

Counting Patterns in Degenerate Graphs in Constant Space

Balagopal Komarath

Department of Computer Science
IIT Gandhinagar, India
bkomarath@iitgn.ac.in

Anant Kumar

Department of Computer Science
IIT Gandhinagar, India
kumar_anant@iitgn.ac.in

Akash Pareek

Department of Computer Science
Université Libre de Bruxelles, Belgium
akash.pareek@ulb.be

Abstract

For an arbitrary, fixed graph (pattern graph), we study the algorithmic complexity of counting homomorphisms, subgraph isomorphisms, and induced subgraph isomorphisms from the pattern graph to n -vertex, d -degenerate graphs as input. Recent work by Bressan (Algorithmica, 2021) has shown that this problem has efficient dynamic programming algorithms using a graph parameter called DAG treewidth. Bressan used DAG treewidth to design a fast algorithm for counting homomorphisms, subgraph isomorphisms, and induced subgraph isomorphisms that use polynomial space. Bera, Gishboliner, Levanzov, Seshadhri, and Shapira (SODA, 2021) provided a characterization of graphs with DAG treewidth one.

In this paper, we introduce a new graph parameter called DAG treedepth and show that it yields efficient divide and conquer algorithms that use only constant space (in the unit-cost RAM model). Specifically, we show:

- An algorithm for counting subgraphs isomorphic to sparse pattern graphs using only constant space.
- We derive an induced minor-based characterization for graphs of DAG treedepth up to two.
- For pattern graphs upto nine vertices, the induced subgraphs can be counted in $O(n^3)$ time using constant space.
- An algorithm for counting induced subgraphs that matches the running time given by Bressan but only uses constant space.

Apart from the DAG treedepth result, we also focus on DAG treewidth. For DAG treewidth, we show that we can count homomorphisms, subgraph isomorphisms, and induced subgraph isomorphisms faster than Bressan's algorithm (2021). We further show that for all pattern graphs up to 11 vertices, we can count induced subgraphs in quadratic time.

Beyond these algorithmic results, we also analyze DAG treedepth structurally and compare it with treedepth. We also show that DAG treedepth and DAG treewidth are upper- and lower-bounded by treedepth and treewidth of graphs from an associated family of graphs.

1 Introduction

For simple, undirected graphs G (host) and H (pattern), we consider the problem of counting the number of occurrences of H in G under the following three fundamental notions of containment.

1. The number of subgraphs of G isomorphic to H . That is, the cardinality of $\{(V', E') \mid (V', E') \cong G, V' \subseteq V(G), E' \subseteq E(G)\}$.
2. The number of induced subgraphs of G isomorphic to H . That is, the cardinality of $\{G[S] \mid S \subseteq V(G)\}$, where $G[S] = (S, \{\{u, v\} \in E(G) \mid u, v \in S\})$ is the subgraph of G induced by vertices in S .
3. The number of homomorphisms from H to G . That is, the cardinality of $\{\phi : V(H) \mapsto V(G) \mid \{u, v\} \in E(H) \text{ if and only if } \{\phi(u), \phi(v)\} \in E(G)\}$.

Counting subgraphs and homomorphisms is at the heart of many problems in combinatorics, database theory, and network analysis [FG04, Lov12, ANRD15, CDM17b, DK25, KNOZ25]. These problems capture fundamental questions such as detecting specific patterns in large networks, evaluating conjunctive queries. When the host graph G is unrestricted, the complexity of such counting tasks is well understood, as it is closely tied to the structural measures of the pattern graph H . In particular, the running time for counting subgraphs is governed by the *treewidth* of H in the setting of unrestricted space, and by the *treedepth* of H when constant space is imposed [DST02, KKMS23].

However, in many real-world settings, host graphs are not arbitrary but exhibit additional structure such as sparsity or bounded degeneracy [PS20, JSP15]. The class of *d-degenerate graphs* provides a natural way to model such sparse networks. A graph is said to be *d-degenerate* if every subgraph contains a vertex of degree at most d . This family includes many important graph classes, such as planar graphs, graphs excluding a fixed minor and, preferential attachment graphs [Ses23]. Studying counting problems on degenerate graphs can therefore lead to more refined algorithmic bounds that better reflect the structure of real data. In this direction, Bressan [B21] introduced a new structural parameter of the pattern graph, called the *DAG treewidth*, and demonstrated that when the host graph G is *d-degenerate*, counting homomorphisms can be performed significantly faster for patterns of small DAG treewidth. More precisely:

Theorem 1 *If the DAG treewidth of a k -vertex graph H is w , then we can count the number of homomorphisms from H to any n -vertex, d -degenerate G in time $O(f(k, d)n^w)$.*

Bressan’s algorithm for counting homomorphisms is based on a dynamic programming (DP) approach, which is similar to treewidth-based DP algorithms. By analyzing DAG treewidth structurally, Bressan [B21] further showed that, for any k -vertex pattern, we can count the number of induced subgraphs in time $O(f(k, d)n^{k/4})$. Bera, Pashanasangi, and Seshadhri [BPS21] used DAG-treewidth to characterized the set of patterns countable in linear-time by deriving induced minor obstructions for graphs of DAG treewidth 1. They showed that if the largest induced cycle in H has length at most 5, then there is a near-linear time algorithm for counting H -homomorphisms in bounded degeneracy graphs. More works on studying (near) linear-time algorithms for bounded degeneracy graphs includes [BR22, BGLS22, PPS22, PPS25].

Despite this progress, the study of quadratic-time algorithms for bounded-degeneracy graphs remains relatively unexplored. One reason could be that the structural understanding of DAG treewidth quickly becomes intractable. As we show later in the paper, even deciding whether a DAG H has $\text{dtw}(H) \leq 2$ is NP-complete. Formally,

Theorem 2 *Let H be a DAG, then deciding whether $\text{dtw}(H) \leq 2$ is NP-complete.*

This hardness result suggests that DAG treewidth may not be the right parameter for capturing tractable classes beyond linear time in bounded-degeneracy graphs. Therefore, a natural question is whether or not there is an analogue to *treedepth* for counting homomorphisms in d -degenerate graphs. To this end, we aim to design a divide-and-conquer algorithm that operates in constant space. We consider the pattern graph and degeneracy of host graph as fixed. Therefore, if the space only depends on the pattern graph and degeneracy, we regard it as constant-space. To achieve this, we define a new parameter called *DAG treedepth* (dtd), analogous to *treedepth*. Unlike the DAG treewidth parameter, deciding whether a DAG H has $dtd(H) \leq k$ for some constant k is in XP. Informally, we show the following.

