Subgraph Counting in Subquadratic Time for Bounded Degeneracy Graphs

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

C. Seshadhri @ ORCID
University of California, Santa Cruz, CA, 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.

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.66

Related Version *Full Version*: `https://arxiv.org/abs/2410.08376`

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 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 non-trivial 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)

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 current fastest algorithm for k-cycle detection in general graphs (see (3) and [29]). 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.** *Let κ denote the degeneracy of the input graph G.*

- *There is an algorithm that computes $\mathrm{Sub}_{\mathcal{C}_6}(G)$ and $\mathrm{Sub}_{\mathcal{C}_7}(G)$ in time $f(\kappa)\tilde{O}(n^{d_3})$, where $d_3 < 1.41$. Moreover, unless there is an $O(m^{d_3-\varepsilon'})$-time algorithm for counting triangles for some $\varepsilon' > 0$, there is no $f(\kappa)O(n^{d_3-\varepsilon})$-time algorithm for any $\varepsilon > 0$.*
- *There is an algorithm that computes $\mathrm{Sub}_{\mathcal{C}_8}(G)$ and $\mathrm{Sub}_{\mathcal{C}_9}(G)$ in time $f(\kappa)\tilde{O}(n^{d_4})$, where $d_4 < 1.48$. Moreover, unless there is an $O(m^{d_4-\varepsilon'})$-time algorithm for counting 4-cycles for some $\varepsilon' > 0$, there is no $f(\kappa)O(n^{d_4-\varepsilon})$-time algorithm for any $\varepsilon > 0$.*
- *There is an algorithm that computes $\mathrm{Sub}_{\mathcal{C}_{10}}(G)$ in time $f(\kappa)\tilde{O}(n^{d_5})$, where $d_5 < 1.63$. Moreover, unless there is an $O(m^{d_5-\varepsilon'})$-time algorithm for counting 5-cycles for some $\varepsilon' > 0$, there is no $f(\kappa)O(n^{d_5-\varepsilon})$-time algorithm for any $\varepsilon > 0$.*

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

[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.

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 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 $\text{Sub}_{\vec{H}}(\vec{G})$.

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

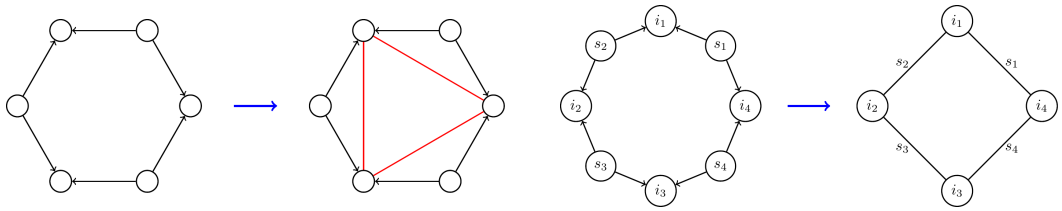

Figure 1 (a) The 6-cycle obstruction, this orientation has three sources intersecting with each other, this oriented pattern can not be counted in linear time in bounded degeneracy graphs. Adding an edge connecting the end-points of every out-out wedge gives a triangle. (b) An example of how the oriented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

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).

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. Let P be a hypergraph, 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 from 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 from 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 second 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.

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

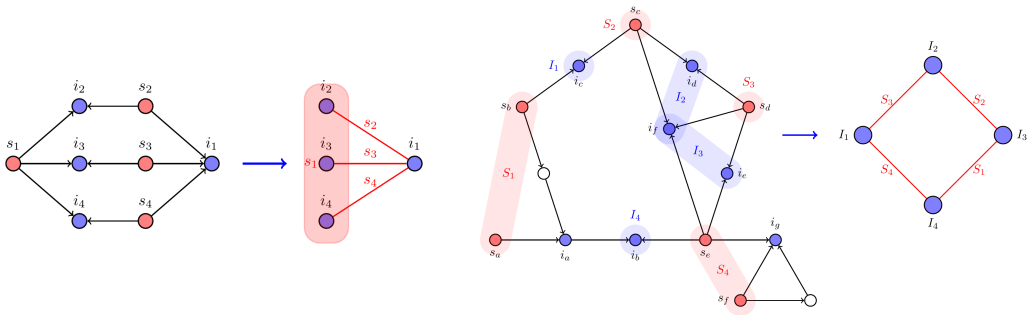

Figure 2 Two more complex examples of P-reducibility.

