree decomposition of the graphs $\vec{P}$ and $\vec{P}'$, with $u \in \mathit{Reach}_{\vec{P}}(r)$ and $u' \in \mathit{Reach}_{\vec{P}'}(r')$.

a) u is a source in $\vec{P}$ and u' is a source in $\vec{P}'$: In this case we have that $r = u$ and $r' = u'$. Additionally, in $\vec{P}''$ the merged vertex u will still be a source. Hence we can form a valid DAG-tree decomposition T'' of $\vec{P}''$ with $\tau = 1$ by merging the bags r and r' into a single bag.
b) u is a source in $\vec{P}$ but u' is not a source: In this case $r = u$, and in $\vec{P}''$ the merged vertex will not be a source, however r' will still be one. We can create a valid DAG-tree decomposition T'' of $\vec{P}''$ by replacing the bag containing r in T by the bag containing r' and merging it with the r' of T'. Note that all the original sources in $\vec{P}$ (except u) are still sources reaching the same vertices, while r' now reaches the same vertices of P that were reachable by u, hence all the reachability conditions will be satisfied.
c) u is not a source in $\vec{P}$ but u' is a source in $\vec{P}'$: This case is analogous to the previous.
d) Both u and u' were not sources in $\vec{P}$ and $\vec{P}'$ respectively: In this case we can construct a valid DAG-tree decomposition T'' by connecting r and r' with an edge. In order for this to not be a valid DAG-tree decomposition, we would need a source $s \neq r \in \vec{P}$ and a source $s' \neq r' \in \vec{P}'$ such that $\mathit{Reach}(s) \cap \mathit{Reach}(s') \not\subseteq \mathit{Reach}(r)$ or $\mathit{Reach}(s) \cap \mathit{Reach}(s') \not\subseteq \mathit{Reach}(r')$. Note that $\mathit{Reach}(s) \cap \mathit{Reach}(s')$ can at most contain only the merged vertex u and all the vertices reachable by it, but both r and r' will also reach those vertices, hence the DAG-tree decomposition is valid.

Fig. 6 shows the construction of the DAG-tree decomposition in every case. ◀

▶ **Lemma B.2.** *Given any $\mathcal{C}_k$-computable undirected pattern H, and an undirected connected pattern H' with $LICL(H') < 6$, the graphs resulting of the following operations are $\mathcal{C}_k$-computable:*

- *Merge any vertex $u \in H$ with any vertex $u' \in H'$.*
- *Combine any edge $(u, v) \in E(H)$ with any edge $(u', v') \in E(H')$ by merging u with u' and v with v', and removing the duplicate edge.*

Proof. Let H'' be the graph resulting of any of the operations. Consider any acyclic orientation $\vec{H}'' \in \Sigma(H'')$, let $\vec{H}$ and $\vec{H}'$ be the subgraphs of $\vec{H}''$ induced by the vertices of H and H' respectively, note that $\vec{H}$ and $\vec{H}'$ are themselves acyclic orientations of H and H' respectively. Because $LICL(H') < 6$ we have that $\tau(\vec{H}') = 1$.

Now consider the pattern $\vec{H}$, we have two cases:

- $\tau(\vec{H}) = 1$. In this case we can directly invoke Lemma B.1, giving that the merged directed pattern $\vec{H}''$ also has $\tau(\vec{H}') = 1$.

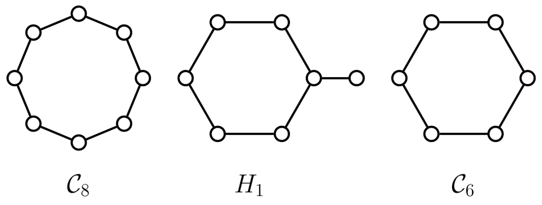

Figure 7 The spasm of $\mathcal{C}_8$, only including patterns with LICL greater than 5.

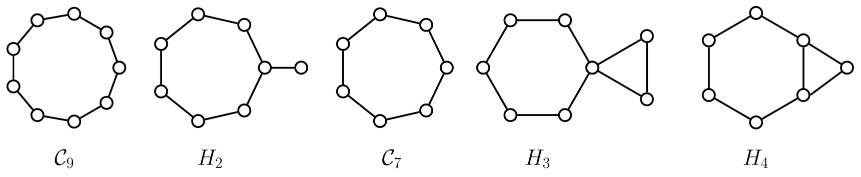

Figure 8 The spasm of $\mathcal{C}_9$, only including patterns with LICL greater than 5.

- $\vec{H}$ is $\mathcal{C}_l$-reducible, for some $l \leq k$. Let $S_1, ..., S_l$ be the sources of the $\mathcal{C}_l$-reduction of $\vec{H}$. There must exist a set of sources S_i that can reach the merged vertex/vertices. We define the following source sets for $\vec{H}''$: For $j \neq i$ we set $S''_j = S_j$ and for the remaining set of sources we set $S''_i = (S_i \cap S(\vec{H}'')) \cup S(\vec{H}')$. Note that if the merged vertex was a source in $\vec{H}$ it is possible that it is no longer a source in $\vec{H}''$. We set $I''_i = \vec{H}(S_i) \cap \vec{H}(S_{i+1})$. If s_i is still a source in $\vec{H}''$ then we set $s''_i = s_i$, otherwise there must be a source s'' that can reach the old s_i in $\vec{H}''$. Such source will also reach all the vertices in I_{i-1} and I_i. We can see that this is a valid $\mathcal{C}_l$-reduction: The intersection will be valid as the only changes are that I''_i and I''_{i-1} might include additional vertices from H'. The source sets are also valid as all the sources will be part of exactly one source set. The other conditions in Definition 3.1 will follow from our construction.

◀

Now we have all the tools to prove Lemma 4.6. We break the result into different lemmas. The result for $\mathcal{C}_6$ and $\mathcal{C}_7$ follows directly from Lemma 4.3, as every pattern in those spasms will have at most 7 vertices. We prove next the result for $\mathcal{C}_8$ and $\mathcal{C}_9$.

▶ **Lemma B.3.** *All the patterns in Spasm($\mathcal{C}_8$) and Spasm($\mathcal{C}_9$) are $\mathcal{C}_4$-computable.*

Proof. Fig. 7 shows all the patterns in the spasm of $\mathcal{C}_8$ with LICL greater than 5, similarly, Fig. 8 shows all the patterns in the spasm of $\mathcal{C}_9$ with LICL greater than 5. From Lemma 2.3 we have that the rest of the patterns in the spasms will have a DAG-treewidth of 1 in all their orientations. From Lemma 4.5 we have that all the cycle patterns in both spasms will be $\mathcal{C}_4$ or $\mathcal{C}_3$-computable.

Using Lemma 4.3 we have that the patterns H_1 and H_4 are $\mathcal{C}_3$-computable, as they only have 7 vertices. For H_2 and H_3 we can use Lemma B.2 to show that they are also $\mathcal{C}_3$-computable, H_2 is formed by combining a $\mathcal{C}_7$ with an edge, and H_3 is formed by combining a $\mathcal{C}_6$ with a $\mathcal{C}_3$. ◀

Finally, we prove the result for $\mathcal{C}_{10}$.

▶ **Lemma B.4.** *All the patterns in Spasm($\mathcal{C}_{10}$) are $\mathcal{C}_5$-computable.*

Proof. Fig. 9 shows all the patterns in the spasm of $\mathcal{C}_{10}$ with *LICL* greater than 5. We verify that all the patterns are $\mathcal{C}_5$-computable:

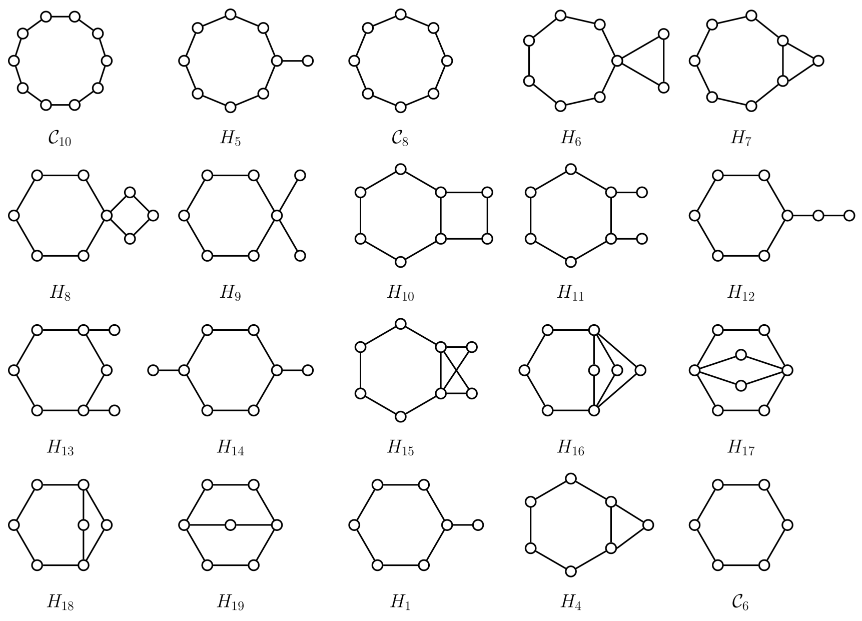

Figure 9 The spasm of $\mathcal{C}_{10}$, including only patterns with LICL greater than 5.

