Near-Linear Time Homomorphism Counting in Bounded Degeneracy Graphs: The Barrier of Long Induced Cycles

Suman K. Bera*, Noujan Pashanasangi*, and C. Seshadhri*

University of California, Santa Cruz

{sbera,npashana,sesh}@ucsc.edu

Abstract

Counting homomorphisms of a constant sized pattern graph H in an input graph G is a fundamental computational problem. There is a rich history of studying the complexity of this problem, under various constraints on the input G and the pattern H. Given the significance of this problem and the large sizes of modern inputs, we investigate when near-linear time algorithms are possible. We focus on the case when the input graph has bounded degeneracy, a commonly studied and practically relevant class for homomorphism counting. It is known from previous work that for certain classes of H, H-homomorphisms can be counted exactly in near-linear time in bounded degeneracy graphs. Can we precisely characterize the patterns H for which near-linear time algorithms are possible?

We completely resolve this problem, discovering a clean dichotomy using fine-grained complexity. Let m denote the number of edges in G. We prove the following: if the largest induced cycle in H has length at most 5, then there is an $O(m \log m)$ algorithm for counting H-homomorphisms in bounded degeneracy graphs. If the largest induced cycle in H has length at least 6, then (assuming standard fine-grained complexity conjectures) there is a constant $\gamma > 0$, such that there is no $o(m^{1+\gamma})$ time algorithm for counting H-homomorphisms.

1 Introduction

Analyzing occurrences of small pattern graphs in a large input graph is a fundamental algorithmic problem with a rich line of work in both theory and practice [Lov67, CN85, FG04, DJ04, Lov12, ANRD15, CDM17, PSV17, RW20]. A common version of this problem is *homomorphism counting*, which has numerous applications in logic, properties of graph products, partition functions in statistical physics, database theory, and network science [CM77, BW99, DG00, BCL+06, PSV17, DRW19, PS20]. Denote the pattern simple graph as $H = (V(H), E(H))$, which is thought of as fixed (or constant-sized). The input simple graph is denoted by $G = (V(G), E(G))$. An H-homomorphism is an edge-preserving map $f : V(H) \to V(G)$. Formally, $\forall (u, v) \in E(H)$, $(f(u), f(v)) \in E(G)$. When f is an injection, this map corresponds to the common notion of subgraphs. We use $\mathrm{Hom}_H(G)$ to denote the count of the distinct H-homomorphisms.

The problem of computing $\mathrm{Hom}_H(G)$ for various choices of H is a deep subfield of study in graph algorithms [IR78, AYZ97, BW99, DG00, DST02, DJ04, BCL+06, CDM17, Bre19, RW20].

*All the authors are supported by NSF TRIPODS grant CCF-1740850, NSF CCF-1813165, CCF-1909790, and ARO Award W911NF1910294.

Indeed, even the case of H being a triangle, clique, or cycle have led to long lines of results. Many practical and theoretical algorithms for subgraph counting are based on homomorphism counting, and it is known that subgraph counts can be expressed as a linear combination of homomorphism counts [CDM17]. Let $n = |V(G)|$ and $k = |V(H)|$. It is known that computing $\mathrm{Hom}_H(G)$ is $\#W[1]$-hard when parameterized by k (even when H is a k-clique), so we do not expect $n^{o(k)}$ algorithms for general H [DJ04]. Yet the n^k barrier can be beaten when H has structure. Notably, Curticapean-Dell-Marx proved that if H has treewidth at most 2, then $\mathrm{Hom}_H(G)$ can be computed in $\mathrm{poly}(k) \cdot n^\omega$, where ω is the matrix multiplication constant [CDM17].

For many modern applications in network science, since n is very large, one desires linear-time algorithms. Indeed, one might argue that the "right" notion of computational efficiency in these settings is (near) linear time. Motivated by these concerns, we investigate the barriers for achieving linear time algorithms to count $\mathrm{Hom}_H(G)$, especially when G is a "real-world" graph.

We focus on the class of *bounded degeneracy graphs*. This is the class of graphs where all subgraphs have a constant average degree. A seminal result of Chiba-Nishizeki proves that clique counting can be done in linear time for such graphs [CN85]. Importantly, this paper introduced the technique of graph orientations for subgraph counting, that has been at the center of many state-of-the-art practical algorithms [ANRD15, JSP15, PSV17, OB17, JS17, PS20]. The degeneracy has a special significance in the analysis of real-world graphs, since it is intimately tied to the technique of "core decompositions".

Many algorithmic ideas for homomorphism or subgraph counting on bounded degeneracy graphs have been quite successful in practice, and form the basis of state-of-the-art algorithms. Can we understand the limits of these methods? Theoretically, when is homomorphism counting possible in near-linear time on bounded degeneracy graphs? Chiba-Nishizeki [CN85] proved that clique and 4-cycle counting can be done in linear time. A result of the authors shows that near-linear time is possible when $|V(H)| = k \leqslant 5$ [BPS20]. In a significant generalization, Bressan [Bre19] defines an intricate notion of *DAG treewidth*, and shows (among other things) that a near-linear time algorithm exists when the DAG treewidth of H is one. These results lead us to the main question addressed by our work.

Can we characterize the pattern graphs H for which $\mathrm{Hom}_H(G)$ is computable in near-linear time (when G has bounded degeneracy)?

Our main result is a surprisingly clean resolution of this problem, assuming fine-grained complexity results. There is a rich history of *complexity dichotomies* for homomorphism detection and counting problems [HN90, DG00, DJ04, Gro07, RW20]. In this work, we discover such a dichotomy for near-linear time algorithms.

Let $\mathrm{LICL}(H)$ be the length of the largest induced cycle in H.

Theorem 1.1. *Let G be an input graph with n vertices, m edges, and degeneracy κ. Let $f : \mathbb{N} \to \mathbb{N}$ denote some explicit function. Let $\gamma > 0$ denote the constant from the* Triangle Detection Conjecture.

If $\mathrm{LICL}(H) \leqslant 5$: there exists an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa) \cdot m \log n$.

If $\mathrm{LICL}(H) \geqslant 6$: assume the Triangle Detection Conjecture. *For any function $g : \mathbb{N} \to \mathbb{N}$, there is no algorithm with (expected) running time $g(\kappa)o(m^{1+\gamma})$ that computes $\mathrm{Hom}_H(G)$.*

(We note that the condition on H involves induced cycles, but we are interested in counting non-induced homomorphisms.)

Abboud and Williams introduced the Triangle Detection Conjecture on the complexity of determining whether a graph has a triangle [AW14]. Assuming that there is no $O(m^{1+\gamma})$ time

triangle detection algorithm, they proved lower bounds for many classic graph algorithm problems. It is believed that the constant γ could be arbitrarily close to 1/3 [AW14].

Conjecture 1.2 (Triangle Detection Conjecture [AW14]). *There exists a constant $\gamma > 0$ such that in the word RAM model of $O(\log n)$ bits, any algorithm to detect whether an input graph on m edges has a triangle requires $\Omega(m^{1+\gamma})$ time in expectation.*

1.1 Main Ideas

Background for the Upper Bound. We begin with some context on the main algorithmic ideas used for homomorphism/subgraph counting in bounded degeneracy graphs. Any graph G of bounded degeneracy has an acyclic orientation $G^{\rightarrow}$, where all outdegrees are bounded. Moreover, $G^{\rightarrow}$ can be found in linear time [MB83]. For any pattern graph H, we consider all possible acyclic orientations. For each such orientation $H^{\rightarrow}$, we compute the number of $H^{\rightarrow}$-homomorphisms (in $G^{\rightarrow}$). (Directed homomorphisms are maps that preserve the direction of edges.) Finally, we sum these counts over all acyclic orientations $H^{\rightarrow}$. This core idea was embedded in the seminal paper of Chiba-Nishizeki, and has been presented in such terms in many recent works [PSV17, OB17, Bre19, BPS20].

Since $G^{\rightarrow}$ has bounded outdegrees, for any bounded, rooted tree $T^{\rightarrow}$ (edges pointing towards leaves), *all* $T^{\rightarrow}$-homomorphisms can be explicitly enumerated in linear time. To construct a homomorphism of $H^{\rightarrow}$, consider the rooted trees of a DFS forest $T_1^{\rightarrow}, T_2^{\rightarrow}, \ldots$ generated by processing the sources first. We first enumerate all homomorphisms of $T_1^{\rightarrow}, T_2^{\rightarrow}, \ldots$ in linear time. We need to count how many tuples of these homomorphisms can be "assembled" into $H^{\rightarrow}$-homomorphisms. (We note that the number of $H^{\rightarrow}$-homomorphisms can be significantly super-linear.) The main idea is to index the rooted tree homomorphisms appropriately, so that $H^{\rightarrow}$-homomorphisms can be counted in linear time. This requires a careful understanding of the shared vertices among the rooted DFS forest $T_1^{\rightarrow}, T_2^{\rightarrow}, \ldots$.

The previous work of the authors showed how this efficient counting can be done when $|V(H)| \leqslant 5$, though the proof was ad hoc [BPS20]. It did a somewhat tedious case analysis for various H, exploiting specific structure in the various small pattern graphs. Bressan gave a remarkably principled approach, introducing the notion of the *DAG treewidth* [Bre19]. We will take some liberties with the original definition, for the sake of exposition. Bressan defined the DAG treewidth of $H^{\rightarrow}$, and showed that when this quantity is 1, $\mathrm{Hom}_H^{\rightarrow}(G^{\rightarrow})$ can be computed in near-linear time. The DAG treewidth is 1 when the following construct exists. For any source s of $H^{\rightarrow}$, let $R(s)$ be the set of vertices in $H^{\rightarrow}$ reachable from s. The sources of $H^{\rightarrow}$ need to be be arranged in a tree $\mathcal{T}$ such that the following holds. If s lies on the (unique) path between s_1 and s_2 (in $\mathcal{T}$), then $R(s_1) \cap R(s_2) \subseteq R(s)$. In some sense, this gives a divide-and-conquer framework to construct (and count) $H^{\rightarrow}$-homomorphisms. Any $H^{\rightarrow}$-homomorphism can be broken into "independent pieces" that are only connected by the restriction of the homomorphism to $R(s)$. By indexing all the tree homomorphisms appropriately, the total count of $H^{\rightarrow}$-homomorphisms can be determined in near-linear time by dynamic programming. Note that we need the DAG treewidth of *all* acyclic orientations of H to be 1, which is a challenging notion to describe succinctly.

From Induced Cycles to DAG tree decompositions. We observe an interesting contrast between the previous work of the authors and Bressan's work. The former provides a simple family of H for which $\mathrm{Hom}_H(G)$ can be computed in near-linear time in bounded degeneracy graphs, yet the proofs were ad hoc. The latter gave a principled algorithmic approach, but it does not succinctly describe what kinds of H allow for such near-linear algorithms. Can we get the best of both worlds?

Indeed, that is what we achieve. By a deeper understanding of *why* $|V(H)| \leqslant 5$ was critical in [BPS20] and generalizing it through the language of DAG tree decompositions, we can prove:

the DAG treewidth of H is one iff $\text{LICL}(H) \leqslant 5$.

When $\text{LICL}(H) \leqslant 5$, for any acyclic orientation $H^{\rightarrow}$, we provide a (rather complex) iterative procedure to construct the desired DAG tree decomposition $\mathcal{T}$. The proof is intricate and involves many moving parts. The connection between induced cycles and DAG tree decompositions is provided by a construct called the *unique reachability graph*. For any set S of sources in $H^{\rightarrow}$, construct the following simple, undirected graph $UR(S)$. Add edge (s, s') if there exists a vertex that is in $R(s) \cap R(s')$, but not contained in any $R(s'')$, for $s'' \in S \setminus \{s, s'\}$. A key lemma states that if $UR(S)$ contains a cycle (for any subset S of sources), then H contains an induced cycle of at least twice the length. Any cycle in a simple graph has length at least 3. So if $UR(S)$ has a cycle, then H has an induced cycle of length at least 6. Thus, if $\text{LICL}(H) \leqslant 5$, for all S, the simple graph $UR(S)$ is a forest.

For any set S of sources, we will (inductively) construct a *partial* DAG tree decomposition that only involves S. Let us try to identify a "convenient" vertex $x \in S$ with the following property. We inductively take the partial DAG tree decomposition $\mathcal{T}'$ of $S \setminus \{x\}$, and try to attach x as a leaf in $\mathcal{T}'$ preserving the DAG tree decomposition conditions (that involve reachability). By carefully working out the definitions, we identify a specific intersection property of $R(x)$ with the reachable sets of the other sources in $S \setminus \{x\}$. When this property holds, we can attach x and extend the partial DAG tree decomposition, as described above. When the property fails, we prove that the degree of x in $UR(S)$ is at least 2. But $UR(S)$ is a forest, and thus contains a vertex of degree 1. Hence, we can always identify a convenient vertex x, and can iteratively build the entire DAG tree decomposition.

We also prove the converse. If $\text{LICL}(H) \geqslant 6$, then the DAG treewidth (of some orientation) is at least two. This proof is significantly less complex, but crucially uses the unique reachability graph.