Theorem 3 *Let k be a constant. Given a DAG H , it can be verified in $O(n^{f(k)})$ time whether $dtd(H) \leq k$ or not, where $f(k)$ is some computable function.*

1.1 Our Results We begin by introducing a new structural parameter, called the *DAG treedepth* (see [Definition 15](#)), which serves as a natural analogue of *treedepth*. The DAG treedepth parameter enables the design of efficient, constant-space, divide-and-conquer algorithms for counting homomorphisms into d -degenerate host graphs. In particular, we establish the following result:

Theorem 4 *Let d be some constant. If DAG treedepth of a k -vertex graph H is t , then there is an $O(n^t)$ -time, constant space algorithm for counting homomorphisms from H to an n -vertex, d -degenerate host graph G given as input.*

Leveraging the notion of DAG treedepth, we design counting algorithms for induced subgraphs in d -degenerate graphs whose running times match the bounds of Bressan [B21], obtained using DAG treewidth. A key advantage of our framework is its space efficiency, our algorithms operate in constant space, whereas Bressan’s dynamic programming formulation requires polynomial space.

Theorem 5 *Let H be a k -vertex graph. Then, we can count the number of induced subgraphs of a given n -vertex, d -degenerate graph G that is isomorphic to H in $O(n^{k/4+O(1)})$ -time and constant space for some function f .*

We further show that for patterns with few edges, we can count subgraphs even faster while using only constant space.

Theorem 6 *Let H be a k -vertex, ℓ -edge graph. Then, we can count the number of subgraphs of a given n -vertex graph G that is isomorphic to H in $O(n^{\ell/5+O(1)})$ -time and constant space for some function f .*

Next, following the characterization of patterns admitting linear-time and linear-space algorithms by Bera, Pashanasangi, and Seshadhri [BPS19], we extend this framework to derive an induced-minor-based characterization (see [Definition 7](#)). We characterize patterns upto cubic-time, constant-space algorithms. Notably, such a characterization is not attainable using DAG treewidth, even for the quadratic-time regime ([Theorem 2](#)).

Theorem 7 *A graph H has DAG treedepth at most two if and only if it is $\{C_6, P_7, H_1, H_2\}$ -induced-minor-free (See [Figure 2a](#) and [Figure 2b](#)).*

We next show a number of applications of DAG treedepth. We assume the following conjecture.

Conjecture 1 *Let d be a constant. For any constant $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time constant-space algorithm for counting $\text{hom}(P_4, H)$, where the input graph H is a d -degenerate graph.*

Using the above conjecture we show that

Theorem 8 *Patterns that are countable as subgraphs in linear time and constant space are precisely the graphs with $\text{dtd} = 1$ and the star graphs.*

Next, we assume the following conjecture ¹

Conjecture 2 *For any constant $\varepsilon > 0$, there is no $O(n^{3-\varepsilon})$ -time and constant-space algorithm for counting C_3 (triangles) in an arbitrary input graph G .*

Again, using the above conjecture we show that

Theorem 9 *There is no $O(n^{3-\varepsilon})$ -time and constant-space algorithm for counting graphs with $\text{dtd} = 3$ in a bounded-degeneracy graph G with degeneracy equal to two.*

Note that C_6 is one of the graphs with $\text{dtd} = 3$. Further, for upper bounds, we show that

Theorem 10 *Let d be a constant. For all patterns with at most nine vertices, we can count the number of occurrences as induced subgraphs in $O(n^3)$ time and constant space for an n -vertex d -degenerate graph given as input.*

We also show that the nine in the above theorem is tight by showing a conditional lower bound for a ten-vertex pattern.

We further show that the DAG treedepth of a graph is related to the treedepth of an associated family of graphs $\mathcal{G}$ (See [Definition 14](#) for this family). More specifically,

Theorem 11 *Let $\vec{H}$ be a DAG with DAG treedepth d . Then, there are graphs H_1 and H_2 in $\mathcal{G}$ such that $\text{td}(H_1) \leq \text{dtd}(\vec{H}) \leq \text{td}(H_2)$.*

Shifting our focus to DAG treewidth, we show an analogous result for treewidth and DAG treewidth:

Theorem 12 *Let $\vec{H}$ be a DAG with DAG treewidth w . Then, there are graphs H_1 and H_2 in $\mathcal{G}$ such that $(\text{tw}(H_1) + 1)/2 \leq \text{dtw}(\vec{H}) \leq \text{tw}(H_2) + 1$.*

Next, we improve the exponent in Bressan's algorithm to count any k -vertex induced subgraph in d -degenerate graphs from $k/4$ to $k/5$ using polynomial space.

Theorem 13 *Let H be a k -vertex graph and let d be a constant. Then there is an $O(n^{k/5+O(1)})$ -time and polynomial space algorithm to count the number of induced subgraphs of a given n -vertex, d -degenerate graph G that are isomorphic to H .*

In [\[BPS19\]](#) the authors proved that all patterns on at most five vertices can be counted in linear time and recently in [\[PPS24\]](#), the authors extended it to show that all patterns on at most nine vertices can be counted in sub-quadratic time. For DAG treewidth we further extend this result by showing the following.

Theorem 14 *For any pattern graph H , if $|V(H)| \leq 11$, then $\text{dtw}(\vec{H}) \leq 2$.*

In a recent work [\[PPS24\]](#), the author conjectured the following:

¹Note that we know subcubic algorithms for triangle counting using fast matrix multiplication. But they are not constant-space

Conjecture 3 (Conjecture 1.8 of [PPS24]) *There is no $o(m^2)$ algorithm for computing $\text{sub}(\mathcal{H}_\Delta, G)$.*

In this work [PPS24], the author defined a hypergraph $\mathcal{H}_\Delta$, which can equivalently be viewed as a DAG. Specifically, the vertices of the hypergraph correspond to the non-source vertices of the DAG, and the hyperedges represent the set of source vertices. Each hyperedge consists of the non-source vertices reachable from the corresponding source vertex. This construction yields a ten-vertex DAG, which can be interpreted as one-subdivision of K_4 .

Following a similar reduction as in [BPS21], the problem of computing $\text{sub}(\mathcal{H}_\Delta, G)$ reduces to counting K_4 subgraphs in an arbitrary graph. Since the matched treewidth of K_4 is 4, it follows that there is no $o(n^2)$ combinatorial algorithm for this task [KKMS23]. Hence, we provide an affirmative answer to the conjecture, establishing that there is no $o(n^2)$ algorithm for computing $\text{sub}(\mathcal{H}_\Delta, G)$.

We also establish several additional bounds relating DAG treedepth and DAG treewidth, which may be of independent interest.

1.2 Related work Subgraph counting is a fundamental problem in graph algorithms, with its complexity and tractability heavily influenced by the structural properties of the host graph. In *nowhere dense* and *bounded degeneracy* graphs, several works have established exact and approximate algorithms alongside complexity dichotomies. A recent dichotomy hierarchy precisely characterizes linear-time subgraph counting in such graphs [PPS25]. In the dense graph regime, Bressan et al. [BPS21] classified pattern counting complexity using DAG-treewidth, providing a sharp dichotomy for somewhere dense graph classes [BGM23], while related work has addressed the complexity of counting homomorphic cycles in degenerate graphs [GLY23], and pattern counting in directed graphs parameterized by outdegree [BLR23, BR22]. Recent studies have also considered counting patterns when the host graph is sparse [KKMS23]. For approximate counting, several works have proposed efficient approximation algorithms for subgraph counting problems, including a general approximation framework for sparse graphs [LPSXZ25], and fast approximation algorithms specifically for triangle counting in streaming and distributed settings [BS20]. Beyond graphs, the study of subhypergraph counting has recently gained traction. In [BDDRW25], the authors extended the homomorphism basis framework to hypergraphs, while [PPS25b] characterized the class of pattern hypergraphs H that can be counted in near-linear time when the host graph G is a hypergraph.