- From Lemma 4.3, we have that patterns H_1, H_4, H_{18}, H_{19} are $\mathcal{C}_3$-computable, as they have 7 vertices.
- From Lemma 4.5 we have that $\mathcal{C}_{10}, \mathcal{C}_8, \mathcal{C}_6$ are $\mathcal{C}_5$, $\mathcal{C}_4$ and $\mathcal{C}_3$-computable respectively.
- From Lemma B.2 we have that H_5 is $\mathcal{C}_4$-computable, as it is obtained by combining a $\mathcal{C}_8$ with a 2-path.
- Again using Lemma B.2 we can show that the following patterns are $\mathcal{C}_3$-computable:
  - H_6: $\mathcal{C}_7$ + $\mathcal{C}_3$ along a vertex.
  - H_7: $\mathcal{C}_7$ + $\mathcal{C}_3$ along an edge.
  - H_8: $\mathcal{C}_6$ + $\mathcal{C}_4$ along a vertex.
  - H_9: H_1 + 2-path along a vertex.
  - H_{10}: $\mathcal{C}_6$ + $\mathcal{C}_4$ along an edge.
  - H_{11}: H_1 + 2-path along a vertex.
  - H_{12}: H_1 + 2-path along a vertex.
  - H_{13}: H_1 + 2-path along a vertex.
  - H_{14}: H_1 + 2-path along a vertex.
  - H_{15}: H_4 + $\mathcal{C}_3$ along an edge.

  Only left is to show that H_{16} and H_{17} are also $\mathcal{C}_5$-computable:

- First, we look at H_{17}, consider any acyclic orientation of it, if it has 3 sources or less then using Lemma 4.1 we know that the either $\tau = 1$ or the oriented pattern is $\mathcal{C}_3$-reducible. Hence, we consider the orientations with 4 or more sources, the only option to have 4 sources is by having both central vertices as sources, and then one of the two top vertices as source and one of the two bottom vertices also as source. One can verify that all the resultant orientations will have a $\tau = 1$, this can be seen as the two central sources will reach the same vertices and the top and bottom source can only intersect in vertices also reachable by the central ones.

- Now we look at H_{16}, consider any acyclic orientation. If it has 3 sources again we can use Lemma 4.1. Otherwise if it has 5 sources it will have at most 3 intersection vertices, which by Lemma 4.2 indicated that the pattern has $\tau = 1$ or it is $\mathcal{C}_3$-reducible. Now consider the orientations with 4 sources, it must happen that at least 2 of the sources are in the right side (the triple 2-path), but such sources will reach the same intersection vertices. Meaning that they can be put in the same set of sources, obtaining a $\mathcal{C}_3$-reducible pattern. ◀

B.6 Proof of Lemma 4.7

▶ **Lemma 4.7.** *For all $6 \leq k \leq 10$, there is an algorithm that computes* $\mathrm{Sub}_{\mathcal{C}_k}(G)$ *in time* $f(\kappa)O(n^{d_{\lfloor k/2 \rfloor}})$.

Proof. From Lemma 4.6 we have that for all $6 \leq k \leq 10$ all patterns in $\mathit{Spasm}(\mathcal{C}_k)$ are $\mathcal{C}_{\lfloor k/2 \rfloor}$-computable. From Lemma 1.5 we have that we can compute Col-WSub$_{\mathcal{C}_k}$ in time $O(m^{d_k})$. Hence we can use Lemma 3.6 to compute $\mathrm{Hom}_H(G)$ in time $f(\kappa)O(n^{d_{\lfloor k/2 \rfloor}})$ for all patterns in the spasm of $\mathcal{C}_k$, then we can obtain $\mathrm{Sub}_H(G)$ using inclusion-exclusion. ◀

C Proofs of Section 5

C.1 Proof of Lemma 1.4

For reference we restate the definition of $\mathcal{P}_{i,s}$ and $\mathcal{P}_k$.

▶ **Definition 5.1.** *[$\mathcal{P}_{i,s}$, $\mathcal{P}_k$] We define $\mathcal{P}_{i,s}$ as the set of hypergraphs P with i vertices and s hyperedges such that:*

1. *Every vertex has degree at least 2.*
2. *Every hyperedge contains at least 2 vertices.*
3. *No hyperedge is a subset of any other hyperedge.*
4. *For every pair of distinct vertices $u, v \in V(P)$ the set of hyperedges containing u can not be equal or a subset of the set of hyperedges containing v.*

For any $k \geq 7$, we define $\mathcal{P}_k$ recursively as the union of $\mathcal{P}_{k-1}$ and all sets $\mathcal{P}_{i,s}$, with $i + s = k$ and $i, s \geq 4$, with $\mathcal{P}_6 = \{\mathcal{C}_3\}$.

We can show that if we have a graph P satisfying the first three conditions definition, then every P-reducible pattern can also be reduced to some graph in $\mathcal{P}_{i,s}$, for some i and s.

▶ **Lemma C.1.** *Let P be a hypergraph with $i \geq 4$ vertices and $s \geq 4$ hyperedges such that:*

1. *Every vertex has degree at least 2.*
2. *Every hyperedge contains at least 2 vertices.*
3. *No hyperedge is a subset of any other hyperedge.*

Then, for some $i' \leq i$, there exists a hypergraph $P' \in \mathcal{P}_{i',s}$ such that every directed pattern $\vec{H}$ that is P-reducible is also P'-reducible.

Proof. Let $E(u)$ be the set of hyperedges of P that contain the vertex u, and let R be the sum of arities of all hyperedges in P.

If P already satisfies the fourth condition of Definition 5.1 then $P \in \mathcal{P}_{i,s}$ and we are done. Otherwise, there is a pair $u, v \in V(P')$ such that $E(u) \subseteq E(v)$. We show that in that case we can construct a graph P' with $|V(P')| \leq i$ and $E(P') = s$ that still satisfies the initial three constraints and has sum of arities $R' < R$, and for which every pattern that is P-reducible is also P'-reducible:

We have $E(u) \subseteq E(v)$, we distinguish two cases:

- If $E(u) = E(v)$ then we can just merge the vertices u and v into a single vertex uv, every hyperedge that contained both u and v will just contain uv instead. P' has the same number of edges than P but one vertex less, the total arity R of P' will also be less than the one of P. Only left is to show that every P-reducible pattern $\vec{H}$ is also P'-reducible. Let $\{S_e : e \in E(P)\}$ and $\{I_v : v \in V(P)\}$ be the sets of sources and intersection vertices of $\vec{H}$ that achieve P-reducibility, setting $I_{uv} = I_u \cup I_v$ and removing I_u and I_v while maintaining all the other intersection and source sets the same will achieve P'-reducibility: for any of the vertices in $I(\vec{H})$ the connectivity condition from Definition 3.1 will not be affected as uv will neighbor all the vertices that u and v already neighbor in P. Our construction guarantees that the two final conditions from Definition 3.1 are satisfied.
- If $E(u) \subset E(v)$ then we have that every edge containing u also contains v. We construct P' by adding the vertex uv and removing v, for every hyperedge that contained both u and v we instead only contain uv, except for a single hyperedge e that will contain both u and uv. P' will have the same number of vertices and hyperedges than P but the total arity will reduce by at least 1. We show that every P-reducible pattern $\vec{H}$ is also P'-reducible.
  Let $\{S_e : e \in E(P)\}$ and $\{I_v : v \in V(P)\}$ be the sets of sources and intersection vertices of $\vec{H}$ that achieve P-reducibility. We set $I_{uv} = I_u \cup I_v$ and remove the set I_v while maintaining all the other sets the same. We can show that the new sets will achieve P'-reducibility: the edge e that connects both uv and v guarantees that the intersection vertices in I_u and I_v will still satisfy the connectivity condition from Definition 3.1. Our construction guarantees that the two final conditions from Definition 3.1 are also satisfied.

If P' satisfies the last condition from Definition 5.1 then it must be the case that $P' \in \mathcal{P}_{i',s}$ for some $i' \leq i$. Otherwise another pair of vertices must be violating the fourth condition, and we can then repeat the previous process. Eventually we are guaranteed to reach a graph P' which satisfies all the conditions and it is in $\mathcal{P}_{i',s}$ for some $i' \leq i$. This is guaranteed because the value of R will decrease in every iteration, but it can not be lower than $2s$ (twice the number of hyperedges). ◀

We can now prove Lemma 1.4.