The Lower Bound: Triangles Become Long Induced Cycles. We start with the simple construction of [BPS20] that reduces triangle counting in arbitrary graphs to 6-cycle counting in bounded degeneracy graphs. Given a graph G where we wish to count triangles, we consider the graph G' where each edge of G is subdivided into a path of length 2. Clearly, triangles in G have a 1-1 correspondence with 6-cycles in G'. It is easy to verify that G' has bounded degeneracy.

Our main idea is to generalize this idea for any H where $\text{LICL}(H) = 6$. The overall aim is to construct a graph G' where each H-homomorphism corresponds to a distinct induced 6-cycle in G', which comes from a triangle in G. We will actually fail to achieve this aim, but get "close enough" to prove the lower bound.

Let $\overline{H}$ denote the pattern obtained after removing the induced 6-cycle from H. Let us outline the construction of G'. We first take three copies of the vertices of G. For every edge (u, v) of G, connect copies of u and v that lie in *different* copies by a path of length two. Note that each triangle of G has been converted into six 6-cycles. We then add a single copy of $\overline{H}$, and connect $\overline{H}$ to the remaining vertices (these connections depend on the edges of H). This completes the description of G'. Exploiting the relation of degeneracy to vertex removal orderings, we can prove that G' has bounded degeneracy.

It is easy to see that every triangle in G leads to a distinct H-homomorphism. Yet the converse is potentially false. We may have "spurious" H-homomorphisms that do not involve the induced 6-cycles that came from triangles in G. By a careful analysis of G', we can show the following. Every spurious H-homomorphism *avoids* some vertex in the copy of $\overline{H}$ (in G').

These observations motivate the problem of *partitioned*-homomorphisms. Let $\mathbf{P}$ be a partition of the vertices of G' into k sets. A partitioned-homomorphism is an H-homomorphism where each vertex is mapped to a different set of the partition. We can choose $\mathbf{P}$ appropriately, so that the

triangle count of G is the number of partitioned-homomorphisms scaled by a constant (that only depends on the automorphism group of H). Thus, we reduce triangle counting in arbitrary graphs to counting partitioned-homomorphisms in bounded degeneracy graphs.

Our next insight is to give up the hope of showing a many-one linear-time reduction from triangle counting to H-homomorphisms, and instead settle for a Turing reduction. This suffices for the lower bound of Theorem 1.1. Using inclusion-exclusion, we can reduce a single instance of partitioned-homomorphism counting to 2^k instances of vanilla H-homomorphism counting. The details are somewhat complex, but this description covers the basic ideas.

When $\mathrm{LICL}(H) > 6$, we replace edges in G by longer paths, to give longer induced cycles. The partitions become more involved, but the essence of the proof remains the same.

2 Related Work

Counting homomorphisms has a rich history in the field of parameterized complexity theory. Díaz et al. [DST02] designed a dynamic programming based algorithm for the $\mathrm{Hom}_H(G)$ problem with runtime $O(2^k n^{\mathrm{tw}(H)+1})$ where $\mathrm{tw}(H)$ is the treewidth of the target graph H. Dalmau and Jonsson [DJ04] proved that $\mathrm{Hom}_H(G)$ is polynomial time solvable if and only if H has bounded treewidth, otherwise it is $\#W[1]$-complete. More recently, Roth and Wellnitz [RW20] consider a *doubly restricted* version of $\mathrm{Hom}_H(G)$, where both H and G are from *restricted* graph classes. They primarily focus on the parameterized dichotomy between poly-time solvable instances and $\#W[1]$-completeness.

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 [RS83, RS84, RS86]; although it has been discovered before under different names [BB73, Hal76]. Over the years, tree decompositions have been used extensively to design fast divide-and-conquer algorithms for combinatorial problems. Degeneracy is a nuanced measure of sparsity and has been known since the early work of Szekeres-Wilf [SW68]. The family of bounded degeneracy graphs is quite rich: it involves all minor-closed families, bounded expansion families, and preferential attachment graphs. Most real-world graphs tend to have small degeneracy ([GG06, JS17, SERF18, BCG20, BS20], also Table 2 in [BCG20]), underscoring the practical importance of this class. The degeneracy has been exploited for subgraph counting problems in many algorithmic results [CN85, Epp94, ANRD15, JSP15, PSV17, OB17, JS17, PS20].

Bressan [Bre19] introduced the concept of DAG treewidth to design faster algorithms for homomorphism and subgraph counting problems in bounded degeneracy graphs. They prove the following dichotomy for the subgraph counting problem. 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 (Theorem 3.4). On the other hand, assuming the exponential time hypothesis [IPZ98], the subgraph counting problem does not admit any $f(\kappa, k)m^{o(\tau(H)/\ln \tau(H))}$ algorithm, for any positive function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Previous work of the authors shows that for every $k \geqslant 6$, there exists some pattern H with k vertices, such that $\mathrm{Hom}_H(G)$ cannot be counted in linear time, assuming fine-grained complexity conjectures [BPS20]. We note that these results do not give a complete characterization like Theorem 1.1. They define classes of H that admit near-linear or specific polynomial time algorithms, and show that *some* H (but not all) outside this class does not have such efficient algorithms.

We remark here that in an independent and parallel work, Gishboliner, Levanzov, and Shapira [GLS20] effectively prove the same characterization for linear time homomorphism counting.

The problem of approximately counting homomorphism and subgraphs have been studied ex-

tensively in various Big Data models such as the property testing model [ELRS17, ERS18, AKK18, ERS20], the streaming model [BKS02, MMPS11, KMSS12, AGM12, JSP13, PTTW13, MVV16, BC17, BS20], and the map reduce model [Coh09, SV11, KPP+14]. These works often employ clever sampling based techniques and forego exact algorithms.

Almost half a century ago, Itai and Rodeh [IR78] gave the first non-trivial algorithm for the triangle detection and finding problem with $O(m^{3/2})$ runtime. Currently, the best known algorithm for the triangle detection problem uses fast matrix multiplication and runs in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$ [AYZ97]. Improving on the exponent is a major open problem, and it is widely believed that $m^{4/3}$ (corresponding to $\omega = 2$) is a lower bound for the problem. Thus, disproving the Triangle Detection Conjecture would require a significant breakthrough. See [AW14] for a detailed list of other classic graph problems whose hardness is derived using Triangle Detection Conjecture.

3 Preliminaries

We use G to denote the input graph and H to denote the target pattern graph. We consider both $G = (V(G), E(G))$ and $H = (V(H), E(H))$ to be simple, undirected, and connected graphs. Throughout the paper, we use m and n to denote $|V(G)|$ and $|E(G)|$ respectively, for the input graph G. We denote $|V(H)|$ by k.

We use $\#\mathrm{Hom}_H$ to denote the problem of counting homomorphisms for a fixed pattern graph H. We use $\#\mathrm{Sub}_H$ for the analogous subgraph counting problem.

If a subset of vertices $V' \subseteq V(G)$ is deleted from G, we denote the remaining subgraph by $G - V'$. We use $G[V']$ to denote the subgraph of G induced by V'. The length of the largest induced cycle in H is denoted by $\mathrm{LICL}(H)$.

We say that a graph G is *k-degenerate* if each non-empty subgraph of G has minimum vertex degree of at most k. The smallest integer k such that G is k-degenerate is the *degeneracy* of graph G. To denote the degeneracy of a graph G, we use $\kappa(G)$ or simply κ if G is clear from the context. Observe that by definition, if a graph is k-degenerate, all its subgraphs are also k-degenerate, so the degeneracy of each subgraph of G is at most $\kappa(G)$. We note that the *arboricity* is a closely related notion, and is related by a constant factor to the degeneracy.

We will heavily use acyclic orientations of graphs. For the graph H (say), we use $H^{\rightarrow}$ to denote an acyclic orientation of the simple graph H. Vertex orderings and DAGs are closely related to degeneracy. Given a vertex ordering $\prec$ of a graph G, we can obtain a DAG $G^{\rightarrow}_{\prec}$ by orienting each edge $\{u, v\} \in E(G)$ from u to v if $u \prec v$.

Now, we define a homomorphism from H to G. We denote the number of homomorphism from H to G by $\mathrm{Hom}_H(G)$.

Definition 3.1. A homomorphism from H to G is a mapping $\pi : V(H) \to V(G)$ such that, $\{\pi(u), \pi(v)\} \in E(G)$ for all $\{u, v\} \in E(H)$. If H and G are both directed, then π should preserve the directions of the edges. If π is injective, then it is called an embedding of H in G.

Next, we define a match (also called copy) of H in G.

Definition 3.2. A match of H in G is a subgraph of G that is isomorphic to H. If a match of H is an induced subgraph of G, then it is an induced match of H in G.

Observe that each embedding of H in G corresponds to a match of G.

DAG tree decompositions. Bressan [Bre19] defined the notion of *DAG tree decompositions* for DAGs, analogous to the widely popular tree decompositions for undirected graphs. The crucial

difference in this definition is that only the set of source vertices in the DAG are considered for creating the nodes in the tree. Let D be a DAG and $S \subseteq V$ be the set of source vertices in D. For a source vertex $s \in S$, let $\textsc{reachable}_D(s)$ denote the set of vertices in D that are reachable from s. For a subset of the sources $B \subseteq S$, let $\textsc{reachable}_D(B) = \bigcup_{s \in B} \textsc{reachable}_D(s)$. When the underlying DAG is clear from the context, we drop the subscript D.

Definition 3.3 (DAG tree decomposition [Bre19])**.** Let D be a DAG with source vertices S. A DAG tree decomposition of D is a tree $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ with the following three properties.

1. Each node $B \in \mathcal{B}$ (referred to as a "bag" of sources) is a subset of the source vertices S: $B \subseteq S$.
2. The union of the nodes in $\mathcal{T}$ is the entire set S: $\bigcup_{B \in \mathcal{B}} B = S$.
3. For all $B,\ B_1,\ B_2 \in \mathcal{B}$, if B lies on the unique path between the nodes B_1 and B_2 in $\mathcal{T}$, then $\textsc{reachable}(B_1) \cap \textsc{reachable}(B_2) \subseteq \textsc{reachable}(B)$.

The *DAG treewidth* of a DAG D is then defined as the minimum over all possible DAG tree decompositions of D, the size of the maximum *bag*. For a simple undirected graph H, the DAG treewidth is the maximum DAG treewidth over all possible acyclic orientations of H. We denote the DAG treewidth of D and H by $\tau(D)$ and $\tau(H)$, respectively.

Bressan [Bre19] gave an algorithm for solving the $\#\mathrm{Hom}_H$ problem in bounded degeneracy graphs.

Theorem 3.4 (Theorem 16 in [Bre19])**.** *Given an input graph G on m edges with degeneracy κ and a pattern graph H on k vertices, there is an $O(\kappa^k m^{\tau(H)} \log n)$ time algorithm for solving the $\#\mathrm{Hom}_H$ problem.*

4 LICL and Homomorphism Counting in Linear Time

We prove that the class of graphs with LICL $\leqslant 5$ is equivalent to the class of graphs with $\tau = 1$.

Theorem 4.1. *For a simple graph H, $\mathrm{LICL}(H) \leqslant 5$ if and only if $\tau(H) = 1$.*

By Theorem 3.4, this implies that $\mathrm{Hom}_H(G)$ can be determined in near-linear time when $\mathrm{LICL}(H) \leqslant 5$ and G has bounded degeneracy.

We first prove that, for a simple graph H, if $\mathrm{LICL}(H)$ is at most five, then $\tau(H) = 1$. This is discussed in Section 4.2. Then we prove the converse: if $\mathrm{LICL}(H)$ is at least six, then $\tau(H) \geqslant 2$. We take this up in Section 4.3.

Outline of the Proof Techniques. Before discussing the proofs in detail, we provide a high level description of the proof techniques.

Fix an arbitrary acyclic orientation $H^{\rightarrow}$ of H. We use S to denote the set of source vertices. We describe a recursive procedure to build a DAG tree decomposition of width one, starting from a single source in S.

Note that property (2) in the definition of DAG tree decomposition (Definition 3.3) requires the union of nodes in the tree to cover all the source vertices in S. So, we need to be careful, if we wish to use induction to construct the final DAG tree decomposition. To this end, we relax the property (1) and (2) of DAG tree decomposition and define a notion of *partial DAG tree decomposition*. In a partial DAG tree decomposition with respect to a subset $S_p \subseteq S$, the nodes in the tree are subsets of S_p and the union of the nodes cover the set S_p. The requirement of property (3) remains the

same. The width of the tree is defined same as before. We formalize this in Definition 4.5. Now, consider a subset $S_{r+1} \subseteq S$ of size $r+1$. We show how to build a partial DAG tree decomposition of width one for S_{r+1}, assuming there exists a partial DAG tree decomposition of width one for any subset of S of size r.