1.3 Paper Organization We begin with preliminaries, introducing the necessary definitions, notations, and background. In Section 3, we define DAG treedepth, followed by its structural properties in Section 3.1. We then identify obstructions for DAG treedepth 0, 1, and 2 in Section 3.2, provide application of DAG treedepth in Section 3.3 and present constant-space algorithms for computing $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$ in Section 3.4.

In Section 4, we shift focus to DAG treewidth. We present key properties of DAG treewidth in Section 4.1, and use these to design faster algorithms for counting $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$ along with some other additional results in Section 4.2.

2 Preliminaries

In this section, we present several definitions that will be used throughout the paper. All graphs considered are simple, and may be either directed or undirected. For standard terminology in graph theory, we refer the reader to the textbook by Douglas West [We01]. We use the following common notations for some well-known graphs: e or (uv) for edges in graph, P_k for k -vertex paths, C_k for k -cycles, K_k for k -cliques, $K_k - e$ for k -clique with one edge missing, $K_{m,n}$ for complete bipartite graphs. A k -star is a $(k + 1)$ -vertex graph with a vertex u adjacent to vertices $v_1, \dots, v_k$ and no other edges. A

star graph is a k -star for some k . For a graph G and $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by the vertices in S . We begin with the definition of graph homomorphisms.

Definition 1 (Graph Homomorphism) Given two graphs G and H , a graph homomorphism from G to H is a map $\phi : V(G) \rightarrow V(H)$ such that if $(u, v) \in E(G)$, then $(\phi(u), \phi(v)) \in E(H)$.

A homomorphism ϕ is called a partial homomorphism if $\phi(v)$ is not necessarily defined for all $v \in V(G)$ and if $(u, v) \in E(G)$ and $\phi(u)$ and $\phi(v)$ are defined, then $(\phi(u), \phi(v)) \in E(H)$.

We now recall several fundamental graph-theoretic parameters, including treewidth, treedepth, and a few others.

Definition 2 (Tree Decomposition) A tree decomposition of a graph G is a pair $(T, \{B(t)\}_{t \in V(T)})$, where T is a tree and each node $t \in V(T)$ is assigned a subset $B(t) \subseteq V(G)$, called a bag.

A tree decomposition of a graph G has the following properties:

- **Connectivity Property:** For every vertex $v \in V(G)$, the set of nodes $\{t \in V(T) \mid v \in B(t)\}$ forms a connected component in T .
- **Edge Property:** For every edge $e = \{u, v\} \in E(G)$, there exists a node $t \in V(T)$ such that both u and v are in $B(t)$.

Definition 3 (Treewidth) Let the width of a tree decomposition be defined as $\max_{t \in V(T)} |B(t)| - 1$. Then, the treewidth of a graph G , denoted as $\text{tw}(G)$, is the minimum width over all possible tree decompositions of G .

Definition 4 (Elimination Tree) An elimination tree of a connected graph G is a rooted tree (T, r) , where $r \in V(G)$ is the root, and each subtree rooted at a child of r is an elimination tree of a connected component of the graph $G \setminus \{r\}$. For an empty graph (i.e., a graph with no vertices), the elimination tree is defined to be empty.

The depth of an elimination tree (T, r) is the length (i.e., the number of vertices) of the longest path from the root r to any leaf in the tree.

Definition 5 (Treedepth) The treedepth of a graph G , denoted as $\text{td}(G)$, is the minimum depth among all possible elimination trees of G .

Definition 6 (Edge Contraction) An edge contraction is an operation in which an edge (u, v) is removed from the graph and its two vertices u and v are merged into a single vertex uv . All edges incident to u or v are now considered incident to the new vertex uv .

Definition 7 (Minor and Induced Minor) A graph G is said to be a minor of another graph G' if G can be obtained from G' by deleting vertices, deleting edges, and contracting edges. A graph G is an induced minor of a graph G' if it can be obtained by contracting edges and deleting vertices of G' .

Let H be a pattern graph and G be the host graph. We denote the number of homomorphism from H to G by $\text{hom}(H, G)$, number of subgraphs of H in G by $\text{sub}(H, G)$, and number of induced subgraphs of H in G by $\text{ind}(H, G)$.

Definition 8 (Spasm) Let $\mathcal{I}$ denote the set of all independent sets in a graph H . For any $I \in \mathcal{I}$, let $\text{Merge}(H, I)$ be the graph obtained by merging all the vertices in I into a single vertex. Then,

$$\text{Spasm}(H) = \{H\} \cup \bigcup_{I \in \mathcal{I}} \text{Spasm}(\text{Merge}(H, I)).$$

Note that for any pattern graph H , the number of subgraph isomorphisms from H to a host graph G is just the linear combination of all possible graph homomorphisms from $\text{Spasm}(H)$ to G . That is, there exist a constant $\alpha_{H'} \in \mathbb{Q}$ such that:

$$\text{sub}(H, G) = \sum_{H'} \alpha_{H'} \text{hom}(H', G)$$

where H' ranges over all graphs in $\text{Spasm}(G)$. This identity is a key tool in the literature for counting subgraphs (see, for example, [AYZ97, CDM17]).

Definition 9 (*Directed Acyclic Graph (DAG)*) Given an undirected graph $H = (V_H, E_H)$, an acyclic orientation of H is an assignment of a direction to each edge $\{u, v\} \in E_H$, converting it to either $(u \rightarrow v)$ or $(v \rightarrow u)$, such that the resulting directed graph $\vec{H}$ is a directed acyclic graph (DAG) that is, $\vec{H}$ contains no directed cycles.

Let $\vec{H}$ be a directed acyclic graph (DAG), and let $S = S(\vec{H})$ denote the set of *sources* in $\vec{H}$, i.e., the set of vertices with indegree zero. Otherwise, we denote $u \in V(\vec{H} \setminus S)$ as a non-source vertex. For any two vertices $u, v \in V(\vec{H})$, we say that v is *reachable* from u if there exists a directed path from u to v in $\vec{H}$.

For a source vertex $s \in S$, let $R_H(s)$ denote the set of all vertices reachable from s in $\vec{H}$. More generally, for a set of sources $B = \{s_1, s_2, \dots, s_\ell\} \subseteq S$, we define, $R_H(B) = \bigcup_{i=1}^{\ell} R_H(s_i)$. When the underlying graph is clear from the context, we define the set of reachable vertices from s as $R(s)$.

Definition 10 (*Subdivision Vertex*) Given an undirected graph $G = (V, E)$, the subdivision of an edge $\{u, v\} \in E$ is the process of replacing it with two edges $\{u, w\}$ and $\{w, v\}$, where w is a new vertex previously not in V . The vertex w is called a subdivision vertex.