▶ **Lemma 1.4.** *Let $k \geq 6$, every connected pattern H with k vertices is $\mathcal{P}_k$-computable.*

Proof. Let $\vec{H}$ be any directed acyclic graph with k vertices. If $\vec{H}$ admits a DAG-tree decomposition with $\tau = 1$ then we are done, if $\vec{H}$ has at most 3 sources or 3 intersection vertices then $\vec{H}$ is $\mathcal{C}_3$-reducible and we are also done.

Otherwise, $\vec{H}$ has at least 4 sources and 4 intersection vertices. Let $S = S(\vec{H})$ be the set of sources of $\vec{H}$. Initialize sets $S_e = \{s_e\}$ for every source $s_e \in S$, we also construct trees of sources $\mathcal{T}_e$ for each source s_e containing a single node s_e. Let I^* be the set of intersection vertices reachable by two distinct sets S_e, S_e'. For every set S_e, let $I(e)$ be the set of intersection vertices in I^* reachable by S_u.

If there exists $e \neq e' \in S$ such that $I(e) \subseteq I(e')$, then set $S_{e'} = S_e \cup S_{e'}$ and delete S_e, combine the trees $\mathcal{T}_e$ and $\mathcal{T}_{e'}$ by adding an edge between s_e and $s_{e'}$, and update the set I^*. Repeat until we obtain $s \leq |S|$ sets of sources such that there are no $e \neq e'$ such that $I(e) \subseteq I_{e'}$. Let S^* be the collection of the remaining sets. For every set $S_e \in S^*$, the graph $\vec{H}(S_e)$ has a DAG-treewidth of 1, as $\mathcal{T}_e$ is a valid DAG-tree decomposition decomposition of $\vec{H}(S_e)$.

Consider the hypergraph P where every intersection vertex in I^* is a vertex, and every source set S_e is a hyperedge e containing the vertices in $I(e)$.

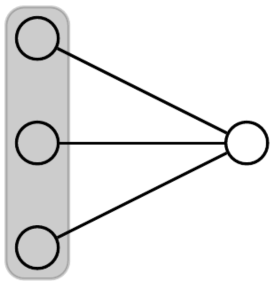

Figure 10 The hypergraph $\mathcal{H}_1$. Formed by one hyperedge of arity 3 (in gray) and three normal edges.

We can show that $\vec{H}$ is P-reducible. For each edge $e \in E(P)$ we use S_e as the corresponding source of sets, and for each vertex $v \in V(P)$ corresponding to the intersection vertex $v' \in I^*$ we set $I_v = \{v'\}$. We can verify the conditions from Definition 3.1:

- For every vertex $v \in V(P)$ and every edges $e \ni v$ we have that $v' \in I(e)$, which implies that $I_v \subseteq I(e)$.
- For every hyperedge $e \in V(P)$ we have that e contains all the vertices v for which $v' \in I(e)$. Hence $\bigcup_{v \in e} I_v = I(e)$.

We show that P must satisfy the first three conditions from Definition 5.1:

1. By construction, every vertex in I^* must be reachable by at least two distinct sets, hence the corresponding vertex in P will have a degree of at least 2.
2. Every set S_e must reach at least two different intersection vertices in I^*, otherwise there we could combine it with a different set of sources, with other set of sources that reaches the same intersection vertex.
3. A hyperedge e being a subset of another hyperedge e' would imply $I(e) \subseteq I_{e'}$, which is not allowed by the construction.

Let i be the number of vertices in P and s the number of edges. Note that $s \geq 3$, otherwise the original pattern $\vec{H}$ has DAG-treewidth of 1, this implies $i \geq 3$, or the construction would violate one of the previous conditions. If $s = 3$ then $i = 3$ and the $P = \mathcal{C}_3$ which means $\vec{H}$ is $\mathcal{C}_3$-reducible.

Otherwise we have that $i, s \geq 4$ and $i + s \leq k$. Using Lemma C.1 we have that for some $i' \leq i$ there is a pattern $P' \in \mathcal{P}_{i',s}$ such that $\vec{H}$ will be P'-reducible, and we have that $\mathcal{P}_{i',s} \subseteq \mathcal{P}_k$. ◀

C.2 Proof of Lemma 5.2

From the definition of $\mathcal{P}_9$ we have that:

$$\mathcal{P}_9 = \mathcal{P}_8 \cup \mathcal{P}_{4,5} \cup \mathcal{P}_{5,4} = \{\mathcal{C}_3\} \cup \mathcal{P}_{4,4} \cup \mathcal{P}_{4,5} \cup \mathcal{P}_{5,4}$$

We will show which patterns form each of these sets. First, we can show that $\mathcal{P}_{4,4}$ is formed by the 4-cycle, the 3-simplex and $\mathcal{H}_1$ (Fig. 10).

▷ Claim C.2. $\mathcal{P}_{4,4} = \{\mathcal{C}_4, \mathcal{S}_3, \mathcal{H}_1\}$

Proof. Let P be a hypergraph in $\mathcal{P}_{4,4}$. Note that P can not have any hyperedge of arity 4, otherwise all the other hyperedges would be subsets of it, breaking condition 3 in Definition 5.1. Hence P may contain only hyperedges with arity 2 and 3, consider the following cases:

- P has 4 hyperedges of arity 2: We can show that in this case P must be the 4-cycle. If P were to contain a triangle then we have that the vertex not in the triangle will have a degree of at most 1, this contradicts condition 1 in Definition 5.1 and hence P must not contain a triangle. The only graph with 4 vertices and 4 edges that does not contain a triangle is the 4-cycle.
- P has 3 hyperedges of arity 2 and 1 of arity 3: Let e be the hyperedge of arity 3 and v the vertex not in e. Every vertex in e must be part of an additional edge in order to have degree 2, the only vertex they can connect without violating condition 3 in Definition 5.1 is v, hence the three edges with arity 2 will connect v with each of the three vertices in e, giving the pattern $\mathcal{H}_1$ seen in Fig. 10.
- P has 2 hyperedges of arity 2 and 2 of arity 3: The two hyperedges of arity 3 must intersect in two vertices, let u, u' be those vertices and v, v' the two remaining vertices. The remaining two edges of arity 2 can not contain u or u' as they would be subsets of one of the hyperedges of arity 3. Hence only the other two vertices v, v' can be part of an edge. But this only gives one possible edge, hence we can not have a graph $P \in \mathcal{P}_{4,4}$ with this configuration.
- P has 1 hyperedges of arity 2 and 3 of arity 3: The two hyperedges of arity 3 must intersect in two vertices, let u, u' be those vertices and v, v' the two remaining vertices. The third hyperedge of arity 3 can not contain both u and u' so it will contain v, v' and either u or u'. But this means that every pair of vertices is already part of some hyperedge, and we can not have an extra edge of arity 2 without violating the condition 3 in Definition 5.1.
- P has 4 hyperedges of arity 3: There are 4 possible hyperedges of arity 3 in a graph with 4 vertices, hence P must contain all of them, the result is the simplex of arity 3, $\mathcal{S}_3$.

◀

$\mathcal{P}_{4,5}$ only contains the diamond graph $\mathcal{D}$, also known as K_4 minus one edge.

▷ Claim C.3. $\mathcal{P}_{4,5} = \{\mathcal{D}\}$.

Proof. We first can show that any graph $P \in \mathcal{P}_{5,4}$ may not contain any hyperedge of arity greater than 2: If it were to contain a hyperedge of arity 4 then all the other hyperedges would be subset of it, breaking condition 3 in Definition 5.1. Also P can not contain 5 hyperedges of arity 3 as there are at most 4 such hyperedges with 4 vertices. Hence P must contain at least an edge of arity 2. We consider the rest of cases:

- If P contains 3 or 4 hyperedges of arity 3 then every pair of vertices in P will be together in one of the hyperedges and we can not add any edges of arity 2. Hence this can not happen.
- If P contains 2 hyperedges of arity 2, then there is only one pair of vertices that is not together in some hyperedge, but need to add 3 edges.
- If P contains only 1 hyperedge of arity 3, we can add an edge connecting every vertex in the hyperedge with the free vertex, but we will still be missing one edge.

Therefore, P can only contain edges of arity 2. The only pattern with 4 vertices and 5 edges is the diamond or K_4 minus one edge. ◀

Finally, $\mathcal{P}_{5,4}$ only contains the hypergraph $\mathcal{H}_2$ (Fig. 11).

▷ Claim C.4. $\mathcal{P}_{5,4} = \{\mathcal{H}_2\}$.

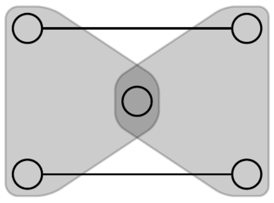

Figure 11 The hypergraph $\mathcal{H}_2$. Formed by two hyperedges of arity 3 (in gray) and two normal edges.

Proof. Let P be a hypergraph with 5 vertices and 4 hyperedges that satisfies all the conditions in Definition 5.1. We prove that $P = \mathcal{H}_2$. P can not contain a hyperedge with arity 5, as then the other hyperedges would necessarily be subsets of it.