Let $x \in S_{r+1}$ and S_{-x} denote the set after removing the element x: $S_{-x} = S_{r+1} \setminus \{x\}$. Let $\mathcal{T}_{-x}$ be a partial DAG tree decomposition of width one for the set S_{-x} (such a tree exists by assumption). We identify a "good property" of the tree $\mathcal{T}_{-x}$ that enables construction of a width one partial DAG tree decomposition for the entire set S_{r+1}. The property is the following: there exists a leaf node ℓ in $\mathcal{T}_{-x}$ connected to the node $d \in \mathcal{T}_{-x}$ such that $\textsc{reachable}(x) \cap \textsc{reachable}(\ell) \subseteq \textsc{reachable}(d)$. We make this precise in Definition 4.7. Assume $\mathcal{T}_{-x}$ has this good property, and $\ell \in \mathcal{T}_{-x}$ be the leaf that enables $\mathcal{T}_{-x}$ to posses the good property. Then we first construct a width one partial DAG tree decomposition $\mathcal{T}_{-\ell}$ for the set $S_{-\ell} = S_{r+1} \setminus \{\ell\}$ and after that add ℓ as a leaf node to the node d in $\mathcal{T}_{-\ell}$. We prove that the resulting tree is indeed a valid width one partial DAG tree decomposition for S_{r+1}(we prove this in Claim 4.9). To complete the proof, it is now sufficient to show the existence of an element $x \in S_{r+1}$ such that a partial DAG tree decomposition for $\mathcal{T}_{-x}$ has the good property. This is the key technical element that distinguishes graphs with LICL at most 5 from those with LICL at least six.

We make a digression and discuss this key technical element further. We consider a graph that captures certain reachability aspects of the source vertices in $H^{\rightarrow}$. We define this as the unique rechability graph, UR_{S_p}, for a subset of the source vertices $S_p \subseteq S$. The vertex set of UR_{S_p} is simply the set S_p. Two vertices s_1 and s_2 in UR_{S_p} are joined by an edge if and only if there exists a vertex $v \in V(H^{\rightarrow})$ such that only s_1 and s_2 among the vertices in S_p, can reach v in $H^{\rightarrow}$. We prove that, if the underlying undirected graph H has $\text{LICL}(H) \leqslant 5$, then the graph UR_{S_p}, for any subset $S_p \subseteq S$, is acyclic. This is given in Lemma 4.3 in Section 4.1.

Now, coming back to the proof of Lemma 4.4, we show that there must exist an element $x \in S_{r+1}$ such that a partial DAG tree decomposition for $\mathcal{T}_{-x}$ has the "good property", as otherwise the unique reachability graph $\text{UR}_{S_{r+1}}$ over the set S_{r+1} has a cycle. However, this contradicts $\text{LICL}(H) \leqslant 5$ (Lemma 4.3). This is established in Claim 4.10. This completes the proof of Lemma 4.4.

Now consider Lemma 4.11. Observe that, to prove this lemma, it is sufficient to exhibit a DAG $H^{\rightarrow}$ of H with $\tau(H^{\rightarrow}) \geqslant 2$. We first prove in Lemma 4.12 that if the the unique reachability graph $\text{UR}_{S(H^{\rightarrow})}$ has a triangle, then $\tau(H^{\rightarrow}) \geqslant 2$. Then, for any graph H with $\text{LICL}(H) \geqslant 6$, we construct a DAG $H^{\rightarrow}$ such that the unique reachability graph $\text{UR}_{S(H^{\rightarrow})}$ has a triangle. It follows that $\tau(H) \geqslant 2$.

4.1 Main Technical Lemma

In this section, we describe our main technical lemma. We define a unique reachability graph for a DAG $H^{\rightarrow}$ over a subset of source vertices $S_p \subseteq S(H^{\rightarrow})$. This graph captures a certain reachability aspect of the source vertices in S_p in the graph $H^{\rightarrow}$. More specifically, two vertices s_1 and s_2 are joined by and edge in UR_{S_p} if and only if there exists a vertex v in $V(H^{\rightarrow})$ such that only s_1 and s_2 among the vertices in S_p, can reach v in $H^{\rightarrow}$.

Definition 4.2 (Unique reachability graph)**.** Let $H^{\rightarrow}$ be a DAG of H with source vertices S and $S_p \subseteq S$ be a subset of S. We define a unique reachability graph $\text{UR}_{S_p}(S_p, E_{S_p})$ on the vertex set S_p, and the edge set E_{S_p} such that there exists an edge $e = \{s_1, s_2\} \in E_{S_p}$, for $s_1, s_2 \in S_p$, if and only if the set $(\textsc{reachable}_{H^{\rightarrow}}(s_1) \cap \textsc{reachable}_{H^{\rightarrow}}(s_2)) \setminus \textsc{reachable}_{H^{\rightarrow}}(S_p \setminus \{s_1, s_2\})$ is non-empty.

We are interested in the existence of a cycle in UR_{S_p}. We show that a cycle in UR_{S_p} is closely

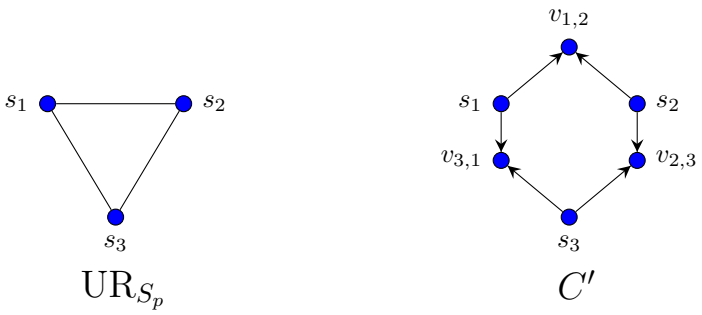


Figure 1: Let $S_p = \{s_1, s_2, s_3\}$. On the left, we give an example of a UR_{S_p} graph with a triangle. On the right, we give a possible example of the vertices $v_{1,2}$, $v_{2,3}$, and $v_{3,1}$ ($v_{i,j}$ is as defined in the proof of Lemma 4.3). C' forms an induced cycle of length six in H.

related to an induced cycle in H, the underlying undirected graph of $H^{\rightarrow}$. More specifically, we prove that if LICL of H is at most five, then UR_{S_p} must be acyclic for each subset $S_p \subseteq S$.

Lemma 4.3. *Let $H^{\rightarrow}$ be a DAG of H with source vertices S and $S_p \subseteq S$ be an arbitrary subset of S. Let $\mathrm{UR}_{S_p}(S_p, E_{S_p})$ be the unique reachability graph for the subset S_p. If $\mathrm{LICL}(H) \leqslant 5$, then UR_{S_p} is acyclic.*

Proof. We in fact prove a stronger claim. We show that if UR_{S_p} has an ℓ-cycle, then $\mathrm{LICL}(H) \geqslant 2\ell$. Consider an edge $\{s_i, s_j\}$ in E_{S_p}. Let UNIQUE-REACHABLE(s_i, s_j) denote the set of vertices in $H^{\rightarrow}$ reachable from s_i and s_j both, but non-reachable from any other vertices in S_p. Let $\mathrm{dist}(s, t)$ denote the length of the shortest directed path from s to t in $H^{\rightarrow}$. We set $\mathrm{dist}(s, t) = \infty$, if t is not reachable from s. Now, let $v_{i,j}$ be the vertex in the UNIQUE-REACHABLE(s_i, s_j) set with the smallest total distance (directed) from s_i and s_j (breaking ties arbitrarily). Formally, $v_{i,j} = \arg\min_v \mathrm{dist}(s_i, v) + \mathrm{dist}(s_j, v)$, where $v \in$ UNIQUE-REACHABLE(s_i, s_j).

Let $C = s_1, s_2, \ldots, s_\ell, s_1 (= s_{\ell+1})$ be an ℓ-cycle in UR_{S_p}. Then using C and the vertices $v_{i,i+1}$, for $i \in [\ell]$ (abusing notation, we take $v_{\ell,\ell+1} = v_{\ell,1}$), we construct a cycle of length at least 2ℓ in H. Denote by $p_{s \to v}$ the directed path from a source vertex $s \in S_p$ to a vertex $v \in H^{\rightarrow}$. Intuitively, inserting the paths $p_{s_i \to v_{i,i+1}}$ and $p_{s_{i+1} \to v_{i,i+1}}$ between the source s_i and s_{i+1}, for each $i \in [\ell]$ (again, taking $s_{\ell+1} = s_1$), induces a cycle of length at least 2ℓ in H. See Figure 1 for a simple demonstration of this. We make this formal below.

Let $V(p)$ denote the set of vertices of a path p. Any edge between vertices in $V(p_{s_i \to v_{i,i+1}})$ other than the edges of $p_{s_i \to v_{i,i+1}}$, results in either a path between s_i and $v_{i,i+1}$ shorter than $\mathrm{dist}(s_i, v_{i,i+1})$ or a directed cycle in $H^{\rightarrow}$. Thus, the edges of $p_{s_i \to v_{i,i+1}}$ are the only edges between vertices in $V(p_{s_i \to v_{i,i+1}})$. It is easy to see that any edge between $V(p_{s_i \to v_{i,i+1}}) \setminus \{v_{i,i+1}\}$ and $V(p_{s_{i+1} \to v_{i,i+1}}) \setminus \{v_{i,i+1}\}$ result in a vertex $v'_{i,i+1}$ where $\mathrm{dist}(s_i, v'_{i,i+1}) + \mathrm{dist}(s_{i+1}, v'_{i,i+1}) < \mathrm{dist}(s_i, v_{i,i+1}) + \mathrm{dist}(s_{i+1}, v_{i,i+1})$, therefore no such edges exist. Also, any edge from a vertex in REACHABLE$_{H^{\rightarrow}}(S_p \setminus \{s_i, s_{i+1}\})$ to a vertex in $V(p_{s_i \to v_{i,i+1}})$ or $V(p_{s_{i+1} \to v_{i,i+1}})$ implies that $v_{i,i+1} \in$ REACHABLE$_{H^{\rightarrow}}(S_p \setminus \{s_i, s_{i+1}\})$, which is not true by definition. Hence, there are no such edges either.

We use $E(p)$ to denote the set of edges of a path p. For convenience, we assume $\ell + 1$ in the

index is to be treated as 1 instead. Let

$$V_{C'} = \bigcup_{p \in P} V(p), \qquad E_{C'} = \bigcup_{p \in P} E(p),$$
$$\text{where } P = \bigcup_{i \in [\ell]} \{p_{s_i \to v_{i,i+1}}, p_{s_{i+1} \to v_{i,i+1}}\}.$$

Now, it is easy to see that the graph induced by the set $V_{C'}$ is indeed an induced cycle C'. There are 2ℓ paths in P, and each path $p \in P$ has at least two vertices. As we showed above, these paths do not share edges. Thus the length of C' is at least 2ℓ, and $\mathrm{LICL}(H) \geqslant 2\ell$. Considering the contrapositive, we deduce that if $\mathrm{LICL}(H) \leqslant 5$, then UR_{S_p} is acyclic. □

4.2 DAG Treewidth for Graphs with LICL at most Five

In this section, we prove the following lemma.

Lemma 4.4. *For every simple graph H, if $\mathrm{LICL}(H) \leqslant 5$ then $\tau(H) = 1$.*

We introduce some notation. We start with defining the notion of partial DAG tree decomposition. In this definition, we consider a tree decomposition with respect to a subset of the source vertices of the original DAG.

Definition 4.5 (partial DAG tree decomposition)**.** Let $H^{\to}$ be a DAG with source vertices S. For a subset $S_p \subseteq S$, a partial DAG tree decomposition of $H^{\to}$ with respect to S_p is a tree $\mathcal{T} = (\mathcal{B}, \mathcal{E})$ with the following three properties.

1. Each node $B \in \mathcal{B}$ (referred to as a "bag") is a subset of the sources in S_p: $B \subseteq S_p$.
2. The union of the nodes in $\mathcal{T}$ is the entire set S_p: $\bigcup_{B \in \mathcal{B}} B = S_p$.
3. For all $B, B_1, B_2 \in \mathcal{B}$, if B is on the unique path between B_1 and B_2 in $\mathcal{T}$, then we have $\textsc{reachable}(B_1) \cap \textsc{reachable}(B_2) \subseteq \textsc{reachable}(B)$.

The partial DAG treewidth of $\mathcal{T}$ is $\max_{B \in \mathcal{B}} |B|$. Abusing notation, we use $\tau(\mathcal{T})$ to denote the partial DAG treewidth of $\mathcal{T}$.

Observe that, when $S_p = S$, we recover the original definition of DAG tree decomposition. Our proof strategy is to show by induction on the size of the subset S_p that there exists a partial DAG tree decomposition of width one for each $S_p \subseteq S$. In particular, when $S_p = S$, it follows that there exists a DAG tree decomposition for $H^{\to}$ of width one.

We next define INTERSECTION-COVER for a pair of vertices, based on the third property of the DAG tree decomposition. We generalize this notion to a subset of source vertices $S_p \subseteq S$ and define a notion of S_p-COVER. These notions will be useful in identifying a suitable source vertex in an existing partial DAG tree decomposition to attach a new node to it.

Definition 4.6 (INTERSECTION-COVER and S_p-COVER)**.** Let $H^{\to}$ be a DAG with sources S. Let s_1 and s_2 be a pair of sources in S. We call a source $s \in S$ an INTERSECTION-COVER of s_1 and s_2 if $\textsc{reachable}(s_1) \cap \textsc{reachable}(s_2) \subseteq \textsc{reachable}(s)$. Furthermore, assume $S_p \subseteq S$ is a subset of the sources S. We call a source $s \in S$, a S_p-COVER of $s_1 \in S$ if for each source $s_2 \in S_p$, s is an INTERSECTION-COVER for s_1 and s_2.