Definition 11 (*Hypergraph*) A hypergraph is a pair $\mathcal{H} = (V, E)$, where V is a finite set of vertices, and $E \subseteq 2^V$ is a non-empty subset of V , called hyperedges.

Definition 12 (*Generalized hypertree decomposition (GHD)*) A GHD of a hypergraph $\mathcal{H} = (V, E)$ is a tuple $\langle T, (B_u)_{u \in N(T)}, (\lambda_u)_{u \in N(T)} \rangle$, such that $T = \langle N(T), E(T) \rangle$ is a rooted tree and the following conditions holds:

1. for each $e \in E$, there is a node $u \in N(T)$ with $e \subseteq B_u$.
2. for each $v \in V$, the set $\{u \in N(T) \mid v \in B_u\}$ is connected in T .
3. for each $u \in N(T)$, λ_u is a function $\lambda_u : E \rightarrow \{0, 1\}$ with $B_u \subseteq B(\lambda_u)$

Definition 13 (*Generalized hypertree width of $\mathcal{H}$ ($\text{ghw}(\mathcal{H})$)*) The width of a GHD is the maximum weight of the function λ_u overall nodes u in T . The $\text{ghw}(\mathcal{H})$ is defined as the minimum width overall GHDs.

In [B21], Bressan described a way to construct a bipartite graph $\text{BIP}(\vec{H})$ from any DAG $\vec{H}$. For completeness, we briefly explain this construction and then describe a simpler version called G_S , which only uses the source vertices of $\vec{H}$. We will use G_S later in several results to get useful bounds.

2.1 Construction of G_S from $\text{BIP}(\vec{H})$: Given a directed acyclic graph $\vec{H}$ with a fixed acyclic orientation, Bressan [B21] constructed a bipartite graph $\text{BIP}(\vec{H})$ as follows:

- The vertex set of $\text{BIP}(\vec{H})$ denoted as $V(\text{BIP}(\vec{H})) = (S, V(H) \setminus S)$, where S is the set of sources in $\vec{H}$.

- An edge $\{u, v\} \in E(\text{BIP}(\vec{H}))$ if and only if $v \in R_H(u)$; that is, v is reachable from the source vertex u in $\vec{H}$.

We now further simplify this structure to construct a new graph G_S from $\text{BIP}(\vec{H})$ by performing a series of edge contractions.

Definition 14 Let $\vec{H}$ be a DAG. For each non-source vertex in $\text{BIP}(\vec{H})$, we contract one of its incident edges, resulting in a total of ℓ contractions, where ℓ is the number of non-source vertices. After all contractions are applied, the non-source vertices are eliminated, and the resulting graph G_S consists solely of the source vertices. We denote by $\mathcal{G}_{\vec{H}}$ (we drop the subscript when the DAG is clear from the context), the set of all such graphs.

We will later demonstrate how to utilize this family of graphs to derive several bounds for DAG treedepth and DAG treewidth. In the following section, we define our new parameter named DAG treedepth and prove all the results related to it.

3 DAG Treedepth

In [B21], Bressan introduced DAG treewidth (dtw), a structural parameter analogous to treewidth, and used it to design efficient algorithms for computing $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$. Motivated by this, we define a related parameter called DAG treedepth (dtd), analogous to treedepth. While DAG treewidth enables the design of fast algorithms but suffers from hardness constraints (Theorem 2), DAG treedepth allows the design of constant-space algorithms and does not exhibit the same intractability barriers.

In this section, we first formally define DAG treedepth and present several of its properties that may be of independent interest. In Section 3.2, we use these properties to identify obstruction sets for DAG treedepth 0, 1, and 2. In Section 3.3, we provide several application of DAG treedepth. In Section 3.4, we leverage these to compute $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$ using constant space, improving the space complexity of the fast algorithms of [B21] from polynomial to constant.

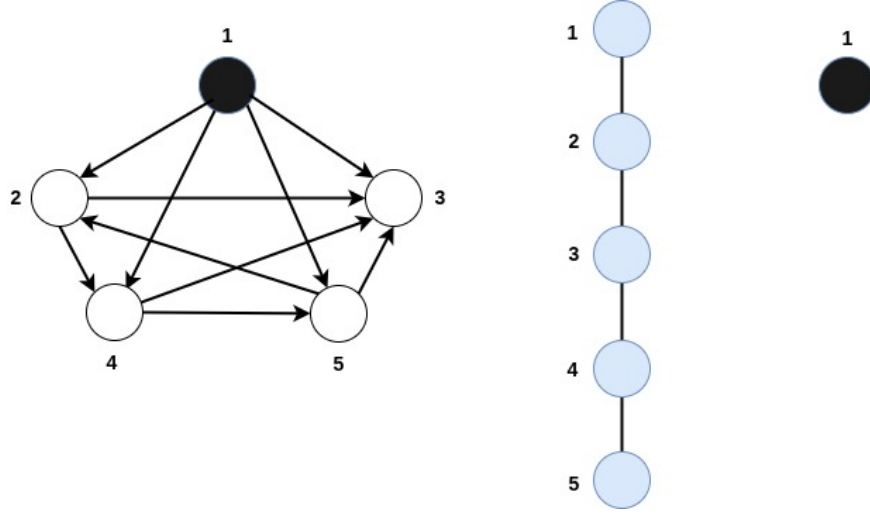
Definition 15 Let H be any DAG. A DAG elimination forest of H is defined as a collection of rooted trees, constructed recursively as follows:

- If H is empty, the forest is empty.
- If H is a single source, the forest is a single-node tree.
- If the underlying undirected graph of H is disconnected, then construct DAG elimination trees for each connected component and their union is a DAG elimination forest for H .
- Otherwise, H has more than one source and its underlying undirected graph is connected. In this case, the forest is a tree. Pick a source s arbitrarily, delete that source and all vertices reachable from it in H . The root of the tree is s and its sub-trees are the trees in the DAG elimination forest for the remaining DAG.

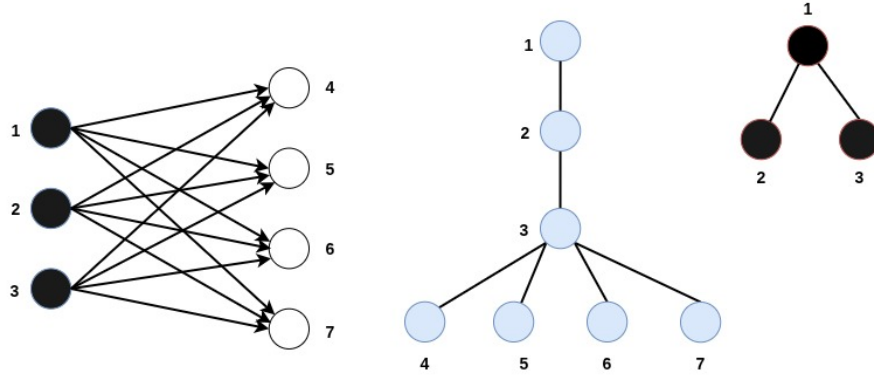
The DAG treedepth of a DAG H is defined as the minimum depth of any elimination forest constructed in the above fashion.