Suppose P contains an arity 4 hyperedge, let e be such hyperedge, every vertex in e must be part of at least another hyperedge to have degree of at least 2, also, for every pair of vertices $u, u' \in e$ the set of edges containing u can not be a subset or equal to the set of edges containing u'. However we only have 3 remaining hyperedges (lets call them $\{e_1, e_2, e_3\}$), we can create at most 3 subsets of the remaining hyperedges that will not be subset of each other, either $\{e_1\}, \{e_2\}, \{e_3\}$ or $\{e_1, e_2\}$,$\{e_1, e_3\}$,$\{e_2, e_3\}$, we can assign each subset to one of the vertices but the forth vertex will necessarily create a conflict either by being contained by a subset or a superset of those groups of hyperedges.

Hence, P can not contain an arity 4 hyperedge. Neither can P contain only arity 2 hyperedges, as in that case the average degree (and hence the minimum degree) of P would be below 2. Hence P must be formed by arity 2 and 3 hyperedges. Similarly if P contains 1 arity 3 hyperedge and 3 arity 2 edges we will have the average degree is below 2. We show that P can not contain two arity 3 hyperedges that intersect in more than one vertex:

Consider otherwise, there are two hyperedges e, e' intersecting in u, u'. Let $e = \{u, u', v\}$ and $e' = \{u, u', v'\}$. We have two more hyperedges. Both hyperedges must include the additional vertex w in order for it to have a degree of 2. We need u and u' to be part of at least some hyperedge without the other to avoid breaking the fourth condition of Definition 5.1, hence one of the remaining hyperedges must contain $\{u, w\}$ and the other $\{u, w\}$. However, to avoid breaking the condition we need both v and v' to be in some hyperedge without u and without u', we can achieve this for v by adding v to the previous hyperedges, but then we will not have any additional hyperedge connecting v', as the maximum arity is 3. Therefore, P can not contain such hyperedges.

If P contains 3 or more arity-3 hyperedges then at least two of them will intersect in more than one vertex. Thus, P must contain 2 hyperedges with arity 2 and two more with arity 3. The only hypergraph of that form that satisfies all constraints is $\mathcal{H}_2$. ◀

Combining the three previous claims gives Lemma 5.2.

C.3 Proof of Lemma 5.3

Let $\mathcal{S}_r$ be the r-simplex. We can show that every pattern that is $\mathcal{S}_r$-reducible is also $\mathcal{C}_{r+1}$-reducible. See Fig. 12 for an example.

▶ **Lemma C.5.** *For all $r \geq 3$, every $\mathcal{S}_r$-reducible pattern is $\mathcal{C}_{r+1}$-reducible.*

Proof. In this proofs all the indexes are given in module $r + 1$.

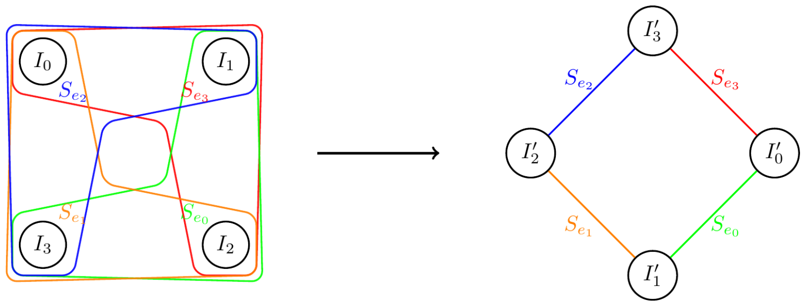

Figure 12 An example of apply Lemma C.5 to the 3-simplex, which becomes a 4-cycle. Here $I'_i = I_{i+1} \cup I_{i+2}$.

Let $\vec{H}$ be a $\mathcal{S}_r$-reducible pattern, for each vertex $v \in V(P)$ let I_s be the sets of intersection vertices that achieve the $\mathcal{S}_r$-reducibility. Similarly for each $e \in E(P)$ let S_e be the set of sources that achieve the $\mathcal{S}_r$-reducibility.

Arbitrary sort the $r+1$ vertices of $\mathcal{S}_r$ $v_0, ..., v_r$, and let $e_i = \{V(\mathcal{S}_r) \setminus \{v_i\}\}$. We define $I(e)$ as the set of intersection vertices reached by the set of sources S_e in $\vec{H}$, hence we will have $I(e_i) = \bigcup_{j \neq i} I_j$.

We define $r+1$ new intersection sets as follows, $I'_i = \bigcup_{j=i+1}^{r-1+i} I_i$ for $i \in [0,4]$. Note that for all i, $I'_i \cup I'_{i+1} = I(e_i)$ and both $I'_i \subseteq I(e_i)$ and $I'_i \subseteq I(e_{i-1})$. Hence we can use the original source sets S_e with the new intersection sets I'_v to achieve $\mathcal{C}_{r+1}$-reducibility, assigning I'_i to the i-th vertex of the cycle and S_{e_i} to the edge connecting the i and $i+1$-th vertices. ◀

We can now prove Lemma 5.3.

▶ **Lemma 5.3.** *If a pattern H is $\mathcal{P}_9$-computable, then it is also $\mathcal{P}_9^*$-computable*

Proof. Let H be a $\mathcal{P}_9$-computable pattern. Consider any of its orientations $\vec{H}$:

- If $\vec{H}$ is P-reducible for some $P \in \mathcal{P}_9 \setminus \{\mathcal{S}_3\}$ then we have that $P \in \mathcal{P}_9^*$.
- Otherwise, $\vec{H}$ is $\mathcal{S}_3$-reducible, but in that case using Lemma C.5 we have that $\vec{H}$ is $\mathcal{C}_4$-reducible, which is in $\mathcal{P}_9^*$.

Hence, all the orientations of H reduce to some pattern in $\mathcal{P}_9^*$, and hence H will be $\mathcal{P}_9^*$-computable. ◀

C.4 Proof of Lemma 5.4

From Lemma 1.5 we have for $\mathcal{C}_3$ and $\mathcal{C}_4$, we can compute Col-WSub in time $O(m^{d_3})$ and $O(m^{d_4})$ respectively. We verify that for the remaining three graphs in $\mathcal{P}_9^*$ $(\mathcal{D}, \mathcal{H}_1, \mathcal{H}_2)$ we can compute Col-WSub in $\tilde{O}(m^{5/3})$ time.

▶ **Lemma C.6.** *There is an algorithm that for any colored, weighted input graph G with m edges computes Col-WSub$_{\mathcal{H}_1}(G)$ in $O(m^{5/3})$ time.*

Proof. Let v be the vertex of $\mathcal{H}_1$ that is not part of the arity-3 hyperedge. Set $\Delta = m^{1/3}$. We can split the vertices of $V^{(v)}(G)$ into two groups, depending on their degree: let $V^H = \{v \in V^{(v)}(G) : d(v) > \Delta\}$ and $V^L = \{v \in V^{(v)}(G) : d(v) \leq \Delta\}$. Every copy of $\mathcal{H}_1$ in G must contain a vertex in either V^L or V^H. We show how to list all copies of P:

- For every vertex $u \in V^H$ we can iterate over every hyperedge of arity 3 in G and verify if u connects with each of the vertices in the hyperedge and they have the right colors.

There are at most $O(\frac{m}{\Delta})$ vertices in V^H. Thus this step will take $O(m\frac{m}{\Delta}) = O(m^{5/3})$ total time.

- For every vertex $u \in V^L$ we can list all triplet of edges connecting u to vertices in each of the other three layers, forming a 3-star and then check if there is a hyperedge that contains the three endpoints of the star. There will be at most Δ such edges, hence we can list all combinations in time $O(m\Delta^2) = O(m^{5/3})$.

For each copy of $\mathcal{H}_1$ that we find we can just multiply the edge/hyperedge weights and aggregate the products to obtain the value of Col-WSub$_{\mathcal{H}_1}(G)$. ◀

▶ **Lemma C.7.** *There is an algorithm that for any colored, weighted input graph G with m edges computes Col-WSub$_{\mathcal{D}}(G)$ in $\tilde{O}(m^{3/2})$ time.*

Proof. Let e be the diagonal edge in $\mathcal{D}$, let u_1 and u_2 be the endpoints and u_3, u_4 the other two vertices in the diamond.