We now introduce one final piece of notation. Assume $S_p \subset S$ be a subset of the source vertices in the DAG $H^{\rightarrow}$. Let x be some source vertex in S that does not belong to S_p. Let $\mathcal{T}_{S_p}$ denote a partial DAG tree decomposition of width one for S_p. Now, consider a leaf node ℓ in $\mathcal{T}_{S_p}$. Let d denote the only node in $\mathcal{T}_{S_p}$ that is adjacent to ℓ. We claim that if d is an INTERSECTION-COVER for ℓ and x, then we can construct a partial DAG tree decomposition for $S_p \cup \{x\}$ of width one (we will make this more formal and precise in the following paragraph). We identify such source and partial DAG tree decomposition pair $(x, \mathcal{T}_{S_p})$ as a GOOD-PAIR.

Definition 4.7 (GOOD-PAIR)**.** Let $x \in S(H^{\rightarrow})$ be a source vertex and $\mathcal{T}_{S_p}$ be a partial DAG tree decomposition of width one for $S_p \subset S(H^{\rightarrow})$ where $x \notin S_p$. We call the pair $(x, \mathcal{T}_{S_p})$ a GOOD-PAIR if there exists a leaf node $\ell \in \mathcal{T}_{S_p}$ connected to the node $d \in \mathcal{T}_{S_p}$ such that d is an INTERSECTION-COVER for x and ℓ.

We prove a final technical lemma that provides insight into the process of adding a new source vertex to an existing partial DAG tree decomposition of width one.

Lemma 4.8. *Let $H^{\rightarrow}$ be a DAG of H with sources S and $S_p \subset S$ be a subset of S. Assume $\mathcal{T}$ is a partial DAG tree decomposition for S_p with $\tau(\mathcal{T}) = 1$. Consider a source $s \in S$ such that $s \notin S_p$. If $d \in S_p$ is a S_p-COVER of s, then connecting s to d in $\mathcal{T}$ as a leaf results in a tree $\mathcal{T}'$ that is a partial DAG tree decomposition for $S_p \cup \{s\}$. Furthermore, $\tau(\mathcal{T}') = 1$.*

Proof. We first prove that $\mathcal{T}'$ is a partial DAG tree decomposition for $S_p \cup \{s\}$. The properties (1) and (2) of partial DAG tree decomposition (see Definition 4.5) trivially hold for $\mathcal{T}'$. If $\mathcal{T}$ has one or two nodes, then by definition of S_p-COVER, $\mathcal{T}'$ satisfies property (3). So we assume $\mathcal{T}$ has at least 3 nodes.

Note that $\mathcal{T}$ and $\mathcal{T}'$ are identical barring the leaf node s. Hence, for any three nodes s_1, s_2, and s_3 in T' with $s \notin \{s_1, s_2, s_3\}$, property (3) of partial DAG tree decomposition (Definition 4.5) holds. Now, consider s with two other nodes s_1 and s_2 in $\mathcal{T}'$ where s_1 is on the unique path between s and s_2. If $s_1 = d$, then property (3) holds as d is a S_p-COVER of s. So assume $s_1 \neq d$. But then, s_1 is on the unique path between d and s_2 (by construction of $\mathcal{T}'$). Since property (3) holds for d, s_1, and s_2 in $\mathcal{T}$, we have $\textsc{reachable}(d) \cap \textsc{reachable}(s_2) \subseteq \textsc{reachable}(s_1)$. We also have $\textsc{reachable}(s) \cap \textsc{reachable}(s_2) \subseteq \textsc{reachable}(d)$ as d is a S_p-COVER of s. Hence, $\textsc{reachable}(s) \cap \textsc{reachable}(s_2) \subseteq \textsc{reachable}(s_1)$. Therefore, property (3) holds. Thus, $\mathcal{T}'$ is a partial DAG tree decomposition of $S_p \cup \{s\}$. As $\tau(\mathcal{T}) = 1$, it follows immediately from the construction that $\tau(\mathcal{T}') = 1$. □

We now have all the ingredients to prove Lemma 4.4. For the sake of completeness, we restate the lemma.

Lemma 4.4. *For every simple graph H, if $\mathrm{LICL}(H) \leqslant 5$ then $\tau(H) = 1$.*

Proof. The DAG treewidth of a simple graph H is defined as the maximum DAG treewidth of any DAG $H^{\rightarrow}$ obtained from H. So, we prove that $\tau(H^{\rightarrow}) = 1$ for each DAG $H^{\rightarrow}$ of H. In the rest of the proof, we fix a DAG of H, and call it $H^{\rightarrow}$. Let $S(H^{\rightarrow})$ denote the set of all source vertices in $H^{\rightarrow}$. When $H^{\rightarrow}$ is clear from the context, we simply use S.

Let $S_p \subseteq S$ denote a subset of S. We prove by induction on the size of the subset S_p that there exists a partial DAG tree decomposition (Definition 4.5) of width one for each $S_p \subseteq S$. In particular, when $S_p = S$, it follows that there exists a DAG tree decomposition for $H^{\rightarrow}$ of width one.

The base case of $|S_p| = 1$ is trivial: put the only source in S_p in a bag B as the only node in the partial DAG tree decomposition for $H^{\rightarrow}$. Similarly, for $|S_p| = 2$, put the two sources in two

separate bags and connect them by an edge in the partial DAG tree decomposition for $H^{\rightarrow}$. The resulting tree is a partial DAG tree decomposition of width one. Now assume that, it is possible to build a partial DAG tree decomposition of width one for any subset $S_p \subset S$ where $|S_p| \leqslant r$, and $1 \leqslant r < |S|$. We show how to construct a partial DAG tree decomposition of width one for any subset $S_p \subseteq S$ where $|S_p| = r + 1$ for $r \geqslant 2$.

Fix a subset $S_{r+1} \subseteq S$ of size $r + 1$. Consider an arbitrary source $x \in S_{r+1}$. By induction hypothesis, we can construct a partial DAG tree decomposition of width one for the set $S_{-x} = S_{r+1} \setminus \{x\}$. We call the tree $\mathcal{T}_{-x}$. Now recall that, we call the pair $(x, \mathcal{T}_{-x})$ a GOOD-PAIR if there exists a leaf node $\ell \in \mathcal{T}_{-x}$ connected to the node $d \in \mathcal{T}_{-x}$ such that d is an INTERSECTION-COVER for x and ℓ (see Definition 4.7). We argue the existence of a GOOD-PAIR $(x, \mathcal{T}_{-x})$ and give a constructive process to find a width one partial DAG tree decomposition of S_{r+1} from such a GOOD-PAIR $(x, \mathcal{T}_{-x})$.

A good-pair leads to a partial DAG tree decomposition of width one. We first show that if there exists a source $x \in S_{r+1}$ and a width one partial DAG tree decomposition $\mathcal{T}_{-x}$ for $S_{-x} = S_{r+1} \setminus \{x\}$ such that $(x, \mathcal{T}_{-x})$ is a GOOD-PAIR, then there exists a width one partial DAG tree decomposition for S_{r+1}. In fact, we give a simple constructive process to find such a partial DAG tree decomposition: construct a width one partial DAG tree decomposition $\mathcal{T}_{-\ell}$ for $S_{-\ell} = S_{r+1} \setminus \{\ell\}$, and then connect ℓ as a leaf to d in $\mathcal{T}_{-\ell}$.

Claim 4.9. *Let $x \in S_{r+1}$ be a source vertex and $\mathcal{T}_{-x}$ be a width one partial DAG tree decomposition for $S_{-x} = S_{r+1} \setminus \{x\}$ such that $(x, \mathcal{T}_{-x})$ is a* GOOD-PAIR. *Then, there exists a partial DAG tree decomposition $\mathcal{T}$ for S_{r+1} with $\tau(\mathcal{T}) = 1$.*

Proof. Since $(x, \mathcal{T}_{-x})$ is a GOOD-PAIR, there exists a leaf node $\ell \in \mathcal{T}_{-x}$ connected to the node $d \in \mathcal{T}_{-x}$ such that d is an INTERSECTION-COVER for x and ℓ. We build a partial DAG tree decomposition of width one for $S_{-\ell} = S_{r+1} \setminus \{\ell\}$ (such a tree exists by induction hypothesis), and then add ℓ as a leaf node to the node d. We prove that the resulting tree, denoted as $\mathcal{T}$, is partial DAG tree decomposition for S_{r+1} with $\tau(\mathcal{T}) = 1$.

Since ℓ is only connected to d in $\mathcal{T}_{-x}$, d is a S_{-x}-COVER of ℓ. Also, d is an INTERSECTION-COVER of ℓ and x, so d is a S_{r+1}-COVER of ℓ. Therefore, by applying Lemma 4.8, it follows that $\mathcal{T}$ is a partial DAG tree decomposition of S_{r+1} with $\tau(\mathcal{T}) = 1$. □

Existence of a good-pair. We have shown how to construct a partial DAG tree decomposition for the set S_{r+1} if there exists a GOOD-PAIR $(x, \mathcal{T}_{-x})$ where x is a source in S_{r+1} and $\mathcal{T}_{-x}$ is a width one partial DAG tree decomposition for S_{-x}. We now show that for any set S_{r+1}, there always exists a GOOD-PAIR $(x, \mathcal{T}_{-x})$.

Claim 4.10. *There exists a vertex $x \in S_{r+1}$ and a width one partial DAG tree decomposition $\mathcal{T}_{-x}$ for $S_{-x} = S_{r+1} \setminus \{x\}$, such that $(x, \mathcal{T}_{-x})$ is a* GOOD-PAIR.

Proof. Assume for contradiction, the claim is false. Consider the unique reachability graph on the vertex set S_{r+1}, denoted by $\mathrm{UR}_{S_{r+1}}(S_{r+1}, E_{S_{r+1}})$ (see Definition 4.2). Let $x \in S_{r+1}$ be an arbitrary source vertex. By assumption, $(x, \mathcal{T}_{-x})$ is not a GOOD-PAIR. So, for each leaf node $\ell \in \mathcal{T}_{-x}$ connected to the node $d \in \mathcal{T}_{-x}$, d is not an INTERSECTION-COVER for x and ℓ. Then, there exists a vertex v in $H^{\rightarrow}$, such that $v \in \textsc{reachable}(x) \cap \textsc{reachable}(\ell)$, but $v \notin \textsc{reachable}(d)$. On the other hand, by construction, d is a S_{-x}-COVER for ℓ (d is the only node connected to ℓ in $\mathcal{T}_{-x}$). Hence, v is reachable from none of the source vertices in S_{-x}, other than ℓ. Therefore, the edge $\{x, \ell\} \in E_{S_{r+1}}$. Now, $\mathcal{T}_{-x}$ has at least two leaves, so the degree of the source vertex x in $\mathrm{UR}_{S_{r+1}}$ is at least 2. The same argument holds for each $x \in S_{r+1}$. Hence, the degree of each vertex in

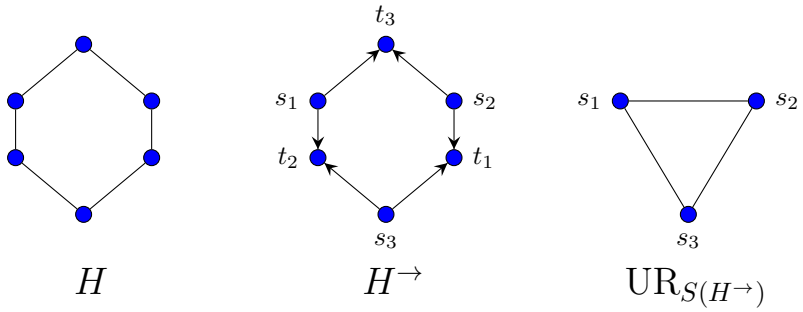


Figure 2: Let H be a six cycle. In the middle figure, we show the DAG $H^{\rightarrow}$ of H for which $\tau(H^{\rightarrow}) \geqslant 2$. On the right, we show the UR graph corresponding to $H^{\rightarrow}$. It is a triangle as t_1 is only reachable from $\{s_2, s_3\}$, t_2 is only reachable from $\{s_1, s_3\}$, and t_3 is only reachable from $\{s_1, s_2\}$.

$\mathrm{UR}_{S_{r+1}}$ is at least two. Now $|S_{r+1}| \geqslant 3$ (recall $r \geqslant 2$), thus there exists a cycle $\mathcal{C}$ in $\mathrm{UR}_{S_{r+1}}$ of length at least 3. By applying Lemma 4.3, we have $\mathrm{LICL}(H) \geqslant 6$. But $\mathrm{LICL}(H) \leqslant 5$, so this leads to a contradiction. Hence, the claim is true. □

We proved by induction that for any non-empty subset $S_p \subseteq S$, there exist a partial DAG tree decomposition for S_p with width one. In the case when $S_p = S$, the partial DAG tree decomposition is a DAG tree decomposition for $H^{\rightarrow}$. This completes the proof of Lemma 4.4. □

4.3 DAG Treewidth for Graphs with LICL at least Six

In this section, we prove the following lemma.

Lemma 4.11. *For every simple graph H, if* $\mathrm{LICL}(H) \geqslant 6$ *then* $\tau(H) \geqslant 2$.