Remark 1 Notice that the definition of DAG treedepth is analogous to the definition of treedepth except that at each step we can delete a source and all non-sources reachable from it instead of only a single node. For DAG treedepth, the nodes of the DAG elimination forest correspond exactly to the sources in the DAG.

Observation 1 *Figure 1 shows two examples where the treedepth is unbounded. For instance, cliques and complete bipartite graphs have unbounded treedepth. However, the DAG treedepth of a clique is 1, and for complete bipartite graphs, it is 2.*



(a) The first figure is K_5 with one source. The second figure is the elimination tree of K_5 . The third figure is the DAG Elimination tree of K_5 .



(b) The first figure is $K_{3,4}$ with 3 sources. The second figure is the elimination tree of $K_{3,4}$. The third figure is the DAG Elimination tree of $K_{3,4}$.

Figure 1: The black nodes are sources and the white nodes are non-sources. In figure (a), the DAG treedepth is 1, but the treedepth is 5. In figure (b), DAG treedepth is 2, but treedepth is 4.

Next, we provide one of our main theorems, which is implied by [Algorithm 1](#).

Theorem 4 *Let d be some constant. If DAG treedepth of a k -vertex graph H is t , then there is an $O(n^t)$ -time, constant space algorithm for counting homomorphisms from H to an n -vertex, d -degenerate host graph G given as input.*

Proof See [Algorithm 1](#). The required count is obtained by $\text{COUNT-HOM}(G, T, r, H, \{\})$, where T is DAG elimination tree of depth t and r is the root of T .

We claim that $\text{COUNT-HOM}(G, T, v, H, \sigma)$ correctly computes the number of homomorphisms that extend the partial homomorphism σ to all the sources in the sub-tree of T rooted at v and all the

non-sources reachable from those sources when σ is a partial homomorphism that maps all ancestors of v in T and all non-sources reachable from those vertices. We prove this using an induction on the height of the node v in T .

In the base case, v is a leaf. In this case, the algorithm is correct because we are simply iterating over all choices for mapping v . Any non-source reachable from v is already mapped in σ or only reachable from v . So once we fix the image of v and all vertices only reachable from it, we only have to check whether this added images to σ are consistent with the other defined vertices in σ . This is constant-time.

If v is an internal node, then observe that after mapping v and all non-sources reachable from it, the subgraphs of H spanned by subtrees of v in T are disjoint. That is, if u and w are two sources in separate subtrees of v in T , then for any non-source x reachable from v and w is also reachable from v or one of its ancestors. So $\sigma(x)$ is defined. We can count extensions of σ to sources in each of the subtrees and new non-sources reachable from them independently and compute the total by multiplying these counts.

In each iteration of the outer loop of [Algorithm 1](#), observe that the loop in line 4 can only have $g(k, d)$ many iterations for some function g as there are at most d outgoing edges from any vertex in H and the length of any simple path from v in H is bounded by k . The number of iterations of loop in line 6 is bounded by k . When a recursive call is made, the depth of v increases by 1. Therefore, the time complexity is $f(k, d)n^t = O(n^t)$ for some function f . Recall that we consider k and d as constants. So we can absorb them into the $O(\cdot)$ notation. Each level of the recursion uses only constant space and the depth of the recursion is at most t . Since we regard t as a constant, the algorithm uses only constant space. $\square$

3.1 Properties of DAG Treedepth: In this section, we establish many important properties of DAG treedepth and treedepth. We start by proving a condition under which two sources u and v in the DAG elimination tree of $\vec{H}$ must lie on the same root-to-leaf path.

Lemma 1 *Let $\vec{H}$ be a directed acyclic graph (DAG) with a fixed orientation. Suppose there are two source vertices u and v such that there exists a vertex $w \in R_H(u) \cap R_H(v)$, and $w \notin R_H(s)$ for any other source $s \neq u, v$. Then, in any DAG elimination tree of $\vec{H}$, the sources u and v must lie on the same root-to-leaf path.*

Proof Assume, for contradiction, that there exists a DAG elimination tree in which u and v do not appear on the same root-to-leaf path. Then, by the definition of an elimination tree, u and v belong to different subtrees, and the sets of vertices discovered by eliminating u and v are disjoint. However, since w is reachable from both u and v and from no other source, the elimination tree would fail to account for the reachability of w , violating the correctness of the elimination tree. Therefore, u and v must lie on the same root-to-leaf path. $\square$

Since DAG treedepth is a new parameter introduced in this work, inspired by the classical notion of treedepth, our goal is to establish a precise relationship between the two. Using the construction $\mathcal{G}$ defined in [Section 2.1](#), we show that for $G_1, G_2 \in \mathcal{G}$, the following holds: $td(G_1) \leq dtd(\vec{H}) \leq td(G_2)$.

Lemma 2 *For a given DAG $\vec{H}$ there exists a $G_S \in \mathcal{G}$ such that $dtd(\vec{H}) \leq td(G_S)$.*

Proof Let T be an elimination tree of the graph G_S . From the construction in [Section 2.1](#), the vertex set of G_S is $V(G_S) = S$, where S is the set of sources in the DAG $\vec{H}$. By definition, every source vertex appears in T .

Algorithm 1 COUNT-HOM(G, T, v, H, σ)

Require: G - The d -degenerate host graph

Require: T - The DAG elimination tree for H

Require: v - A vertex in T (or H)

Require: H - The pattern DAG

Require: σ - A partial homomorphism from H to G

```
1:  $c \leftarrow 0$ 
2: for all  $u \in V(G)$  do
3:    $p \leftarrow 0$ 
4:   for all  $\sigma'$  extending  $\sigma$  to  $\{v\} \cup R(v)$  such that  $\sigma'(v) = u$  do
5:      $p \leftarrow 1$ 
6:     for all children  $w$  of  $v$  in  $T$  do
7:        $p \leftarrow p \times \text{COUNT-HOM}(G, T, w, H, \sigma')$ 
8:     end for
9:   end for
10:   $c \leftarrow c + p$ 
11: end for
12: return  $c$ 
```

To prove the lemma, we need to show that T also satisfies the *reachability intersection property* required for a valid DAG elimination tree. That is, for any pair of source vertices s_i and s_j , if their reachability sets intersect ($R_H(s_i) \cap R_H(s_j) \neq \emptyset$), then either s_i and s_j lie on the same root-to-leaf path in T , or every vertex in $R_H(s_i) \cap R_H(s_j)$ is reachable from some common ancestor source s_k of both s_i and s_j in T .

Formally, we require that $R_H(s_i) \cap R_H(s_j) \subseteq \bigcup_k R_H(s_k)$, where the union is over all source vertices s_k that are common ancestors of s_i and s_j in T . If $R_H(s_i) \cap R_H(s_j) = \emptyset$, there is nothing to prove. If s_i and s_j lie on the same root-to-leaf path in T , the reachability intersection property is also satisfied.