We can enumerate all the copies of the colored triangles $u_1 - u_2 - u_3$ and $u_1 - u_2 - u_4$ in G in time $O(m^{3/2})$. We create a hashmap d_{u_3} and for each triangle of the first type with vertices $v_1 - v_2 - v_3$ we add the product of $w(v_1, v_3)$ and $w(v_2, v_3)$ to d_{u_3}. Similarly we will fill a hashmap d_{u_4} with the triangles of the second type. Then for each edge (v_1, v_2) connecting vertices of the type u_1 and u_2 we will compute $w(v_1, v_2) \cdot d_{u_3}((v_1, v_2)) \cdot d_{u_4}((v_1, v_2))$ and aggregate all the results to obtain Col-WSub$_{\mathcal{D}}(G)$. The total runtime of the algorithm will be $\tilde{O}(m^{3/2})$. ◀

▶ **Lemma C.8.** *There is an algorithm that for any colored, weighted input graph G with m edges computes Col-WSub$_{\mathcal{H}_2}(G)$ in $\tilde{O}(m^{5/3})$ time.*

Proof. To compute Col-WSub$_{\mathcal{H}_2}(G)$ we can perform thresholding on the vertices of G, dividing them in two groups for every color depending on their degrees. Set $\Delta = m^{1/3}$, and let u_i be a vertex $\mathcal{H}_2$ we define $V_H^{(u_i)} = \{v \in V^{(u_i)(G)} : d(v) > \Delta\}$ and $V_L^{(u_i)} = \{v \in V^{(u_i)(G)} : d(v) \leq \Delta\}$. Every appearance of $\mathcal{H}_2$ in G must be as one of the 32 different combinations of vertices from the high and low degree sets. We show how to compute the counts for each combinations. Each of the combinations will correspond to at least one of the 4 following cases, which are shown in Fig. 13. We use u_c for the central vertex and u_1, u_2, u_3, u_4 for the exterior vertices:

1. The central vertex is in the low degree set: In this case we can list all pairs of hyperedges for each vertex in $V_L^{(u_c)}$, and then check if they induce a valid copy of $\mathcal{H}_2$ in constant time, in that case we just aggregate the product of weights of the hyperedges in the copy. There will at most $O(m\Delta) = O(m^{4/3})$ pairs of hyperedges to check.
2. All the vertices have high degree: We split the graph in two parts and list each of the parts separately. Each part will be formed by one hyperedge of arity 3 and one of the normal edges. Both parts intersect in three vertices forming a diagonal. Let u_c, u_2, u_4 be the vertices forming the intersection.
   We can list every copy of each of the halfs. For each half we list all possible hyperedges corresponding to the arity-3 hyperedge and vertices in the high set of the vertex that reach the normal edge. There will be at most $O(m\frac{m}{\Delta}) = O(m^{5/3})$ pairs for each of the halfs. We aggregate the products of weights of each copy into two hashmaps (one for each half) using the values of u_c, u_2, u_4 as the key.
   We then iterate over the hashmaps multiplying the values with similar entries and aggregating the counts. This will take total $\tilde{O}(m^{5/3})$ time.

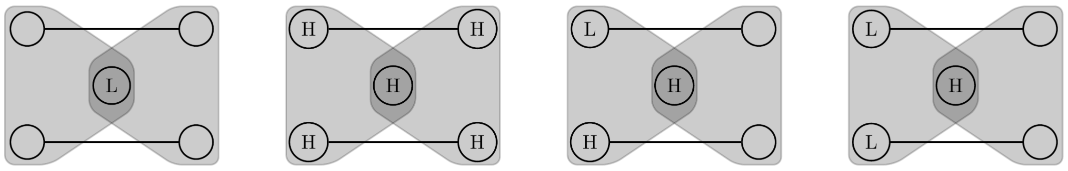

Figure 13 The possible cases for thresholding of hypergraph $\mathcal{H}_2$ that are analyzed in Lemma C.8.

3. A hyperedge contains 1 low degree vertex: Let e be the hyperedge with the low degree vertex and e' the other hyperedge. Let u_1 be the low degree vertex and u_4 the high degree. We split $\mathcal{H}_2$ in two halfs, one containing e and the edge (u_1, u_2) and the other containing e' and the edge (u_3, u_4), note that they intersect at the vertices u_c, u_2, u_4. We can list every copy of each of the halfs. For the half containing the low degree vertex we can just iterate over the vertices in $V_L^{(u_1)}$ and look at each pair of neighbors, there will be at most $O(m\Delta) = O(m^{4/3})$ possible pairs. For the other half we can list all possible hyperedges corresponding to e' and vertices in $V_H^{(u_4)}$. There will be at most $O(m\frac{m}{\Delta}) = O(m^{5/3})$ pairs. For each of the halfs we aggregate the products of weights of each copy into two hashmaps (one for each half) using the values of u_c, u_2, u_4 as the key. We then iterate over the hashmaps multiplying the values with similar entries and aggregating the counts. This will take total $\tilde{O}(m^{5/3})$ time.
4. A hyperedge contains 2 low degree vertices: Let e be the vertex of $\mathcal{H}_2$ with the two low degree vertices. We can iterate over all the hyperedges in G with the same colors containing two low degree vertices, and then iterate over each pair of edges coming from each of the low degree vertices. We can then check in constant time if the hyperedge together with the two edges induce a copy of $\mathcal{H}_2$ in G, in which case we add the product of the weight to the total count. There will be at most $O(m\Delta^2) = O(m^{5/3})$ subgraphs to verify.

◀

D Proof of Lemma 6.1

The proof will closely follow the structure of Sections 4 and 5 in [29], with the necessary modifications, as we are simply adapting the existing algorithms.

D.1 Combinatorial algorithm for cycles

Let P_r be the path with $r+1$ vertices and r edges with consecutive vertices having consecutive colors. For a colored graph G we use $V^{(l)}(G)$ to denote the set of vertices with color l or just $V^{(l)}$ if G is clear from the context. In this section indexes and colors are taken modulo k.

We first show the following auxiliary lemma, equivalent to Lemma 4.1 in [29]:

▶ **Lemma D.1.** *Let G be a weighted colored graph with m edges. There is an algorithm that for integers r, o and $\Delta \geq 1$ computes:*

- *For every pair $u, v \in V(G)$, the total weight $N_{r,o,\Delta}(u, v)$ of paths $P_r = \{p_0, ..., p_r\}$ in G such that $p_i \in V^{(i+o)}(G)$ for all i, $p_0 = u$, $p_r = v$ and $d(p_i) \leq \Delta$ for every $0 < i < r$.*
- *For every pair $u, v \in V(G)$ with $d(u) > \Delta$ and for every function $f : \{1, ..., r\} \to \{Low, High\}$ the total weight $M_{r,o,\Delta,f}$ of paths $P_r = \{p_0, ..., p_r\}$ in G such that $p_i \in V^{(i+o)}(G)$ for all i, $p_0 = u$, $p_r = v$ and for all $i > 0$, $d(p_i) \leq \Delta$ if $f(i) = Low$ and $d(p_i) > \Delta$ if $f(i) = High$.*

- *For every pair $u, v \in V(G)$ with $d(v) > \Delta$ and for every function $f : \{0, ..., r-1\} \to \{Low, High\}$ the total weight $M'_{r,o,\Delta,f}$ of paths $P_r = \{p_0, ..., p_r\}$ in G such that $p_i \in V^{(i+o)}(G)$ for all i, $p_0 = u$, $p_r = v$ and for all $i < r$, $d(p_i) \leq \Delta$ if $f(i) = Low$ and $d(p_i) > \Delta$ if $f(i) = High$.*

The first can be computed in time $\tilde{O}(m\Delta^{r-1})$ and the second and third in time $\tilde{O}(m^2/\Delta)$. There are at most $O(m\Delta^{r-1})$ pairs u, v for which $N_{r,\Delta}(u, v) > 0$.

Proof. For the first quantity, look at the all the edges between $V^{(o)}(G)$ and $V^{(1+o)}(G)$ and fix the position of the first two vertices p_0 and p_1, there will be at most $O(m)$ choices. We have $d(p_1) \leq \Delta$, hence we have at most Δ choices for p_2. Similarly, all vertices $p_2, ..., p_{r-1}$ will have low degree and hence given p_i we will have at most Δ choices for p_{i+1}, we need to do this $r-1$ times. Therefore, there will be at most $O(m\Delta^{r-1})$ possible paths. For each path we multiple the weight of the edges and aggregate the product into the hashmap $N_{r,o,\Delta}$ using the endpoints of the path as key. There will be at most $O(m\Delta^{r-1})$ possible non-zero entries.

For the second quantity, we do induction in r. For $r = 1$ we just need to looks at all the edges with one endpoint u in $V^{(o)}(G)$ with $d(u) > \Delta$ and another endpoint v in $V^{(1+o)}(G)$. We then store the weight of the edge (u, v) in the corresponding $M_{r,o,\Delta,f}(u, v)$ with f depending on the degree of v. There are at most $O(m/\Delta)$ vertices with high degree and they can have at most n neighbors, hence we can complete this step in $O(nm/\Delta) = \tilde{O}(m^2/\Delta)$ time.