We first discuss the simple case of the 6-cycle. Note that, to prove that $\tau(H) \geqslant 2$, it suffices to show that there exists a DAG $H^{\rightarrow}$ of H such that $\tau(H^{\rightarrow}) \geqslant 2$. Let $H^{\rightarrow}$ be a DAG of H as shown in the middle figure in Fig. 2. Let $S = \{s_1, s_2, s_3\}$ be the set of sources in $H^{\rightarrow}$. Consider the unique reachability graph $\mathrm{UR}_S(S, E_S)$, shown on the right in Fig. 2. The graph UR_S is a triangle: t_1 is not reachable from s_1, but reachable from s_2 and s_3, and so on. In any DAG tree decomposition $\mathcal{T}$ of $H^{\rightarrow}$ with width one, all source vertices are a vertex of $\mathcal{T}$ by themselves. So at least one of s_1, s_2, or s_3 (say s_1) would be on the unique path between the other two. But this would violate property (3) of DAG tree decomposition. It follows that $\tau(H) \geqslant 2$. In this case, it is not difficult to argue that $\tau(H) = 2$.

We formalize this intuition to prove in the following lemma: if the UR_S graph of a DAG $H^{\rightarrow}$ with source vertices S has a triangle in it, then it must the case that $\tau(H^{\rightarrow}) \geqslant 2$. Then, to prove $\tau(H) \geqslant 2$ for a graph H, it is sufficient to show the existence of a DAG $H^{\rightarrow}$ such that the corresponding $\mathrm{UR}_{S(H^{\rightarrow})}$ has a triangle.

Lemma 4.12. *Let $H^{\rightarrow}$ be a DAG of H with source vertices S and* $\mathrm{UR}_S(S, E_S)$ *be the* unique reachability *graph for the set S. If* UR_S *has a triangle, then* $\tau(H) \geqslant 2$.

Proof. Assume for contradiction, $\tau(H^{\rightarrow}) = 1$ and $\mathcal{T}$ be a DAG tree decomposition of width one. Let $\{s_1, s_2, s_3\}$ be a triangle in the graph UR_S. As $\mathcal{T}$ is a DAG tree decomposition with width one, all source vertices in S must be a node by themselves in $\mathcal{T}$. Observe that, there must exists a node $s \in \mathcal{T}$ that is between the unique path for a pair of nodes in $\{s_1, s_2, s_3\}$. As otherwise, these three

nodes are all pairwise connected by an edge forming a triangle in $\mathcal{T}$. Wlog, assume s is on the unique path between s_1 and s_2 in $\mathcal{T}$. Since, $\{s_1, s_2\}$ is an edge in UR_S, by definition, there exists a vertex $t_{s_1,s_2} \in V(H^{\rightarrow})$, such that $t \in \textsc{reachable}_{H^{\rightarrow}}(s_1) \cap \textsc{reachable}_{H^{\rightarrow}}(s_2)$, but $t \notin \textsc{reachable}_{H^{\rightarrow}}(s)$. But, this violates the property (3) of DAG tree decomposition in Definition 3.3. So such a tree $\mathcal{T}$ cannot exists and hence, $\tau(H^{\rightarrow}) \geqslant 2$. Therefore, $\tau(H) \geqslant 2$. □

We are now ready to prove the main lemma. We restate the lemma for completeness.

Lemma 4.11. *For every simple graph H, if $\mathrm{LICL}(H) \geqslant 6$ then $\tau(H) \geqslant 2$.*

Proof. Let $\mathrm{LICL}(H) = r$, where $r \geqslant 6$ and $r = 3\ell + q$, for some $\ell \geqslant 2$ where $q \in \{0, 1, 2\}$. Assume $C = v_1, v_2, \ldots, v_r, v_1$ is an induced cycle of length r in H. We construct a DAG $H^{\rightarrow}$ as follows.

Consider an edge $e = (u, v)$ in H. Assume only one of the end point does not belong to $V(C)$ — say $u \notin V(C)$. Then we orient the edge from v to u. Now consider the case when both $u, v \notin V(C)$. We orient such edges in an arbitrary manner ensuring the resulting orientation is acyclic. We now describe the orientation of the edges on the r-cycle C. We mark three vertices s_1, s_2 and s_3 in C as sources that are at least distance two apart from each other. Wlog, assume $s_1 = v_1$, $s_2 = v_{\ell+1}$, and $s_3 = v_{2\ell+1}$. Now we mark three vertices t_1, t_2, and t_3 as sinks such that t_1 is between s_1 and s_2, t_2 is between s_2 and s_3, and t_3 is between s_3 and s_1 in the cycle C. Again, wlong assume $t_1 = v_2$, $t_2 = v_{\ell+2}$, and $t_3 = v_{2\ell+2}$. Finally, orient the edges in C towards the sink vertices and away from the sources. This completes the description of $H^{\rightarrow}$.

Now let S denote the set of source vertices in $H^{\rightarrow}$. Consider the unique reachability graph $\mathrm{UR}_S(S, E_S)$. We claim that UR_S includes a triangle. Indeed, we show that $\{s_1, s_2, s_3\}$ forms a triangle in UR_S. We first argue the existence of the edge $\{s_1, s_2\} \in E_S$. The vertex t_1 is reachable from s_1 and s_2, but not from s_3. Since all the edges that are not part of the cycle C, are oriented outwards from the vertices in C, no other source vertices in S can reach t_1 in $H^{\rightarrow}$. Hence, $\{s_1, s_2\} \in E_S$. Similarly, we can argue the existence of the edges $\{s_2, s_3\}$ and $\{s_1, s_3\}$ in E_{S_p}. Applying Lemma 4.12, it follows that $\tau(H) \geqslant 2$. □

5 LICL and the Homomorphism Counting Lower Bound

In this section, we prove our main lower bound result. We show that for a pattern graph H with $\mathrm{LICL}(H) \geqslant 6$, the $\#\mathrm{Hom}_H$ problem does not admit a linear time algorithm in bounded degeneracy graphs, assuming the Triangle Detection Conjecture (Conjecture 1.2). We state our main theorem below.

Theorem 5.1. *Let H be a pattern graph on k vertices with $\mathrm{LICL}(H) \geqslant 6$. Assuming the Triangle Detection Conjecture, there exists an absolute constant $\gamma > 0$ such that for any function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, there is no (expected) $f(\kappa, k)o(m^{1+\gamma})$ algorithm for the $\#\mathrm{Hom}_H$ problem, where m and κ are the number of edges and the degeneracy of the input graph, respectively.*

Outline of the Proof. We first present an outline of our proof; the complete proof is discussed in Section 5.1. Let tri-cnt denote the problem of counting the number of triangles in a graph. We prove the theorem by a linear time Turing reduction from the tri-cnt problem to the $\#\mathrm{Hom}_H$ problem. Assuming the Triangle Detection Conjecture, any algorithm (possibly randomized) for the tri-cnt problem requires $\omega(m)$ time, where m is the number of edges in the input graph. Given an input instance G of the tri-cnt problem, we construct a graph G_H with bounded degeneracy and $O(|E(G)|)$ edges. We show how the number of triangles in G can be obtained by counting specific homomorphisms of H in G_H.

Let $\text{LICL}(H) = r$ and C be one of the largest induced cycles in H; let $V(C)$ denotes its vertices. We now describe the construction of the graph G_H. The graph G_H has two main parts: (1) the fixed component, denoted as G_{FIXED} (this part is independent of the input graph G and only depends on the pattern graph H) and (2) the core component, denoted as G_{CORE}. Additionally, there are edges that connect these two components, denoted by E_{BRIDGE}. Let $H_{C\text{-EXCLUDED}}$ denote the graph after we remove $V(C)$ from H. More formally, $H_{C\text{-EXCLUDED}} = H - V(C)$. The fixed component G_{FIXED} is a copy of $H_{C\text{-EXCLUDED}}$.

Next, we give an intuitive account of our construction. We discuss the role of G_{CORE} and how its connection to G_{FIXED} through E_{BRIDGE} ensures that the number of triangles in G can be obtained by counting homomorphisms of H. Then we give an overview of the construction.

Intuition behind the Construction. The main idea is to construct G_{CORE} and E_{BRIDGE} in such a way that each triangle in G transforms to an r-cycle in G_{CORE}, that then composes a match of H together with G_{FIXED} (recall that G_{FIXED} is a copy of $H_{C\text{-EXCLUDED}}$). To this end, we design G_{CORE} in r parts, ensuring the following properties hold for each r-cycle in G_{CORE} that contains exactly one vertex in each of these r parts: (1) It composes a match of H together with G_{FIXED} and (2) It corresponds to a triangle in G. Let $\mathcal{P}'$ denote the partition of $V(G_{\text{CORE}})$ into these r parts. Further, assume we construct G_H in a way that, each match of H that contains the vertices of G_{FIXED} and exactly one vertex in each set $V \in \mathcal{P}'$, corresponds to one of the r-cycles described above. It is now easy to see that if we can count these matches of H, we can then obtain the number of the described r-cycles in G_{CORE} and hence the number of triangles in G.

Consider a partition $\mathcal{P}$ of $V(G_H)$ where $|\mathcal{P}| = k$. Assume there is a linear time algorithm ALG for $\#\text{Hom}_H$ in bounded degeneracy graphs. Then, Lemma 5.3 proves that there exists a linear time algorithm that, using ALG, counts the matches of H in G that include exactly one vertex in each set $V \in \mathcal{P}$. These matches are called $\mathcal{P}$-matches of H, as we define formally later in Definition 5.2. Also, each r-cycles in G_{CORE} that contain exactly one vertex in each set $V \in \mathcal{P}'$ is a $\mathcal{P}'$-match of C. Now, we define the partition $\mathcal{P}$ of $V(G_H)$ as follows; $\mathcal{P}$ includes each set in $\mathcal{P}'$ and each of the $k - r$ vertices in G_{FIXED} as a set by itself.

Overall, by construction of G_H, we can get the number of triangles in G by the number of $\mathcal{P}'$-matches of C in G_{CORE}. Further, we can obtain the number of $\mathcal{P}'$-matches of C in G_{CORE} by the number of $\mathcal{P}$-matches of H in G_H that we count using ALG. The following restates the desired properties of G_H we discussed, more formally.

(I) There is a bijection between the set of $\mathcal{P}$-matches of H in G_H and the set of $\mathcal{P}'$-matches of C in G_{CORE}.

(II) The number of triangles in G is a simple linear function of the number of $\mathcal{P}'$-matches of C in G_{CORE}.

Next, we give an overview of the construction of G_H for $r = 6$. We prove that properties (I) and (II) hold for our construction in the general case in Lemma 5.6 and Lemma 5.7, respectively.

Overview of the Construction. In what follows, we give an overview of G_{FIXED}, G_{CORE}, and E_{BRIDGE}. For the ease of presentation, we assume $r = 6$.

(1) G_{FIXED} is a copy of $H_{C\text{-EXCLUDED}}$. We denote the vertex set in G_{FIXED} as $V_{\text{FIXED-SET}}$. Observe that, G_{FIXED} does not depend on the input graph G.

(2) The core component G_{CORE} consists of two set of vertices: $V_{\text{CORE-SET}}$ and $V_{\text{AUXILIARY-SET}}$. We first discuss the sets $V_{\text{CORE-SET}}$ and $V_{\text{AUXILIARY-SET}}$, and then introduce the edge set $E(G_{\text{CORE}})$.

(a) $V_{\text{CORE-SET}}$ consists of three set of vertices, $V_1 = \{w_1, \ldots, w_n\}$, $V_2 = \{x_1, \ldots, x_n\}$, and $V_3 = \{y_1, \ldots, y_n\}$ — each of size n (recall $|V(G)| = n$). The vertices in each of these sets correspond to the vertices in $V(G) = \{u_1, \ldots, u_n\}$.

(b) The construction of $V_{\text{AUXILIARY-SET}}$ depends on r. For $r = 6$, it consists of three sets, denoted as $V_{1,2}$, $V_{2,3}$, and $V_{1,3}$ — each of size $2m$ (recall $|E(G)| = m$). The vertices in each of these sets corresponds to the edges in $E(G)$. We index them using e, for each $e \in E(G)$: $V_{1,2} = \{v_e^{1,2}, v_e^{2,1}\}_{e \in E(G)}$, and so on. The role of these sets will become clear as we describe the edges of G_{CORE}.

(c) Consider an edge $e = \{u_i, u_j\} \in E(G)$ and the pair V_1 and V_2. We connect the vertex $w_i \in V_1$ to the vertex $x_j \in V_2$ by a 2-path via the vertex $v_e^{1,2} \in V_{1,2}$. Similarly, we connect the vertex w_j to the vertex x_i by a 2-path via the vertex $v_e^{2,1}$. In particular, we add the edges $\{w_i, v_e^{1,2}\}$ and $\{v_e^{1,2}, x_j\}$, and the edges $\{w_j, v_e^{2,1}\}$ and $\{v_e^{2,1}, x_i\}$ to the set $E(G_{\text{CORE}})$. We repeat the process for the pairs (V_2, V_3) and (V_1, V_3) for each edge $e \in E(G)$.