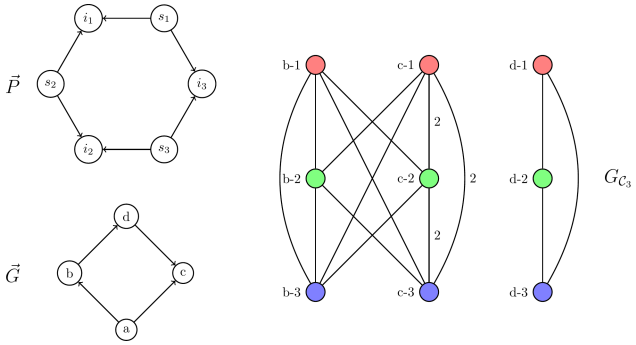

Figure 3 An example of the construction of $G_{\mathcal{C}_3}$, for pattern $\vec{H}$ and input graph $\vec{G}$. The red vertices correspond with i_1 in $\vec{H}$, the green ones with i_2 and the blue ones with i_3. The weight of the edges is 1 except when indicated. For example there are two homomorphisms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

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 $(\phi(I_j)\text{-}j)$ 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 $(u\text{-}j)$ and $(v\text{-}j')$ 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 from 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

[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.

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 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 $\tilde{O}(m^c)$-time 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. Let $\mathcal{P}$ be a set of 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 on the complexity of $\text{Hom}_H(G)$.

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 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 $6 \leq k \leq 10$. For any $\varepsilon > 0$, the existence of an $f(\kappa)O(n^{d_{\lfloor k/2 \rfloor}-\varepsilon})$-time algorithm for counting k-cycles implies an $O(m^{d_{\lfloor k/2 \rfloor}-\varepsilon'})$-time algorithm for counting $\lfloor k/2 \rfloor$-cycles in general graphs, for some $\varepsilon' > 0$.*

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 $O\left(m^{d_5}\right) < O\left(m^{1.63}\right)$.*

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.** *For any $\varepsilon > 0$, there is no $O(m^{2-\varepsilon})$-time 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, then for any $\varepsilon > 0$ there is no algorithm that computes $\mathrm{Sub}_H(G)$ in time $f(\kappa)O(n^{2-\varepsilon})$ 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.

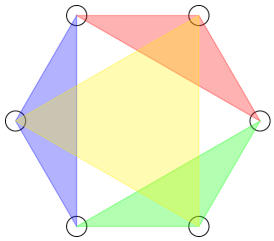

Figure 4 The hypergraph $\mathcal{H}_{\triangle}$.

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 $f(\kappa, k)m^{o(\tau(H)/\ln \tau(H))}$-time 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 Full Version, 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 $H' \in Spasm(H)$, allows to compute $\mathrm{Sub}_H(G)$, as given by the following equation: $\mathrm{Sub}_H(G) = \sum_{H' \in Spasm(H)} f(H')\mathrm{Hom}_{H'}(G)$.

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 $Spasm(H)$.

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 nonnegative value κ 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 $Reach_{\vec{H}}(s)$ to denote the set of vertices reachable from the source s. Abusing notation, for a set $S' \subseteq S(\vec{H})$ we use $Reach_{\vec{H}}(S')$ (or $Reach(S')$ if $\vec{H}$ is clear from the context) to denote the set of vertices reachable from at least one vertex in S'. We use $\vec{H}(S)$ and $\vec{H}(s)$ for the subgraphs of $\vec{H}$ induced by $Reach_{\vec{H}}(S)$ and $Reach_{\vec{H}}(s)$.

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 from 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 near-linear time for graphs of constant degeneracy. Bera et al. showed an exact characterization of which patterns have $\tau(H) = 1$, which depends only on the length of the largest induced cycle of the pattern ($LICL(H)$).

▶ **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 mapped by ϕ 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 generalization of a graph where 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, that is, the hypergraph on $k+1$ vertices with all possible hyperedges of arity k.

3 The reduction procedure

In this section, we explain the reduction procedure underlying the notion of P-reducibility. The idea is to organize the structure of the directed pattern $\vec{H}$ based on the hypergraph P.

First, we partition the set of sources of the directed pattern $\vec{H}$ into $|E(P)|$ disjoint subsets S_e, one for each hyperedge e in $E(P)$. Every source belongs to exactly one subset S_e. This partition allows us to decompose $\vec{H}$ into smaller subgraphs, each corresponding to a hyperedge, where the subgraphs can be efficiently counted using Bressan's algorithm.