For the inductive step, assume that we can compute all the counts for P_{r-1} and we compute P_r. Fix vertex $u \in V^{(o)}(G)$ with $d(u) > \Delta$ and edge (v', v) with $v' \in V^{(r-1+o)}(G)$ and $v \in V^{(r+o)}(G)$, we have at most $O(m/\Delta)$ choices for u and m choices for (v', v), hence at most $O(m^2/\Delta)$ total choices. Using the inductive assumption, for each pair u, v' and any function $g : \{1, ..., r-1\} \to \{\text{Low}, \text{High}\}$ we have the value of $M_{r-1,\Delta,g}(u, v')$. Every P_r in G starting at layer o must also contain a P_{r-1} starting a layer o. Hence, we can then set the function $f : \{1, ..., r\} \to \{\text{Low}, \text{High}\}$ that agrees with g in each value and has $f(r) = \text{Low}$ if $d(v) \leq \Delta$ and $f(r) = \text{High}$ otherwise, and add the product of $M_{r-1,o,\Delta,g}(u, v')$ and the weight of (u, v) to the entry (u, v) in the hashmap $M_{r,o,\Delta,f}$. We only need constant time per pair $u, (v', v)$ and hence this step will take $\tilde{O}(m^2/\Delta)$ total time.

The third quantity is equivalent to the second, just reversing the position of the high degree vertex, and it can be shown in a similar way selecting edges connecting colors o and $o + 1$ and vertices from V^{r+o}. ◀

We can now prove the combinatorial algorithm:

▶ **Lemma D.2.** *Let G be a colored weighted graph with m edges. For all $k > 3$, there is a combinatorial algorithm that computes Col-WSub$_{\mathcal{C}_k}(G)$ in time $\tilde{O}(m^{2-1/\lceil k/2 \rceil})$.*

Proof. Let $\Delta = m^{1/\lceil k/2 \rceil}$. For every function $g; \{1, ..., k\} \to \{\text{Low}, \text{High}\}$ we compute the total weight W_g of colorful k-cycles $\{v_1, .., v_k\}$ in G such that the degree of the $v_i \leq \Delta$ if $g(i) = \text{Low}$ and $v_i > \Delta$ otherwise. We have that $\text{Col-WSub}_{\mathcal{C}_k}(G) = \sum_g W_g$. We distinguish two cases:

- $g(i) = \text{Low}$ for all i. In this case we can split the k-cycle into two paths $P_{\lfloor k/2 \rfloor}$ from layer 0 to layer $\lfloor k/2 \rfloor$ and a $P_{\lceil k/2 \rceil}$ from layer $\lfloor k/2 \rfloor$ to layer 0. We can then express W_g as follows:

$$W_g = \sum_{\substack{u \in V^{(0)} : d(u) \leq \Delta \\ v \in V^{(\lfloor k/2 \rfloor)} : d(v) \leq \Delta}} N_{\lfloor k/2 \rfloor, 0, \Delta}(u, v) \cdot N_{\lceil k/2 \rceil, \lfloor k/2 \rfloor, \Delta}(v, u)$$

Using Lemma D.1 we can compute $N_{\lfloor k/2 \rfloor,0,\Delta}(u,v)$ and $N_{\lceil k/2 \rceil,\lfloor k/2 \rfloor,\Delta}(v,u)$ for every pair u, v in total time $\tilde{O}(m\Delta^{\lceil k/2 \rceil - 1})$, and we will have at most $O(m\Delta^{\lceil k/2 \rceil -1})$ non-zero entries, thus the sum can be computed in time $\tilde{O}(m\Delta^{\lceil k/2 \rceil -1}) = \tilde{O}(m^{2-1/\lceil k/2 \rceil})$.

- There is at least an entry i for which $g(i) = \text{High}$. In this case we split the k-cycle into two paths $P_{\lfloor k/2 \rfloor}$ from layer i to layer $i + \lfloor k/2 \rfloor$ and a $P_{\lceil k/2 \rceil}$ from layer $i + \lfloor k/2 \rfloor$ to layer i. We set f as the restriction of g to the values $\{i+1, ..., i+\lfloor k/2 \rfloor\}$ and f' as the restriction of g to the values $\{i + \lfloor k/2 \rfloor ..., i-1\}$ (modulo k). We can express W_g as follows:

$$W_g = \sum_{\substack{u \in V^{(i)}: d(u) > \Delta \\ v \in V^{(\lfloor k/2 \rfloor + i)}}} M_{\lfloor k/2 \rfloor, i, \Delta, f}(u,v) \cdot M'_{\lceil k/2 \rceil, \lfloor k/2 \rfloor + i, \Delta, f'}(v,u)$$

Using Lemma D.1 we can compute $M_{\lfloor k/2 \rfloor,i,\Delta,f}(u,v)$ and $M'_{\lceil k/2 \rceil, \lfloor k/2 \rfloor + i,\Delta,f'}(v,u)$ for all pairs u, v in total time $\tilde{O}(m^2/\Delta)$. The number of pairs is bounded by $O(m^2/\Delta)$ as there are at most $O(m/\Delta)$ choices for u. Hence we can compute W_g in time $\tilde{O}(m^2/\Delta) = \tilde{O}(m^{2-1/\lceil k/2 \rceil})$.

◀

D.2 Matrix multiplication algorithm for cycles

We prove now the matrix multiplication algorithm. The following lemma together with Lemma D.2 completes the proof of Lemma 6.1. The structure of our proof follows closely Section 5 in [29]. Note that the indexes over $\{0, ..., k-1\}$ will be taken as modulo k.

▶ **Lemma D.3.** *Let G be a colored weighted graph with m edges. For all $k > 3$, there is an algorithm that computes Col-WSub$_{C_k}(G)$ in time $\tilde{O}(m^{c_k})$.*

Proof. Let the colors of G go from 0 to $k-1$. For each color $i \in [0, k-1]$, let $V^{(i)}$ be the set of vertices of G with color i. We partition each of the sets into $\log m$ degree classes. For all $1 \le j < \log n$ we define the set $W_j^{(i)}$ as follows:

$$W_j^{(i)} = \{v \in V^{(i)} | 2^j \le d(v) \le 2^{j+1}\}$$

Every such set will have at most $O(m/2^j)$ vertices. We classify the colorful k-cycles in G into $O(\log^k n)$ distinct classes, according to the degree of the vertices in the cycles.

We fix a tuple of degree classes $f = (f_0, ..., f_{k-1})$ and show how to compute the weighted sum of cycles $\{u_0, u_1, \ldots, u_{k-1}\}$ satisfying $u_i \in W_{f_i}^{(i)}$ for all i. Repeating this process for all classes will only add a poly-logarithmic term.

Let A_i^f be the $|W_{f_i}^{(i)}| \times |W_{f_{i+1}}^{(i+1)}|$ weighted adjacency matrix of G restricted to the edges from vertices in $W_{f_i}^{(i)}$ to vertices in $W_{f_{i+1}}^{(i+1)}$. A_i^f can only have $O(m)$ non-zero entries, and hence we can construct a sparse representation of it in $O(m)$ time. For every $p, q \in \{0, \ldots, k-1\}$ we set:

$$B_{p,q}^f = A_p^f \times A_{p+1}^f \times \cdots \times A_{q-1}^f$$

Note that the trace of the matrix $B_{0,k}^f$ will be equal to the sum of the product of weights of the k-cycles in G satisfying the degree classes f. We can compute trace$(B_{0,k}^f)$ by selecting two points $p, q \in \{0, \ldots, k-1\}$ and computing instead trace$(B_{p,q}^f B_{q,p}^f) = $ trace$(B_{0,k}^f)$.

For all $i \in \{0, k-1\}$, we set $d_i = f_i / \log m$. Hence we can represent the tuple of array classes as $d = \{d_0, ..., d_{k-1}\}$. The value of d_i ranges from 0 to 1 and the sets $W_{f_i}^{(i)}$ will have at most $O(m^{1-d_i})$ vertices. For every $B_{p,q}^f$, let $P_{p,q}^f$ be the minimum such that we can compute

$B^f_{p,q}$ in $O(m^{P^f_{p,q}})$ time (and hence $B^f_{p,q}$ will have at most $O(m^{P^f_{p,q}})$ non-zero entries). We can compute $B^f_{p,q}$ in three different ways:

1. First we compute a sparse representation of $B^f_{p,q-1}$. For every entry $(u,v) \in W^{(p)}_{f_p} \times W^{(q-1)}_{f_{q-1}}$ of $B^f_{p,q-1}$, we look at the neighbors of v in $W^{(q)}_{f_q}$, for every such neighbor v' we update $B^f_{p,q}(u,v') {+}{=} B^f_{p,q-1}(u,v) \cdot w((v,v'))$. Computing $B^f_{p,q-1}$ will take $O(m^{P^f_{p,q-1}})$ time and it will have as much non-zero entries. Every vertex v will have at most $O(m^{d_{q-1}})$ neighbors. Hence, $B^f_{p,q}$ can be computed in time $O(m^{P^f_{p,q-1}+d_{q-1}})$.
2. Similar to the previous method but reversed: We compute $B^f_{p+1,q}$ and then for every entry (u,v) of $B^f_{p+1,q}$ we look at the neighbors u' of u in $W^{(p)}_{f_p}$ and update $B^f_{p,q}(u',v) {+}{=} B^f_{p+1,q}(u,v) \cdot w((u',u))$. We can then compute $B^f_{p,q}$ in time $O(m^{P^f_{p+1,q}+d_{p+1}})$.
3. For some $p < r < q$, we compute $B^f_{p,r}$ and $B^f_{r,q}$ and then compute their product to obtain $B^f_{p,q}$. We need $O(m^{P^f_{p,r}} + m^{1-d_p} + m^{1-d_r})$ time to compute the matrix representation of $B^f_{p,r}$, similarly we need $O(m^{P^f_{r,q}} + m^{1-d_r} + m^{1-d_q})$ to obtain the matrix representation of $B^f_{r,q}$. We need additional $O(m^{M(1-d_p,1-d_r,1-d_q)})$ time to compute the product, where $M(a,b,c)$ is the minimum value of g such that we can multiply matrices with dimensions $m^a \times m^b$ and $m^b \times m^c$ in time $O(m^g)$.