(3) We now describe the edge set E_{BRIDGE} that serves as connections between G_{FIXED} and G_{CORE}. Let σ_{BRIDGE} be a bijective mapping between the two sets $V(C)$ and $\{V_1, V_2, V_3, V_{1,2}, V_{2,3}, V_{1,3}\}$; $\sigma_{\text{BRIDGE}} : V(C) \to \{V_1, V_{1,2}, V_2, V_{2,3}, V_3, V_{1,3}\}$. For each edge $e = \{u, v\} \in E(H)$ such that $u \in V(C)$ and $v \notin V(C)$, we do the following. Let $z_v \in V_{\text{FIXED-SET}}$ denote the vertex corresponding to the vertex v (recall $G_{\text{FIXED-SET}}$ is a copy of $H_{C\text{-EXCLUDED}}$). We connect z_v to all the vertices in the set $\sigma_{\text{BRIDGE}}(u)$ and add these edges to E_{BRIDGE}.

Note that, here $\mathcal{P}' = \{V_1, V_{1,2}, V_2, V_{2,3}, V_3, V_{1,3}\}$. Before diving into the details of deriving the triangle counts in G, we first take an example pattern graph H to visually depict the constructed graph G_H (see Figure 3) and discuss why properties (I) and (II) hold in our construction.

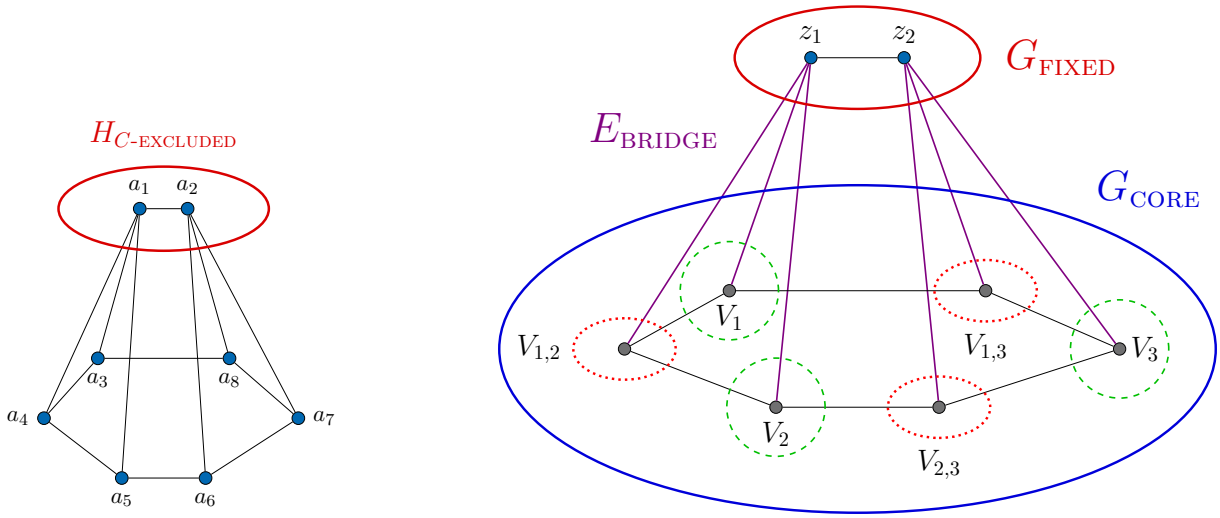
An Illustrative Example. Let H be the graph as shown in Figure 3a. In this example, $\text{LICL}(H) = 6$. Let $C = a_3, a_4, a_5, a_6, a_7, a_8, a_3$ be the induced 6-cycle in H. We demonstrate the constructed graph G_H in Figure 3b. We now discuss the various components of G_H.

(1) The graph G_{FIXED} is shown by the red oval. The vertices z_1 and z_2 compose $V_{\text{FIXED-SET}}$, where z_1 corresponds to a_1 and z_2 corresponds to a_2.

(2) The graph G_{CORE} is shown by the blue oval. For each edge $e = \{u_i, u_j\} \in E(G)$, (for some input graph G, which is not shown in the figure), we add a total of six 2-paths: two between each pair of sets from $\{V_1, V_2, V_3\}$. For instance, between the set V_1 and V_2 these 2-paths are as follows: $\{w_i, v_e^{1,2}, x_j\}$ and $\{w_j, v_e^{2,1}, x_i\}$. The vertices w_i, w_j belong to V_1; x_i, x_j belong to V_2; and $v_e^{1,2}, v_e^{2,1}$ belong to $V_{1,2}$.

(3) Finally we describe the edge set E_{BRIDGE} (the edges in violet). We consider the following bijective mapping σ_{BRIDGE}: $\sigma_{\text{BRIDGE}}(a_3) = V_1$, $\sigma_{\text{BRIDGE}}(a_4) = V_{1,2}$, $\sigma_{\text{BRIDGE}}(a_5) = V_2$, $\sigma_{\text{BRIDGE}}(a_6) = V_{2,3}$, $\sigma_{\text{BRIDGE}}(a_7) = V_3$, $\sigma_{\text{BRIDGE}}(a_8) = V_{1,3}$. Now consider the edge $\{a_3, a_1\} \in E(G)$; $a_3 \in V(C)$ and $a_1 \notin V(C)$. So we connect z_1 (the vertex corresponding to a_1) to each vertex in the set $\sigma_{\text{BRIDGE}}(a_3) = V_1$. We repeat the same process for each edge $\{u, v\}$ in $E(H)$ where $u \in V(C)$ and $v \notin V(C)$.

Observe that in this example, $\mathcal{P} = \{\{z_1\}, \{z_2\}, V_1, V_{1,2}, V_2, V_{2,3}, V_3, V_{1,3}\}$. It is easy to see that E_{BRIDGE} connects G_{CORE} to G_{FIXED}, such that each 6-cycle in G_{CORE} compose a match of H together with G_{FIXED}. Each $\mathcal{P}$-match of H is actually an induced match as the only edges between its vertices



(a) The pattern graph H

(b) The constructed graph G_H. In G_{CORE}, we only depict six edges corresponding to a triangle in G (there would be six more edges corresponding to the same triangle, that we do not show here). Also, we only depict the vertices relevant to the triangle.

Figure 3: G_H constructed for an example pattern graph H

in G_H are the edges that correspond to the match. Therefore, in this example, each $\mathcal{P}$-match of H in G_H include a 6-cycle in G_{CORE} that is actually a $\mathcal{P}'$-match of C. Thus, property (I) holds.

It is not difficult to see that a triangle in G introduces a total of six many 6-cycle in G_{CORE} that are $\mathcal{P}'$-matches of C in G_{CORE}. The converse follows as each $\mathcal{P}'$-match of C, which is a 6-cycle in G_{CORE}, must contain exactly one vertex from each of the three sets in each of $V_{\text{CORE-SET}}$ and $V_{\text{AUXILIARY-SET}}$. So, we could obtain the number of triangles in G by dividing the number of $\mathcal{P}'$-matches of C in G_{CORE} by six. Thus, property (II) holds.

Deriving The Triangle Counts in G. So far, we have shown that properties (I) and (II) hold in G_H for our construction. Therefore, the number of $\mathcal{P}$-matches of H in G_H reveals the number of triangles in G. However, we are interested in utilizing the homomorphism count of H to derive the triangle count in G. Indeed, we obtain the number of $\mathcal{P}$-matches of H in G_H by carefully looking at "restricted" homomorphisms from H to G. One crucial property of the graph G_H that we will require is bounded degeneracy. In fact, our construction of the graph G_H ensures that it has constant degeneracy irrespective of the degeneracy of G (we will formally prove this later in Lemma 5.5).

Let ALG be an algorithm for the $\#\text{Hom}_H$ problem, that runs in $f(\kappa, k) \cdot O(m)$ time for some explicit function f, where m and κ are the number of edges and degeneracy of the input graph, respectively. Then, we can use ALG to count the homomorphisms from H to any subgraph of G_H in time $f(\kappa(G_H), k) \cdot O(m)$. Note that, here we use the fact that for any subgraph G'_H of G_H, $\kappa(G'_H) \leqslant \kappa(G_H)$.

We now solve the final missing piece of the puzzle: how to count the number of $\mathcal{P}$-matches of H in G_H using ALG? We present a two step solution to this question. First, we count the number of "$\mathcal{P}$ restricted" homomorphisms, denoted by $\mathcal{P}$-homomorphism and defined in Definition 5.2, from H to G_H by running ALG on carefully chosen subgraphs of G_H. Intuitively, a "$\mathcal{P}$ restricted" homomorphism is a homomorphism from H to G_H that involves at least one vertex in each part of $\mathcal{P}$. Second, we use the count from the first step to derive the number of $\mathcal{P}$-matches of H in G_H.

We present this in Lemma 5.3.

We now formally define $\mathcal{P}$-match and $\mathcal{P}$-homomorphism.

Definition 5.2 ($\mathcal{P}$-match and $\mathcal{P}$-homomorphism)**.** Let $\mathcal{P} = \{V_1, \ldots, V_k\}$ be a partition of the vertex set $V(G)$ of the input graph G where $|V(H)| = k$ for the pattern graph H. Further assume $|V_i| \geqslant 1$ for each $i \in [k]$. Let $G_{\text{H-match}}$ be a subgraph of G such that $G_{\text{H-match}}$ is a match of H. We call $G_{\text{H-match}}$ a $\mathcal{P}$-match, if it includes exactly one vertex from each set V_i in $\mathcal{P}$: $|V(G_{\text{H-match}}) \cap V_i| = 1$ for each $i \in [k]$. Let $\pi : V(H) \to V(G)$ be a homomorphism from H to G. We call π a $\mathcal{P}$-homomorphism, if the image of π is non-empty in each set V_i: $|\{v : \pi(u) = v \text{ for } u \in V(H)\} \cap V_i| \geqslant 1$ for each $i \in [k]$.

In the following lemma, we prove that it is possible to count the number of $\mathcal{P}$-matches of H in G_H by running ALG on suitably chosen 2^k many subgraphs of G_H.

Lemma 5.3. *Assume that* ALG *is an algorithm for the* $\#\text{Hom}_H$ *problem that runs in time* $O(mf(\kappa, k))$ *for some function* f*, where* $m = E(G)$ *and* $\kappa = \kappa(G)$ *for the input graph* G*, and* $k = V(H)$*. Let* $\mathcal{P} = \{V_1, \ldots, V_k\}$ *be a partition of* $V(G)$ *with* $|V_i| \geqslant 1$ *for each* $i \in [k]$*. Then, there exists an algorithm that counts the number of* $\mathcal{P}$-match *of* H *in* G *with running time* $O(2^k \cdot mf(\kappa, k))$.

Proof. Let $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{2^k-1}$ be the non-empty subfamilies of $\mathcal{P}$. Let $G_1, G_2, \ldots, G_{2^k-1}$ be the subgraphs of G where G_i is induced on the vertex set $V(G) \setminus (\bigcup_{S \in \mathcal{F}_i} S)$, for $i \in [2^k - 1]$. Note that a homomorphism from H to any subgraphs G_i, for $i \in [2^k - 1]$, is also a homomorphism from H to G. Since each G_i is a subgraph of G, the degeneracy $\kappa(G_i) \leqslant \kappa$. Then, ALG can count homomorphisms from H to any G_i in time $O(mf(\kappa, k))$. Using the inclusion-exclusion principle, we can obtain the number of homomorphisms from H to G that are also a homomorphism from H to at least one of the subgraphs in $\{G_1, G_2, \ldots, G_{2^k-1}\}$ in $O(2^k \cdot mf(\kappa, k))$. Hence, we can obtain the number of $\mathcal{P}$-homomorphisms from H to G as follows,

$$\text{Hom}_H(G) - \sum_{1 \leqslant i \leqslant 2^k - 1} (-1)^{|F_i|-1} \text{Hom}_H(G_i)\,.$$

Note that if a homomorphism from H to G does not include any vertex in a set V_i in $\mathcal{P}$, then it is also a homomorphism from H to at least one of the subgraphs in $\{G_1, G_2, G_{2^k-1}\}$. Thus, we do not count such a homomorphism from H to G. Since $k = |V(H)|$, $\mathcal{P}$-homomorphisms of H in G are actually embeddings of H in G that involve exactly one vertex in each part of P. Observe that each such embedding of H in G corresponds to a $\mathcal{P}$-match of H in G. For each match $G_{\text{H-match}}$ of H in G, there are $|Aut(H)|$ embeddings of H in G that map H to $G_{\text{H-match}}$. Thus, by dividing the number of $\mathcal{P}$-homomorphisms from H to G by $|Aut(H)|$, we obtain the number of $\mathcal{P}$-matches of H in G in $O(2^k \cdot mf(\kappa, k))$ time. □

5.1 Proof of Main Theorem

We now present the details of the construction of G_H for the general case and prove Theorem 5.1.

Proof of Theorem 5.1. We present a linear time Turing reduction form the TRI-CNT problem to the $\#\text{Hom}_H$ problem in bounded degeneracy graphs. Let G be the input instance of the TRI-CNT problem where $V(G) = \{u_1, \ldots, u_n\}$ and $|E(G)| = m$. First, we construct a graph G_H based on G and H such that G_H has bounded degeneracy and $O(m)$ edges.