Now consider the case when $R_H(s_i) \cap R_H(s_j) \neq \emptyset$ and s_i, s_j lie on different root-to-leaf paths in T . This can happen only if $\{s_i, s_j\} \notin E(G_S)$. From the construction of G_S , we know that for each non-source vertex $u \in R_H(s_i) \cap R_H(s_j)$, there exists a source s_u such that $u \in R_H(s_u)$, and during the edge contraction process, edges $\{s_i, s_u\}$ and $\{s_j, s_u\}$ were added to G_S . Therefore, s_u is a common ancestor of s_i and s_j in T , and u is in the reachability set of s_u .

Since u was chosen arbitrarily from $R_H(s_i) \cap R_H(s_j)$, the entire intersection is covered by the reachability sets of ancestor sources in T . Hence, T satisfies the reachability intersection condition. Therefore, T is a valid DAG elimination tree of $\vec{H}$, and we conclude that $dtd(\vec{H}) \leq td(G_S)$. $\square$

Lemma 3 For a given DAG $\vec{H}$ there exist a $G_S \in \mathcal{G}$ such that, $dtd(\vec{H}) \geq td(G_S)$.

Proof Let T be a DAG elimination tree of $\vec{H}$ based on a fixed acyclic orientation. Recall from the construction in [Section 2.1](#) that the vertex set of G_S is the set of source vertices of $\vec{H}$. Therefore, the nodes of T correspond to the sources in H . We aim to show that T is also a valid elimination tree for G_S , thereby proving that the treedepth of G_S is at most the DAG treedepth of $\vec{H}$.

Consider any pair of source vertices s_i and s_j :

- If s_i and s_j lie on the same root-to-leaf path in T , then the elimination tree requirement for G_S is trivially satisfied.

- If $R(s_i) \cap R(s_j) = \emptyset$, then s_i and s_j are not adjacent in G_S , and again there is nothing to prove.
- Otherwise, suppose $R(s_i) \cap R(s_j) \neq \emptyset$ and s_i and s_j lie on different root-to-leaf paths in T . Since T is a valid DAG elimination tree, it satisfies the reachability intersection condition: for each non-source vertex $u \in R(s_i) \cap R(s_j)$, there exists a source vertex s_u that is a common ancestor of both s_i and s_j in T such that $u \in R(s_u)$.

By the construction of G_S , for such a vertex u , an edge contraction was performed between u and s_u . Therefore, no edge exists between s_i and s_j in G_S , the shared reachability is captured by their ancestor s_u .

Hence, T is a valid elimination tree for G_S , and the depth of T is at least the treedepth of G_S . Thus, $dtd(\vec{H}) \geq td(G_S)$. $\square$

Remark 2 Note that, unfortunately, dtd is not minor closed like td . One can observe that k_6 has only one source and, therefore, $dtd(K_6) = 1$. However, considering the C_6 subgraph of k_6 , we know that $dtd(C_6) = 3$. Thus, dtd is not even subgraph-closed.

Let I be an induced minor of a graph H , then we show the following theorem.

Theorem 15 If I is an induced minor of H , then $dtd(H) \geq dtd(I)$.

Proof To obtain this result, it suffices to prove the following two claims.

1. If I is an induced subgraph of H then $dtd(H) \geq dtd(I)$.
2. Let H be an undirected graph with $dtd(H) = k$. Let H' be the undirected graph obtained by contracting a single edge in H . Then, $dtd(H') \leq k$.

Proof of Claim 1: Let $dtd(I) = k$. By definition, there exists an acyclic orientation O_1 of I such that $dtd(I) = k$ under this orientation.

We now construct an acyclic orientation of H that extends O_1 and preserves the treedepth. Let $I' \subseteq H$ be an induced subgraph isomorphic to I , with an isomorphism $f : V(I) \rightarrow V(I')$. Fix the orientation of I' in H according to O_1 .

Let $V(H) = V(I') \cup (V(H) \setminus V(I'))$. Partition the remaining vertices based on their distance from I' in H : define layers $D_1, D_2, \dots, D_j$, where

$$D_i = \{v \in V(H) \setminus V(I') \mid \text{dist}_H(v, V(I')) = i\}.$$

Now, define a topological ordering of the vertices of H as:

$$V(I'), D_1, D_2, \dots, D_j.$$

Orient the edges from each D_i to D_{i+1} in the forward direction (i.e., from smaller to larger layers), and similarly orient the edges from $V(I')$ to D_1 forward. For edges within any D_i , choose an arbitrary acyclic orientation.

Note that after this orientation, we have $S(I) = S(\vec{H}) \cap I'$. Also, for any non-source u in I , we have $P_I(u) = P_{\vec{H}}(u)$ i.e., there is no source reaching u other than the source vertices in I .

Now, suppose for contradiction that $dtd(H) < k$ under this orientation. Let T be a valid DAG elimination tree of H with DAG treedepth less than k . Then, the restriction of T to the subgraph I' forms a valid DAG elimination tree of I' , and hence of I (by isomorphism), contradicting the assumption that $dtd(I) = k$. Therefore, $dtd(H) \geq dtd(I)$.

Proof of Claim 2: The DAG treedepth of an undirected graph H is the maximum of the DAG treedepth of DAG $\vec{H}$. Let H' be the graph obtained after contracting an edge $\{u, v\}$. We want to show that $\text{dtd}(H') \leq \text{dtd}(H)$. We pick an arbitrary but fixed acyclic orientation of H' . We copy the same orientation in H . Note that $\{u, v\} \notin E(H')$. So, the orientation is not known. We first give (u, v) orientation to $\{u, v\}$. Note that the orientation of H is acyclic. Let w be the vertex in H' obtained after contracting $\{u, v\}$. Let T be a DAG elimination tree of $\vec{H}$ of width at most k . We will now construct DAG elimination tree T' of $\vec{H}'$.

- **Case 1:** Both u and v are non-source vertices.

In this case, we can assume without loss of generality that $P_H(u) \subseteq P_H(v)$, i.e., all source vertices reaching u also reach v . Note that the set of source vertices remains unchanged after contraction, i.e., $S(\vec{H}') = S(\vec{H})$. We consider the same DAG elimination tree decomposition T of $\vec{H}$. One can see that $P_H(v) = P_{H'}(w)$. So, the source vertices in $P_{H'}(w)$ follow ancestor descendant relation.

- **Case 2:** u is a source vertex.

We consider two subcases depending on whether the new vertex w is a source in $\vec{H}'$ or not.

- **Case A :** w is a source vertex.

This happens only when $P_H(v) = \{u\}$, meaning v was reachable only from u . In this case, $R_H(u) = R_{H'}(w) \setminus \{v\}$. We can construct a DAG elimination tree T' of $\vec{H}'$ by simply replacing u with w . Since no new vertices are introduced and treedepth remain unchanged, $\text{dtd}(\vec{H}') \leq k$.

- **Case B:** w is a non-source vertex.