We can now bound the value of $P^f_{p,q}$:

- From the first method we have $P^f_{p,q} \le P^f_{p,q-1} + d_{q-1}$.
- From the second method we have $P^f_{p,q} \le P^f_{p+1,q} + d_{p+1}$.
- From the third method we have:

$$P^f_{p,q} \le \min_{p<r<q} \max\{P^f_{p,r}, P^f_{r,q}, M(1-d_p, 1-d_r, 1-d_q)\}$$

Any $B^f_{p,p+1}$ can be constructed in time $O(m)$, hence $P^f_{p,p+1} = 1$. Therefore, we can define $P^f_{p,q}$ as follows:

$$\begin{aligned} P^f_{p,p+1} &= 1 \\ P^f_{p,q} &= \min\left\{P^f_{p,q-1} + d_{q-1}, P^f_{p+1,q} + d_{p+1}, \min_{p<r<q} \max\{P^f_{p,r}, P^f_{r,q}, M(1-d_p, 1-d_r, 1-d_q)\}\right\} \end{aligned} \tag{10}$$

Given the sparse representations of $B^f_{p,q}$ and $B^f_{q,p}$ we can compute trace$(B^f_{p,q} \times B^f_{q,p})$ in time equal to the number of non-zero entries in each of the matrices $O(m^{P^f_{p,q}} + m^{P^f_{q,p}})$. We define $C_k(f)$ as the minimum g such that we can compute trace$(B^f_{0,k})$ in time $O(m^g)$, we will have:

$$C_k(f) = \min_{0 \le p < q \le k-1} \max P^f_{p,q}, P^f_{q,p} \tag{11}$$

Equations (10) and (11) are identical to the equations in [29] and [21]. Hence we can define c_k as in [29], giving:

$$c_k = \max_f C_k(f) \tag{12}$$

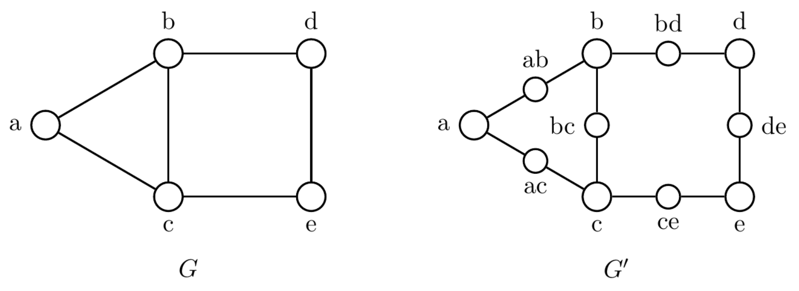

Figure 14 An example of the construction of G' from a graph G. The original graph had one $\mathcal{C}_3$, one $\mathcal{C}_4$ and one $\mathcal{C}_5$. Those cycle become $\mathcal{C}_6, \mathcal{C}_8$ and $\mathcal{C}_{10}$ respectively.

Which means that $O(m^{c_k})$ is an upper bound for computing trace$(B^f_{0,k-1})$, for all f. We can then iterate over all tuples of degree classes f and compute Col-WSub$_{\mathcal{C}_k}(G)$ in time $\tilde{O}(m^{c_k})$. ◀

E Proof of Lemma 1.6

E.1 Even cycles

We first prove Lemma 1.6 for even cycles.

▶ **Definition E.1** (Expanded Graph). *Given a graph G with n vertices and m edges we define G' as the graph resulting of subdividing every edge in G by a 2-edge path. The resultant graph has $m + n$ vertices and $2m$ edges.*

Fig. 14 shows an example of the construction. We can show that G' has constant degeneracy.

▷ Claim E.2. The degeneracy of G' is 2.

Proof. Every connected subgraph with more than 1 vertex must include one of the new vertices, the degree of these vertices is 2 so the degeneracy can not be larger than 2. ◀

We also show that there is a direct relation between the cycles in G and in G'.

▶ **Lemma E.3.** *For any $k \geq 3$, the number of $\mathcal{C}_k$ in G is equal to the number of $\mathcal{C}_{2k}$ in G'.*

Proof. Clearly every k-cycle in G will become a $2k$-cycle in G' as every edge is now a path of two edges. Now consider a cycle of length $2k$ in G', we can can show that it corresponds to exactly one k-cycle in G. Let $v_1, ..., v_{2k}$ be the vertices of the cycles, note that G' is a bipartite graph with all the original vertices of G in one side and the new intermediary vertices in the other, hence every alternate vertex will also be a vertex in G. Let $v_1, v_3, ..., v_{2k-1}$ be those vertices, we can see that there must be an edge connecting v_i with v_{i+2} in G for odd i as there is a 2-path connecting them in G', hence the alternate vertices induce a k-cycle in G. ◀

We can now prove the following lemma.

▶ **Lemma E.4.** *Let $c \geq 1$, if there is an $f(\kappa)O(n^c)$ algorithm for counting $2k$-cycles, then there is a $O(m^c)$ algorithm for counting k-cycles.*

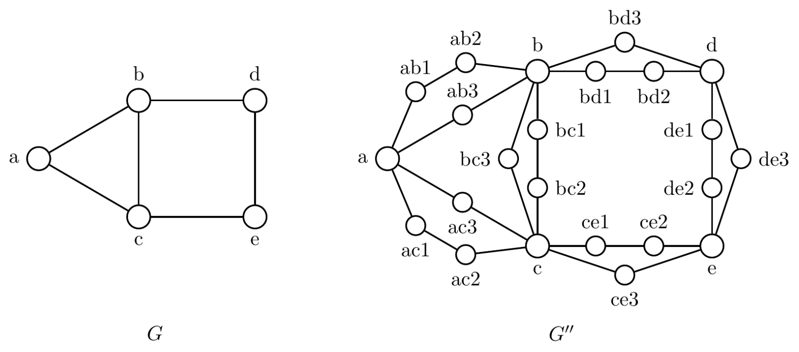

Figure 15 An example of the construction of G'' from a graph G. The original graph had one $\mathcal{C}_3$ and one $\mathcal{C}_4$. The expanded graph has 3 $\mathcal{C}_7$ and 5 $\mathcal{C}_9$.

Proof. Let G be any graph with n vertices, m edges and degeneracy κ, and assume we have a $f(\kappa)O(n^c)$ algorithm for the $Sub_{\mathcal{C}_{2k}}$ problem, we show that we can compute $\mathrm{Sub}_{\mathcal{C}_k}(G)$.

First, compute the expanded graph G' from G. This can be done in $O(m+n)$ time. Use the algorithm for counting the number of $2k$-cycles in G', by Lemma E.3 we have that this amount will be equal to $\mathrm{Sub}_{\mathcal{C}_k}(G)$. The number of vertices in G' is $O(m)$ and the degeneracy is 2, hence the runtime will be $f(2)O(m^c) = O(m^c)$. ◀

The previous lemma, together with Lemma 4.7 gives Corollary 1.7.

▶ **Corollary 1.7.** *There is an algorithm that, for any graph G, computes $\mathrm{Sub}_{\mathcal{C}_5}(G)$ in time $O\left(m^{d_5}\right) \approx O\left(m^{1.63}\right)$.*

E.2 Odd cycles

We now prove the lower bound for 7 and 9-cycles. We need to slightly modify our construction.

▶ **Definition E.5** (Odd Expanded Graph)**.** *Given a graph G with n vertices and m edges we define G'' as the graph resulting of replacing every edge by a 2-path and a 3-path. The resultant graph has $3m+n$ vertices and $5m$ edges.*

Fig. 15 shows an example of the construction. Again, this graph will have a degeneracy of 2. We can show the following:

▶ **Lemma E.6.** *For any odd k:*

- $\mathrm{Sub}_{\mathcal{C}_7}(G'') = 3 \cdot \mathrm{Sub}_{\mathcal{C}_3}(G)$
- $\mathrm{Sub}_{\mathcal{C}_9}(G'') = \mathrm{Sub}_{\mathcal{C}_3}(G) + 4 \cdot \mathrm{Sub}_{\mathcal{C}_4}(G)$

Proof. First consider any $\mathcal{C}_3$ in G given by vertices v_1, v_2, v_3, we can see that they will be part of exactly one $\mathcal{C}_9$ in G'': take the 3-path connecting every pair of vertices from v_1, v_2, v_3, the result is a $\mathcal{C}_9$. The same cycle will also corresponded to three different $\mathcal{C}_7$ in G'': For two of the pairs from v_1, v_2, v_3 take the 2-path and take the 3-path from the remaining pair, there are three possible pairs, giving three cycles.