Construction of G_H. Let $\text{LICL}(H) = r$ where $r \geqslant 6$; and $V(H) = \{a_1, a_2, \ldots, a_k\}$, where $a_{k-r+1}, a_{k-r+2}, \ldots, a_k, a_{k-r+1}$ is an induced r-cycle C. Let $H_{C\text{-EXCLUDED}}$ denote $H - V(C)$. G_H

has two main parts, G_{FIXED} and G_{CORE}. These two parts are connected by the edge set E_{BRIDGE}. G_{FIXED} is a copy of $H_{C\text{-EXCLUDED}}$ and has the vertex set $V_{\text{FIXED-SET}} = \{z_1, z_2, \ldots, z_{k-r}\}$. $z_i \in V_{\text{FIXED-SET}}$ corresponds to a_i in $H_{C\text{-EXCLUDED}}$ for $i \in [k-r]$. Thus, $z_i, z_j \in V_{\text{FIXED-SET}}$ are adjacent iff $\{a_i, a_j\}$ is an edge in $H_{C\text{-EXCLUDED}}$.

G_{CORE} contains two sets of vertices, $V_{\text{CORE-SET}}$, and $V_{\text{AUXILIARY-SET}}$. Vertices in $V_{\text{CORE-SET}}$ correspond to vertices in $V(G)$, and vertices in $V_{\text{AUXILIARY-SET}}$ correspond to the edges in $E(G)$. $V_{\text{CORE-SET}}$ consists of three copies of $V(G)$ without any edges inside them. More precisely, $V_{\text{CORE-SET}}$ is composed of three sets of vertices $V_1 = \{w_1, \ldots, w_n\}$, $V_2 = \{x_1, \ldots, x_n\}$, and $V_3 = \{y_1, \ldots, y_n\}$. For $i \in [n]$, vertices $w_i \in V_1$, $x_i \in V_2$, and $y_i \in V_3$ correspond to $u_i \in V(G)$. There are no edges inside $V_{\text{CORE-SET}}$. We describe $V_{\text{AUXILIARY-SET}}$ next.

$V_{\text{AUXILIARY-SET}}$ corresponds to the vertices of the paths of length $r/3$ that we add between V_1, V_2, and V_3. Let $r = 3\ell + q$, for some $\ell \geqslant 2$ and $q \in \{0, 1, 2\}$. The vertices in $V_{\text{AUXILIARY-SET}}$ consists of the sets of vertices $V_{1,2}$, $V_{2,3}$, and $V_{1,3}$. For each edge $e \in E(G)$ and each pair in $\{V_1, V_2, V_3\}$, we add two sets of vertices to $V_{\text{AUXILIARY-SET}}$. Next, we describe the vertices we add to $V_{1,2}$, $V_{2,3}$, and $V_{1,3}$ for an edge $e \in E(G)$. For the pair V_1 and V_2, we add

$$V_e^{1,2} = \left\{v_{e,1}^{1,2}, \ldots, v_{e,\ell-1}^{1,2}\right\}$$
$$\text{and } V_e^{2,1} = \left\{v_{e,1}^{2,1}, \ldots, v_{e,\ell-1}^{2,1}\right\}$$

to $V_{1,2}$. For the pair V_2 and V_3, we add

$$V_e^{2,3} = \left\{v_{e,1}^{2,3}, \ldots, v_{e,\ell-1+\lfloor q/2 \rfloor}^{2,3}\right\}$$
$$\text{and } V_e^{3,2} = \left\{v_{e,1}^{3,2}, \ldots, v_{e,\ell-1+\lfloor q/2 \rfloor}^{3,2}\right\}$$

to $V_{2,3}$. And finally, for the pair V_1 and V_3, we add

$$V_e^{1,3} = \left\{v_{e,1}^{1,3}, \ldots, v_{e,\ell-1+\lfloor (q+1)/2 \rfloor}^{1,3}\right\}$$
$$\text{and } V_e^{3,1} = \left\{v_{e,1}^{3,1}, \ldots, v_{e,\ell-1+\lfloor (q+1)/2 \rfloor}^{3,1}\right\}$$

to $V_{1,3}$. The following defines $V_{1,2}$, $V_{2,3}$, and $V_{1,3}$ more formally. For $i, j \in \{1, 2, 3\}$ where $i < j$,

$$V_{i,j} = \bigcup_{e \in E(G)} V_e^{i,j} \cup V_e^{j,i}.$$

This completes the description of $V(G_{\text{CORE}})$. We describe $E(G_{\text{CORE}})$ next.

The edges inside G_{CORE} stitch vertices in $V_{\text{AUXILIARY-SET}}$ to form paths of length $r/3$ between each pair in $\{V_1, V_2, V_3\}$. $E(G_{\text{CORE-SET}})$ consists of three sets of edges, $E_{1,2}$, $E_{2,3}$, and $E_{1,3}$. For each edge in G and each pair in $\{V_1, V_2, V_3\}$, we add two sets of edges to G_{CORE}. We describe the edges we add to G_{CORE} for each edge $e = \{u_i, u_j\} \in E(G)$. For the pair V_1 and V_2, we add

$$E_e^{1,2} = \left\{(w_i, v_{e,1}^{1,2}), (v_{e,1}^{1,2}, v_{e,2}^{1,2}), \ldots, (v_{e,\ell-1}^{1,2}, x_j)\right\}$$
$$\text{and } E_e^{2,1} = \left\{(w_j, v_{e,1}^{2,1}), (v_{e,1}^{2,1}, v_{e,2}^{2,1}), \ldots, (v_{e,\ell-1}^{2,1}, x_i)\right\}$$

to $E_{1,2}$. Edges in $E_{1,2}$ form ℓ-paths between V_1 and V_2 with $V_{1,2}$ as interior vertices. For the pair V_2 and V_3, we add

$$E_e^{2,3} = \left\{(x_i, v_{e,1}^{2,3}), (v_{e,1}^{2,3}, v_{e,2}^{2,3}), \ldots, (v_{e,\ell-1+\lfloor q/2 \rfloor}^{2,3}, y_j)\right\}$$
$$\text{and } E_e^{3,2} = \left\{(x_j, v_{e,1}^{3,2}), (v_{e,1}^{3,2}, v_{e,2}^{3,2}), \ldots, (v_{e,\ell-1+\lfloor q/2 \rfloor}^{3,2}, y_i)\right\}$$

to $E_{2,3}$. Edges in $E_{2,3}$ compose $\ell + \lfloor q/2 \rfloor$-paths between V_2 and V_3, by joining the vertices in $V_{2,3}$. And, for the pair V_1 and V_3, we add

$$E_e^{1,3} = \left\{(w_i, v_{e,1}^{1,3}), (v_{e,1}^{1,3}, v_{e,2}^{1,3}), \ldots, (v_{e,\ell-1+\lfloor (q+1)/2 \rfloor}^{1,3}, y_j)\right\}$$
$$\text{and } E_e^{3,1} = \left\{(w_j, v_{e,1}^{3,1}), (v_{e,1}^{3,1}, v_{e,2}^{3,1}), \ldots, (v_{e,\ell-1+\lfloor (q+1)/2 \rfloor}^{3,1}, y_i)\right\}$$

to $E_{1,3}$. The edge set $E_{1,3}$ joins vertices in $V_{1,3}$ to form $\ell + \lfloor (q+1)/2 \rfloor$-paths between V_1 and V_3. We can describe the three sets of edges that compose $E(G_{\text{CORE}})$ more formally as follows. For $i, j \in \{1, 2, 3\}$ where $i < j$,

$$E_{i,j} = \bigcup_{e \in E(G)} E_e^{i,j} \cup E_e^{j,i}.$$

Now, we describe the edge set E_{BRIDGE} that connects G_{FIXED} and G_{CORE}. First, we partition $V_{1,2}$, $V_{2,3}$, and $V_{1,3}$ based on distance to V_1, V_2, and V_3, respectively. For instance, we define $V_{1,2}^i$ to be all the vertices in $V_{1,2}$ with i as the length of the shortest path to a vertex in V_1. Recall that each vertex in $V_{1,2}$ serves as an internal vertex of a path between a vertex in V_1 and a vertex in V_2. Formally, we define

$$V_{1,2}^i = \bigcup_{e \in E(G)} \{v_{e,i}^{1,2}, v_{e,i}^{2,1}\} \text{ for } i \in \{1, \ldots, \ell - 1\},$$
$$V_{2,3}^i = \bigcup_{e \in E(G)} \{v_{e,i}^{2,3}, v_{e,i}^{3,2}\} \text{ for } i \in \{1, \ldots, \ell - 1 + \lfloor q/2 \rfloor\},$$
$$\text{and } V_{1,3}^i = \bigcup_{e \in E(G)} \{v_{e,i}^{1,3}, v_{e,i}^{3,1}\} \text{ for } i \in \{1, \ldots, \ell - 1 + \lfloor (q+1)/2 \rfloor\}.$$

Now that we have partitioned $V_{\text{AUXILIARY-SET}}$, we add the sets V_1, V_2, and V_3 to this partition of $V_{\text{AUXILIARY-SET}}$ to define a partition $\mathcal{P}'$ of $V(G_{\text{CORE}})$ as follows.

$$\begin{aligned}\mathcal{P}' = \big\{&V_1, V_2, V_3,\\ &V_{1,2}^1, \ldots, V_{1,2}^{\ell-1},\\ &V_{2,3}^1, \ldots, V_{2,3}^{\ell-1+\lfloor q/2 \rfloor},\\ &V_{1,3}^1, \ldots, V_{1,3}^{\ell-1+\lfloor (q+1)/2 \rfloor}\big\}.\end{aligned}$$

Observe that $|\mathcal{P}'| = r$. Let $\sigma_{\text{BRIDGE}} : V(C) \to \mathcal{P}'$ be a bijective mapping. We first describe E_{BRIDGE} based on σ_{BRIDGE} and then specify σ_{BRIDGE}. The following describes the edges we add to E_{BRIDGE} for each edge $e = \{u, v\} \in E(H)$ where $u \in V(C)$ and $v \notin V(C)$. Let $z_v \in V_{\text{FIXED-SET}}$ be the vertex corresponding to v. We add an edge between z_v and each vertex in $\sigma_{\text{BRIDGE}}(u)$. We describe σ_{BRIDGE} next.

We set σ_{BRIDGE} to map $V(C)$ to an r-cycle in G_{CORE} that is a $\mathcal{P}'$-match of C (recall Definition 5.2). Recall that $C = a_{k-r+1}, a_{k-r+2}, \ldots, a_k, a_{k-r+1}$. We break this cycle into three parts of length ℓ, $\ell + \lfloor q/2 \rfloor$, and $\ell + \lfloor (q+1)/2 \rfloor$, respectively, starting from a_{k-r+1}. We set $\sigma_{\text{BRIDGE}}(a_{k-r+1})$ to V_1, $\sigma_{\text{BRIDGE}}(a_{k-r+1+\ell})$ to V_2, and $\sigma_{\text{BRIDGE}}(a_{k-r+1+2\ell+\lfloor q/2 \rfloor})$ to V_3. In order for σ_{BRIDGE} to map C to a $\mathcal{P}'$-match of C, we set σ_{BRIDGE} to map vertices of C between a_{k-r+1} and $a_{k-r+1+\ell}$ to vertices of the paths between V_1 and V_2, which are vertices in $V_{1,2}$. Similarly, σ_{BRIDGE} maps vertices of C between

$a_{k-r+1+\ell}$ and $a_{k-r+1+2\ell+\lfloor q/2 \rfloor}$ to $V_{2,3}$, and vertices of C between $a_{k-r+1+2\ell+\lfloor q/2 \rfloor}$ and a_{k-r+1} to $V_{1,3}$. Formally,

$$\begin{aligned}
\sigma_{\text{BRIDGE}}(a_{k-r+1}) &= V_1, \\
\sigma_{\text{BRIDGE}}(a_{k-r+1+i}) &= V^i_{1,2}, \text{ for } i \in \{1, \ldots, \ell-1\}, \\
\sigma_{\text{BRIDGE}}(a_{k-r+1+\ell}) &= V_2, \\
\sigma_{\text{BRIDGE}}(a_{k-r+1+\ell+i}) &= V^i_{2,3}, \text{ for } i \in \{1, \ldots, \ell-1+\lfloor q/2 \rfloor\}, \\
\sigma_{\text{BRIDGE}}(a_{k-r+1+2\ell+\lfloor q/2 \rfloor}) &= V_3, \\
\text{and } \sigma_{\text{BRIDGE}}(a_{k-r+1+2\ell+\lfloor q/2 \rfloor+i}) &= V^i_{2,3}, \text{ for } i \in \{1, \ldots, \ell-1+\lfloor (q+1)/2 \rfloor\}.
\end{aligned}$$

This completes the description of E_{BRIDGE} and hence G_H. Before presenting the details of the reduction, we first show that G_H has bounded degeneracy and $O(m)$ edges.

The following lemma shows that in order to prove a graph G is t-degenerate, we only need to exhibit an ordering $\prec$ of $V(G)$ such that each vertex of G has t or fewer neighbors that come later in the ordering $\prec$. Given a graph G and an ordering $\prec$ of $V(G)$, the DAG $G^{\rightarrow}_{\prec}$ is obtained by orienting the edges of G with respect to $\prec$.