Second, we associate to each vertex $v \in V(P)$ a subset of intersection vertices $I_v \subseteq I(\vec{H})$. Not all the intersection vertices need to belong to one of the subsets; we let I^* denote the set of intersection vertices that appear in at least one I_v. A given intersection vertex may belong to multiple subsets, but we require that the vertices of P corresponding to all subsets containing a common intersection vertex induce a connected subgraph.

To connect the sources to the intersection vertices, we define, for each hyperedge e, the set $I(e)$ as the subset of I^* reachable from the sources in S_e. We require that the sources in S_e reach all the intersection vertices assigned to the vertices v incident to e. Moreover, for each vertex $v \in V(P)$, the sources associated with any hyperedge containing v must reach all the vertices in I_v.

These conditions ensure that we can correctly combine the homomorphism counts for each subgraph $\vec{H}(S_e)$ to compute the homomorphism counts of $\vec{H}$. We now present the full formal definition.

▶ **Definition 3.1** (P-reducible). *A connected DAG $\vec{H}$ is P-reducible if there exist subsets $I_v \subseteq I(\vec{H})$ for every vertex $v \in V(P)$, and subsets $S_e \subseteq S(\vec{H})$ for every hyperedge $e \in E(P)$, satisfying the following conditions:*

- *Let $I^* = \bigcup_v I_v$. For every intersection vertex $i \in I^*$, the set of vertices $\{v : i \in I_v\} \subseteq V(P)$ induces a connected hypergraph in P.*
- *The subsets S_e are disjoint, i.e., $S_e \cap S_{e'} = \emptyset$ for all $e \neq e'$, and they cover all sources, i.e., $\bigcup_{e \in E(P)} S_e = S(\vec{H})$.*
- *For every hyperedge $e \in E(P)$, the subset S_e contains a source s_e such that $\vec{H}(s_e) \cap I^* = \vec{H}(S_e) \cap I^* = I(e)$, and the subgraph $\vec{H}(S_e)$ admits a $\tau = 1$ DAG-tree decomposition rooted at s_e.*
- *For every vertex $v \in V(P)$ and every hyperedge e containing v, we have $I_v \subseteq I(e)$.*
- *For every hyperedge $e \in E(P)$, we have $\bigcup_{v \in e} I_v = I(e)$.*

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 $(\phi(I_v)\text{-}v)$ 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.** $\text{Hom}_{\vec{H}}(\vec{G}) = \textit{Col-WSub}_P(G_P)$

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 $\tilde{O}(m^c)$-time algorithm that for any graph G' computes $\textit{Col-WSub}_P(G')$, 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 $\vec{G}$.*

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 $\text{Col-WSub}_P(G_P)$ in time $f(d)\tilde{O}(n^c)$. From Lemma 3.4 we have that $\text{Col-WSub}_P(G_P)$ will be equal to $\text{Hom}_{\vec{H}}(\vec{G})$. ◀

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 Col-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 $\textit{Col-WSub}_P$ 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 $\text{Hom}_H(G)$ 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* $f(\kappa)\tilde{O}(n^{d_3}) < f(\kappa)O(n^{1.41})$.

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

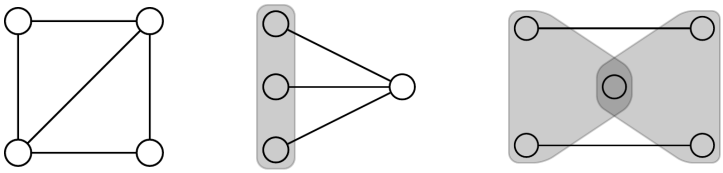

Figure 5 The diamond graph $\mathcal{D}$, the hypergraph $\mathcal{H}_1$ and the hypergraph $\mathcal{H}_2$.

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* 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 Col-WSub$_P(G)$ in $\tilde{O}(m^{5/3})$ time. This together with Lemma 3.6 gives that we compute 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 Hom$_{H'}(G)$ for them and use inclusion-exclusion to obtain the value of 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 Col-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 best known upper bound $\omega \leq 2.371339$ on the matrix multiplication exponent [3], we have that the matrix multiplication algorithm is faster and we get the following bounds on d_k: $d_3 < 1.41$, $d_4 < 1.48$ and $d_5 < 1.63$.

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`.