Now consider any $\mathcal{C}_4$ in G, we can see that it will correspond with 4 different $\mathcal{C}_9$ in G'': let the vertices v_1, v_2, v_3, v_4 be the four vertices of the cycle, for three of the pairs $(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)$ take the 2-path, and take the 3-path for the remaining pair. The result is 4 different $\mathcal{C}_9$.

Now inversely, consider a $\mathcal{C}_7$ in G'', note that it can have at most 3 vertices from the original graph, and hence it can only correspond with $\mathcal{C}_3$, giving the equality of the lemma. Similarly a $\mathcal{C}_9$ in G'' can have either 3 or 4 vertices from the original graph G, and hence it will correspond to either a $\mathcal{C}_3$ or $\mathcal{C}_4$. ◀

We can now show the lower bounds:

▶ **Lemma E.7.**
- *If an $f(\kappa)o(n^{d_3})$ algorithm for* $\mathrm{Sub}_{\mathcal{C}_7}(G)$ *exists then there exist a $o(m^{d_3})$ algorithm for* $\mathrm{Sub}_{\mathcal{C}_3}(G)$*.*
- *If an $f(\kappa)o(n^{d_4})$ algorithm for* $\mathrm{Sub}_{\mathcal{C}_9}(G)$ *exists then there exist a $o(m^{d_4})$ algorithm for* $\mathrm{Sub}_{\mathcal{C}_4}(G)$*.*

Proof. Assume an $f(\kappa)o(n^{d_3})$ algorithm exists for $\mathrm{Sub}_{\mathcal{C}_7}(G)$. For any graph G we can construct G'' in $O(n+m)$ time. Then we can use the $f(\kappa)o(n^{d_3})$ algorithm to compute the number of $\mathcal{C}_7$ in G'' from which we can get the number of $\mathcal{C}_3$ in G using Lemma E.6. G'' has degeneracy of 2 and $O(m)$ vertices, hence the algorithm will run in $o(m^{d_3})$.

Similarly for $\mathcal{C}_9$, assume a $f(\kappa)o(n^{d_4})$ algorithm exists for $\mathrm{Sub}_{\mathcal{C}_9}(G)$. For any graph G we can construct G'' in $O(n+m)$ time, we can also use the existing $O(m^{d_3}) = o(m^{d_4})$ algorithm for computing the number of $\mathcal{C}_3$ in G. We then use the assumed algorithm for $\mathcal{C}_9$ in G''. We can use Lemma E.6 to compute the number $\mathcal{C}_4$ in G from the number of $\mathcal{C}_3$ in G and the number of $\mathcal{C}_9$ in G''. This will take $o(m^{d_4})$. ◀

F Proof of Lemma 1.9

Consider the 10-vertex graph $H_\triangle$ shown in Fig. 16, it has one acyclic orientation $\vec{H}_\triangle$ that is $\mathcal{H}_\triangle$-reducible, that can be seen by replacing every source in $\vec{H}_\triangle$ with a hyperedge.

Let G be any input graph with bounded degeneracy, we can show that a subquadratic algorithm for computing $\mathrm{Sub}_{H_\triangle}(G)$ implies a subquadratic algorithm for computing $\mathrm{Sub}_{\vec{H}_\triangle}(\vec{G})$.

▶ **Lemma F.1.** *If there is a $f(\kappa)o(n^2)$ algorithm for computing* $\mathrm{Sub}_{H_\triangle}(G)$*, then there is a $f(\kappa)o(n^2)$ algorithm for computing* $\mathrm{Sub}_{\vec{H}_\triangle}(\vec{G})$*.*

Proof.

$$\mathrm{Sub}_{H_\triangle}(G) = \sum_{\vec{H}\in\Sigma(H_\triangle)} \mathrm{Sub}_{\vec{H}}(\vec{G}) = \sum_{\vec{H}\in\Sigma(H_\triangle)} \sum_{\vec{H}'\in Spasm(\vec{H})} f(\vec{H},\vec{H}')\mathrm{Hom}_{\vec{H}'}(\vec{G}) \tag{13}$$

The values of $f(\vec{H},\vec{H}')$ are non-zero for each pair $\vec{H},\vec{H}'$. We can hence rewrite $\mathrm{Sub}_{H_\triangle}(G)$ as a linear combination of homomorphism counts. Consider the direct pattern $\vec{H}_\triangle$, note that because it has size 10 it can not be in the spams of any other acyclic orientation of $H_\triangle$ besides its own. Hence it will have a non-zero coefficient of $f(\vec{H}_\triangle,\vec{H}_\triangle)$ in the linear combination.

We can now use Lemma 3.1 in [29] to transform our $f(\kappa)o(n^2)$ algorithm for $\mathrm{Sub}_{H_\triangle}(G)$ into a $f(\kappa)o(n^2)$ algorithm for counting any of the homomorphisms with non-zero coefficient in the linear combination, hence we will have a $f(\kappa)o(n^2)$ algorithm for $\mathrm{Hom}_{\vec{H}_\triangle}(\vec{G})$.

Note that the rest of the patterns in $Spasm(\vec{H}_\triangle)$ might not have non-zero coefficient in the linear combination, however, they will have at most 9 vertices, and hence we can compute the number of homomorphisms for each of them in subquadratic time using Theorem 1.1. We can then use the homomorphism counts of all the graphs in $Spasm(\vec{H}_\triangle)$ to compute $\mathrm{Sub}_{\vec{H}_\triangle}(\vec{G})$ in total time $f(\kappa)o(n^2)$. ◀

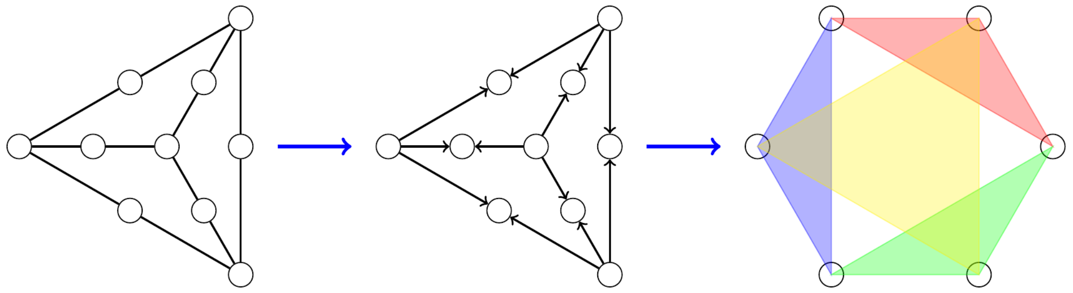

Figure 16 The graph $H_\triangle$, its acyclic orientation $\vec{H}_\triangle$ that is $\mathcal{H}_\triangle$-reducible and the hypergraph $\mathcal{H}_\triangle$.

Now we can prove Lemma 1.9.

▶ **Lemma 1.9.** *If Conjecture 1.8 holds, there is no algorithm that computes* $\mathrm{Sub}_H(G)$ *in time* $f(\kappa)o(n^2)$ *for all patterns* H *with* 10 *vertices.*

Proof. Assume there is a an algorithm that computes $\mathrm{Sub}_H(G)$ in $f(\kappa)o(n^2)$ time for all patterns with 10 vertices. This implies that $\mathrm{Sub}_{H_\triangle}(G)$ can be computed in $f(\kappa)o(n^2)$ time. Using Lemma F.1 we have that there will also be an $f(\kappa)o(n^2)$ algorithm for computing $\mathrm{Sub}_{\vec{H}_\triangle}(\vec{G})$. We show how to compute $\mathrm{Sub}_{\mathcal{H}_\triangle}(G)$ using this algorithm.

Let G be a hypergraph with all hyperedges of arity 3. Construct the graph $\vec{G}'$ by replacing every hyperedge $e = \{u_1, u_2, u_3\}$ by a new vertex v_e and the directed edges (v_e, u_1),(v_e, u_2),(v_e, u_3). This graph will have $O(m)$ edges and a degeneracy of 3.

Note that every copy of $\mathcal{H}_\triangle$ in G will now have become a copy of $\vec{H}_\triangle$ in $\vec{G}'$. Similarly every copy of $\vec{H}_\triangle$ in $\vec{G}'$ will correspond to a copy of $\mathcal{H}_\triangle$ in G. Hence $\mathrm{Sub}_{\mathcal{H}_\triangle}(G) = \mathrm{Sub}_{\vec{H}_\triangle}(\vec{G}')$. We can then use the subquadratic algorithm that we assumed exists to compute $\mathrm{Sub}_{\mathcal{H}_\triangle}(G)$ in time $f(3)o(m^2) = o(m^2)$. But this contradicts Conjecture 1.8. ◀