Here, the contraction removes u from the set of sources, so $S(\vec{H}') = S(\vec{H}) \setminus \{u\}$. v must be reachable by some source vertex s other than u . Moreover,

$$P_{H'}(w) = P_H(v) \setminus \{u\},$$

and for each source $s \in P_{H'}(w)$,

$$R_{H'}(s) = R_H(s) \cup R_H(u).$$

Since $P_H(v)$ follows ancestor descendant relations in T , and since $u \in P_H(v)$, the subtree formed by $P_H(v)$ also includes u . If u appears as an ancestor, then we replace u by $s \in P_{H'}(w)$, which is closest to the root. As these modifications only replace one source with another and do not increase the depth of the tree, the resulting DAG elimination tree T' of $\vec{H}'$ has the same width as T , i.e., $\text{dtd}(\vec{H}') \leq k$.

Finally, after repetitive use of Claim 2, we get an induced subgraph, and then we use Claim 1. Thus, $\text{dtd}(I) \leq \text{dtd}(H)$. □

Next, we show that unlike DAG treewidth, the DAG treedepth parameter is in XP (See [Algorithm 2](#)).

Theorem 16 *Let k be a constant. Given a DAG H , it can be verified in $O(n^{f(k)})$ time whether $\text{dtd}(H) \leq k$ or not, where $f(k)$ is some computable function.*

Proof We can design an XP algorithm to check whether, for a given DAG H , $\text{dtd}(H) \leq k$. The algorithm (Algorithm 2) recursively constructs a DAG elimination tree that satisfies the required properties.

We proceed as follows. Initially, we pick a source vertex s of H , delete s and all non-source vertices reachable from it (denoted by $R(s)$), and obtain the reduced DAG $H' = H - \{s\} - R(s)$. In the elimination tree, s becomes the root.

If H' is connected, we recursively pick a source s_1 from H' , delete s_1 and its reachable vertices, and make s_1 a child of s in the elimination tree.

If H' is disconnected, say H' has connected components $H'_1, H'_2, \dots, H'_r$, we recursively apply the same process to each H'_i . In the elimination tree, the roots of the elimination trees of H'_i are made children of s .

This above process continues until the graph becomes empty. If there exists a sequence of choices of sources that yields an elimination tree of height at most k , then $\text{dtd}(H) \leq k$.

Since at each step we try all possible choices of sources and the depth of recursion is at most k , and there can be at most $O(n)$ sources at any step, the running time is bounded by $O(n^{f(k)})$. Hence, the algorithm runs in XP time with respect to parameter k . $\square$

3.2 DAG Treedepth Obstruction: In this section, we first define a reduced DAG elimination tree and use it to show DAG treedepth obstruction for treedepth 0, 1, and 2.

Definition 16 (*Reduced DAG Elimination Tree*) A DAG elimination tree is called a reduced DAG elimination tree if for every source vertex s , there exists a non-source vertex u such that in the elimination tree, $u \in R(\text{PARENT}(s)) \cap R(T(s))$ and $u \notin R(\text{ancestor of PARENT}(s))$.

One can check that the leaf source vertex has a unique intersection with its parent source in the DAG elimination tree T . We now establish the following lemma.

Lemma 4 Let H be a DAG with $\text{dtd}(H) = d$. Then, there exists a reduced DAG elimination tree of H with maximum depth d .

Proof Suppose that in a DAG elimination tree T , for every source vertex s , there does not exist a non-source vertex u such that $u \in R(\text{PARENT}(s)) \cap R(T(s))$ and $u \notin R(\text{ancestor of PARENT}(s))$. In this case, we can modify the tree by removing the edge between s and $\text{PARENT}(s)$, and instead adding an edge between s and $\text{PARENT}(\text{PARENT}(s))$.

We apply the above modification in a bottom-up fashion. That is, we start with the leaf source vertices and ensure each one is non-conflicting. Then we proceed upward through the tree, level by level. At each step, this process reduces the number of conflicts in the elimination tree. Moreover, the height of the tree does not increase. Therefore, the resulting tree T' is a valid reduced DAG elimination tree with maximum depth d . $\square$

We are now ready to derive the induced minor obstructions for DAG treedepth 0, 1, and 2.

Theorem 17 The induced minor obstructions for connected DAGs with DAG treedepth are as follows:

1. For DAG treedepth 0: the obstruction is K_1 .
2. For DAG treedepth 1: the obstruction is P_3 .
3. For DAG treedepth 2: the obstructions are the graphs C_6, P_7, H_1, H_2 .

Algorithm 2 Checking whether $\text{dtd}(H) \leq k$ for a DAG H

Require: A directed acyclic graph $H = (V, E)$ and an integer $k \geq 1$

Ensure: Returns **true** if $\text{dtd}(H) \leq k$, otherwise **false**

```
1: function CHECKDTD( $H, k$ )
2:   if  $H$  is empty then
3:     return true
4:   end if
5:   if  $k < 1$  then
6:     return false
7:   end if
8:   for each source vertex  $s$  of  $H$  do
9:     Let  $R(s)$  be the set of non-source vertices reachable from  $s$ 
10:    Let  $H' = H - \{s\} - R(s)$ 
11:    if  $H'$  is disconnected then
12:      Let  $\{H'_1, H'_2, \dots, H'_r\}$  be the connected components of  $H'$ 
13:       $valid \leftarrow \text{true}$ 
14:      for each component  $H'_i$  do
15:        if  $\text{CHECKDTD}(H'_i, k - 1)$  is false then
16:           $valid \leftarrow \text{false}$ 
17:          break
18:        end if
19:      end for
20:      if  $valid$  then
21:        return true
22:      end if
23:    else
24:      if  $\text{CHECKDTD}(H', k - 1)$  is true then
25:        return true
26:      end if
27:    end if
28:  end for
29:  return false
30: end function
```

Proof We can easily verify the DAG treedepth of the listed graphs. We now show that these graphs form the complete set of induced minor obstructions for each treedepth level.

Treedepth 0: Any DAG with DAG treedepth one must have at least one source. A single-vertex DAG is trivially an example with one source, and its DAG treedepth is one. In fact, having exactly one source is also a sufficient condition for the DAG treedepth to be one. For example, all DAGs that are cliques (i.e., complete DAGs with a single source) have DAG treedepth one.

Treedepth 1: A DAG with treedepth two must have at least two source vertices. In such a case, there must exist at least one non-source vertex that is reachable from both sources. This implies that the path on three vertices, P_3 , where a single non-source is reachable from two distinct sources, forms a minimal induced minor obstruction for DAG treedepth one.

Treedepth 2: A DAG with treedepth three must have at least three source vertices. Also, there is no source s such that for each pair of sources s_i and s_j , $R(s_i) \cap R(s_j) \subseteq R(s)$. Otherwise we make s as root and all other sources as child and get a DAG elimination tree of depth two.

Let T be its reduced DAG elimination tree, defined over the set of sources S , and let r be the root of T . For every source vertex $s \in S$ that is a leaf of T at depth three, and let s_i be its parent in T . Then there must exist at least one non-source vertex $v \in V \setminus S$ such that: $v \in (R(s) \cap R(s_i)) \setminus R(r)$.