Lemma 5.4. *[Szekeres-Wilf [SW68]] Given a graph G, $\kappa(G) \leqslant t$ if there exists an ordering $\prec$ of $V(G)$ such that the out-degree of each vertex in $G^{\rightarrow}_{\prec}$ is at most t.*

Next, we show that G_H has bounded degeneracy using Lemma 5.4.

Lemma 5.5. $\kappa(G_H) \leqslant k - r + 2$.

Proof. We present a vertex ordering $\prec$ for G_H such that for each vertex $v \in V(G_H)$, the out-degree of v is at most $k - r + 2$ in $G_H{}^{\rightarrow}_{\prec}$. Let $\prec$ be an ordering of $V(G_H)$ such that $V_{\text{AUXILIARY-SET}} \prec V_{\text{CORE-SET}} \prec V_{\text{FIXED-SET}}$, and ordering within each set is arbitrary. Each vertex in $V_{\text{AUXILIARY-SET}}$ is connected to exactly two other vertices in $V(G_{\text{CORE}})$ and at most to all $k - r$ vertices in $V_{\text{FIXED-SET}}$. So the out-degree of each vertex in $V_{\text{AUXILIARY-SET}}$ in $G_H{}^{\rightarrow}_{\prec}$ is at most $k - r + 2$. Since there are no edges inside $V_{\text{CORE-SET}}$, the only out-edges from vertices inside $V_{\text{CORE-SET}}$ is to vertices in $V_{\text{FIXED-SET}}$. Further, the only out-edges from vertinces in $V_{\text{FIXED-SET}}$ are to other vertices in $V_{\text{FIXED-SET}}$. Thus the out-degree of each vertex $v \in V(G_H)$ in $G_H{}^{\rightarrow}_{\prec}$ is at most $k - r + 2$. ☐

Observe that G_H has at most $\kappa(G_H) \cdot |V(G_H)|$ edges. By Lemma 5.5, $\kappa(G_H) < k$, and $|V(G_H)| < 6m\ell + 3n + k$ by construction of G_H. Thus, G_H has $O(m)$ edges.

Details of the Reduction. We define a partition of $V(G_H)$ by adding each vertex in $V_{\text{FIXED-SET}}$ as a set by itself to $\mathcal{P}'$. Formally,

$$\begin{aligned}
\mathcal{P} = \big\{ &\{z_1\}, \{z_2\}, \ldots, \{z_{k-r}\}, \\
&V_1, V_2, V_3, \\
&V^1_{1,2}, \ldots, V^{\ell-1}_{1,2}, \\
&V^1_{2,3}, \ldots, V^{\ell-1+\lfloor q/2 \rfloor}_{2,3}, \\
&V^1_{1,3}, \ldots, V^{\ell-1+\lfloor (q+1)/2 \rfloor}_{1,3} \big\}.
\end{aligned}$$

Observe that $|\mathcal{P}| = k$. Also, since G_H has bounded degeneracy, each subgraph of G_H has bounded degeneracy too. Thus, by Lemma 5.3 we can count $\mathcal{P}$-matches of H in G_H in linear time if there exists an algorithm ALG for $\#\text{Hom}_H$ problem that runs in time $O(mf(\kappa, k))$ for a positive function

f. In Lemma 5.6, we prove that there is a bijection between $\mathcal{P}$-matches of H in G_H and $\mathcal{P}'$-matches of C in G_{CORE}. Further, in Lemma 5.7, we prove that the number of triangles in G is a simple linear function of the number of $\mathcal{P}'$-matches of C in G_{CORE}. So, by counting $\mathcal{P}$-matches of H in G_H, we can obtain the number of triangles in G.

Lemma 5.6. *There exists a bijection between the set of $\mathcal{P}$-matches of H in G_H and the set of $\mathcal{P}'$-matches of C in G_{CORE}.*

Proof. Let H' be a $\mathcal{P}$-match of H in G_H. Observe that by construction of G_H, the only edges of G_H inside $V(H')$ are edges of H'. Therefore, H' is actually an induced match of H in G_H. By construction of G_H, specifically E_{BRIDGE}, the number of edges between $V_{\text{FIXED-SET}}$ and $V(H') \setminus V_{\text{FIXED-SET}}$ is equal to the number of edges between $H_{C\text{-EXCLUDED}}$ and C. As G_{FIXED} is a copy of $H_{C\text{-EXCLUDED}}$, there are exactly $|E(H_{C\text{-EXCLUDED}})|$ edges inside the set of vertices $V_{\text{FIXED-SET}}$. Thus, H' has exactly $|E(C)| = r$ edges inside G_{CORE}. We describe these edges next.

Let w_i, x_j, and y_t be the vertices of H' in V_1, V_2, and V_3, respectively. Inside G_{CORE}, w_i could only be connected to the two vertices of H' in $V^1_{1,2}$ and $V^1_{1,3}$. Furthermore, x_j could only be adjacent to the two vertices of H' in $V^{\ell-1}_{1,2}$ and $V^1_{2,3}$. And finally, y_t could only be neighbors of the two vertices of H' in $V^{\ell-1+\lfloor q/2 \rfloor}_{2,3}$ and $V^{\ell-1+\lfloor (q+1)/2 \rfloor}_{1,3}$. In addition, each vertex in $V_{\text{AUXILIARY-SET}}$ has at most two neighbors inside G_{CORE}, and the same holds in H'. Inside G_{CORE}, H' has exactly r edges, so each vertex is connected (only) to their two possible neighbors specified above. Hence, there exist an ℓ-path between w_i and x_j, an $\ell + \lfloor q/2 \rfloor$-path between x_j and y_t, and an $\ell + \lfloor (q+1)/2 \rfloor$-path between w_i and y_t. Thus, $H' - V_{\text{FIXED-SET}}$ is an r-cycle inside G_{CORE} that includes exactly one vertex in each part of P', and hence is a $\mathcal{P}'$-match of C. It is easy to see that this $\mathcal{P}'$-match of C is actually an induced match. Next, we show the other direction.

Let C' be a $\mathcal{P}'$-match of C in G_{CORE}. It is easy to see that by construction of G_{CORE}, C' is an induced match. By construction of G_H, $G_H[V(C') \cup V_{\text{FIXED-SET}}]$ is an induced match of H. Therefore, C' corresponds to exactly one $\mathcal{P}$-match of H in G_H. ☐

Lemma 5.7. *Let $\mathcal{P}'\text{-match}(C, G_{\text{CORE}})$ denote the set of $\mathcal{P}'$-matches of C in G_{CORE}. The number of triangles in G is equal to $|\mathcal{P}'\text{-match}(C, G_{\text{CORE}})|/6$.*

Proof. Consider a cycle $C' \in \mathcal{P}'\text{-match}(C, G_{\text{CORE}})$. Let w_i, x_j, and y_t be the the only vertices of C' in V_1, V_2, and V_3, respectively. There should be a path between w_i and x_j in C' that does not include y_t, so other than w_i and x_j, it only includes vertices in $V_{\text{AUXILIARY-SET}}$. The only possible such path in G_{CORE} is an ℓ-path between w_i and x_j. Therefore, this ℓ-path exists, and as a result $(u_i, u_j) \in E(G)$. Similarly, C' includes a path between x_j and y_t that only contain vertices in V_{aux} other than x_j and y_t. Therefore, there exists an $\ell + \lfloor q/2 \rfloor$-path between x_j and y_t in G_{CORE}, and hence $(u_j, u_t) \in E(G)$. Finally, a path between w_i and y_t that other than its endpoints, only includes vertices in $V_{\text{AUXILIARY-SET}}$, should be a part of C'. So, there exists an $\ell + \lfloor (q+1)/2 \rfloor$-path between w_i and y_t in G_{CORE}. As a result, $(u_i, u_t) \in E(G)$. Thus, C' corresponds only to the triangle u_i, u_j, u_t in G. Observe that, C' could be specified by its vertices in V_1, V_2, and V_3. Next, we prove the other direction; exactly 6 $\mathcal{P}'$-matches of C in G_{CORE} correspond to each triangle in G.

Consider a triangle T with the vertex set $\{u_i, u_j, u_t\}$ in G and a $\mathcal{P}'$-match C' of C in G_{CORE} that corresponds to T. There are six different bijective mappings from $\{u_i, u_j, u_t\}$ to $\{V_1, V_2, V_3\}$. As we showed above, C' could be specified by its vertices in V_1, V_2, and V_3. So, given a bijective mapping $\sigma_{\text{TRIANGLE}} : \{u_i, u_j, u_t\} \to \{V_1, V_2, V_3\}$, the three vertices $\sigma_{\text{TRIANGLE}}(u_i)$, $\sigma_{\text{TRIANGLE}}(u_j)$, and $\sigma_{\text{TRIANGLE}}(u_t)$ specify C'. Thus, there are exactly 6 $\mathcal{P}'$-matches of C in G_{CORE} that correspond to T. As a result, the number of triangles in G is $|\mathcal{P}'\text{-match}(C, G_{\text{CORE}})|/6$. ☐

Lemma 5.6 and Lemma 5.7 together show that we can obtain the number of triangles in G from the number of $\mathcal{P}$-matches of H in G_H, in constant time. In conclusion, we have proved that if there exists an algorithm ALG for the $\#\text{Hom}_H$ problem that runs in time $O(mf(\kappa, k))$ for a positive function f, then there exists an $O(m)$ algorithm for the TRI-CNT problem. Assuming the TRIANGLE DETECTION CONJECTURE, the problem of TRI-CNT has the worst case time complexity of $\omega(m)$ for an input graph with m edges. Thus, the $O(m)$ Turing reduction from the TRI-CNT problem to $\#\text{Hom}_H$ problem we presented proves Theorem 5.1. □

Observation 5.8. *In the proof of Theorem 5.1, we count* $\mathcal{P}$-homomorphisms *(defined in Definition 5.2) from* H *to* G_H *using the algorithm* ALG. *Since* $|\mathcal{P}| = |V(H)|$, *each* $\mathcal{P}$-homomorphism *from* H *to* G_H *is an embedding of* H *in* G_H. *Thus, we can apply the same argument of Lemma 5.3 assuming there exists an algorithm for counting subgraphs, that has the same running time of* ALG. *Therefore, using the same argument as that of the proof of Theorem 5.1, we can prove the exact same statement of Theorem 5.1 for the* $\#\text{Sub}_H$ *problem.*

6 Conclusion

In this paper, we study the problem of counting homomorphisms of a fixed pattern H in a graph G with bounded degeneracy. We provided a clean characterization of the patterns H for which near-linear time algorithms are possible — if and only if the largest induced cycle in H has length at most 5 (assuming standard fine-grained complexity conjectures). We conclude this exposition with two natural research directions.

While we discover a clean dichotomy for the homomorphism counting problem, the landscape for the subgraph counting problem is not as clear. Our hardness result (Theorem 5.1) holds for the subgraph counting version — if a pattern H has LICL $\geqslant 6$, then there does not exists any near-linear time (randomized) algorithm for finding the subgraph count of H (see Observation observation 5.8). However, the "only if" direction does not follow. It would be interesting to find a tight characterization for the subgraph counting problem.

Both our current work and a previous work [BPS20] attempt at understanding what kind of patterns can be counted in near-linear time in sparse graphs. It would be interesting to explore beyond linear time algorithms. Specifically, we pose the following question: Can we characterize patterns that are countable in quadratic time?

References

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. 31st ACM Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

[AKK18] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, 2018.

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

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

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

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

[BC17] Suman K Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. 34th International Symposium on Theoretical Aspects of Computer Science*, 2017.

[BCG20] Suman K Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *Proc. 47th International Colloquium on Automata, Languages and Programming*, 2020.

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

[BKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

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

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

[BS20] Suman K Bera and C Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 457–467, 2020.

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

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

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

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

[Coh09] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009.

[DG00] Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.

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

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

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

[ELRS17] Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

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

[ERS18] Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proc. 50th Annual ACM Symposium on the Theory of Computing*, pages 722–734, 2018.

[ERS20] Talya Eden, Dana Ron, and C Seshadhri. Faster sublinear approximations of k-cliques for low arboricity graphs. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, 2020.

[FG04] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[GG06] 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.

[GLS20] Lior Gishboliner, Yevgeny Levanzov, and Asaf Shapira. Counting subgraphs in degenerate graphs, 2020.

[Gro07] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM (JACM)*, 54(1):1–24, 2007.

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

[HN90] Pavol Hell and Jaroslav Nešetřil. On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.

[IPZ98] 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.

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[JS17] Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proc. 26th Proceedings, International World Wide Web Conference (WWW)*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017.

[JSP13] Madhav Jha, C Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 589–597, 2013.

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

[KMSS12] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Proc. 39th International Colloquium on Automata, Languages and Programming*, pages 598–609, 2012.

[KPP+14] Tamara G Kolda, Ali Pinar, Todd Plantenga, C Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM Journal on Scientific Computing*, 36(5):S48–S77, 2014.

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

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

[MB83] David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[MMPS11] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In *Proc. 19th Annual European Symposium on Algorithms*, pages 677–688, 2011.

[MVV16] Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016.

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

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

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

[PTTW13] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.

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

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

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

[RW20] 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.

[SERF18] 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.

[SV11] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011.

[SW68] George Szekeres and Herbert S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.