Moreover, for any source vertex $s_1 \in S$ that is a child or grandchild of r , there exist a non-source vertex v that is reachable from both s_1 and r , i.e., $\exists v \in V \setminus S$ such that $v \in R(r) \cap R(s_1)$, otherwise, the graph would not be connected. Otherwise, we can make child of r as the root.

- **Case 1:** The root r has only one child s in the elimination tree T .

If s has only one child s_1 in T , then we know that there exists $u \in R(s_1) \cap R(s)$ and $u \notin R(r)$. Also, there exists $v \in R(r) \cap R(s_1)$ and $v \notin R(s)$ otherwise we can root at s . Also, there exist $w \in R(r) \cap R(s)$ and $w \notin R(s_1)$. Otherwise we can root at s_1 . Thus, the structure induce a C_6 , otherwise, the DAG treedepth would be at most two.

If s has more than one child. Let $\{s_1, s_2, \dots, s_l\}$ be the children of s in T . So, $R(s_i) \cap R(s_j) \subseteq R(s)$. Given T is reduced elimination tree, there exists $u \in R(s) \cap R(s_i)$ and $u \notin R(r)$. Also, there exists $v \in R(r) \cap R(s_i)$ and $v \notin R(s)$, otherwise, we can root at s and get a DAG elimination tree of depth two. Now, there exist w such that either

- $w \in R(s) \cap R(s_j)$ and $w \notin R(s_i)$ or
- $w \in R(s) \cap R(r)$ and $w \notin R(s_i)$

Otherwise, we can root at s_i and get elimination tree of depth two. Considering the case $w \in R(s) \cap R(s_j)$ and $w \notin R(s_i)$, we can check that it forms a structure over seven vertices, such that P_7 is a subgraph. So, the induced minor would be either P_7 or its super graphs such that dtd of supergraph of P_7 is greater than two. We can check that those supergraphs are either H_1 or H_2 . So, the induced minor obstruction are P_7, H_1 and H_2 in this case.

Now, consider the case $w \in R(s) \cap R(r)$ and $w \notin R(s_i)$, we can check that it forms induced C_6 .

- **Case 2:** The root r has more than one child.

In this case, we first handle the case when one child has child in T . So, there exists $u \in R(s) \cap R(s_1)$ and $u \notin R(r)$. Also, there must exist some non-source in intersection set of pairwise source vertices other than s that is not reachable by s . So, there exist $v \in R(s_1) \cap R(r)$ and $v \notin R(s)$. Now, there exist a non-source in pairwise source reachability intersection otherwise we can root at s_1 and get DAG elimination tree of depth two. So, there exist w such that

$w \in R(r) \cap R(s')$ and $w \notin R(s_1)$, where s' is another child of r in T ; or $w \in R(r) \cap R(s)$ and $w \notin R(s_1)$. One can identify P_7 or its super graphs of dtd three as induced minor.

Consider the case when more than one child of r has children in T . Let s and s' be child of r and $\{s_1, s_2, \dots, s_l\}$ are children of s and $\{s'_1, s'_2, \dots, s'_l\}$ are children of s' . Since T is reduced DAG elimination tree, there exist $u \in R(s) \cap R(s_i)$ and $u \notin R(r)$. Also, there exist $v \in R(s') \cap R(s'_i)$ and $v \notin R(r)$. It is possible $u = v$ or $v \in R(s)$. Let $u \neq v$, then we know that there exist $w \in R(s_i) \cap R(r)$ and $w \notin R(s)$. Again, we can check P_7 is subgraph. Now, consider the case $u = v$, then there exist $w' \in R(s') \cap R(r)$ and $w' \notin R(s)$. Again we get P_7 .

Thus, the induced minor obstruction for treedepth two are C_6 , P_7 and super graphs of P_7 such that its dtd is 3. We can check H_1 and H_2 are such graphs. Hence, the induced minor obstruction of tree depth two are C_6, P_7, H_1 and H_2 .

□

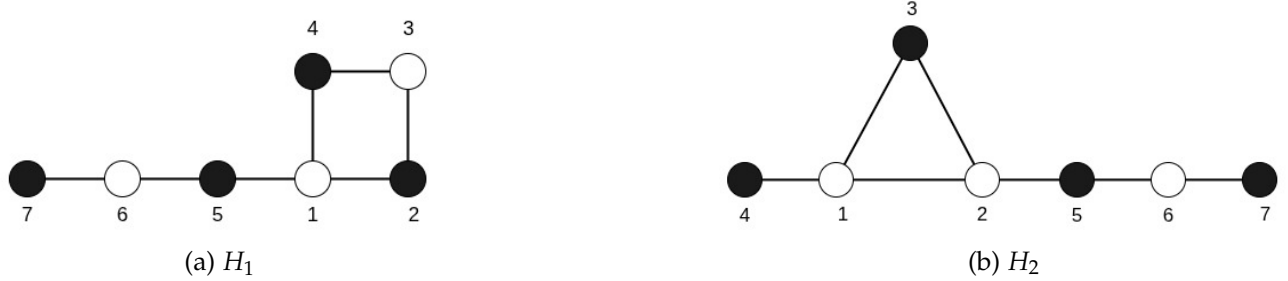


Figure 2: H_1 and H_2 are obstruction for DAG treedepth 2.

In the next two lemmas, we now give relations between the obstructions of treedepth and DAG treedepth. Let H_{sub} be the graph obtained by subdividing each edge of H exactly once. Then we have the following lemma.

Lemma 5 *Let H be a minor obstruction for treedepth $k - 1$. Then $dtd(H_{\text{sub}}) = k$.*

Proof Let T be an elimination tree of H with depth k . Suppose, for contradiction, that there exists a DAG elimination tree T' of H_{sub} with depth at most $k - 1$.

Note that in H_{sub} , the original vertices of H are treated as source vertices, while each subdivision vertex is a non-source. If T' was valid, we could use the same structure as an elimination tree for H with depth at most $k - 1$, contradicting the fact that H is a minor obstruction for treedepth k . More concretely, since H contains an edge $\{u, v\}$, there must exist two vertices u and v not on the same root-to-leaf path in T' . In H_{sub} , this edge is subdivided by a non-source vertex w such that $w \in R(u) \cap R(v)$ and $w \notin R(s)$ for any other source $s \neq u, v$. This violates the reachability intersection condition of DAG elimination trees, so T' is not valid. Hence, no DAG elimination tree of H_{sub} can have depth less than k , therefore $dtd(H_{\text{sub}}) = k$. □

Also, we can list down all supergraphs of H_{sub} whose dtd is k . All these supergraphs will be induced minor obstruction of $k-1$. Now, in next lemma, we show how we can do edge contraction in induced minor obstruction of $dtd = k$ to get minor obstruction of $td = k$.

Lemma 6 *Let H be an induced minor such that $dtd(H) = k$. Then, there exists a sequence of edge contractions between source and non-source vertices such that the resulting graph H' is a minor of treedepth k .*

Proof We observe that after performing edge contractions, the remaining vertices correspond exactly to the set of source vertices in H . Let T be a DAG elimination tree of H with depth k . For each non-source vertex u that is reachable from multiple source vertices, we contract u with the source vertex that is an ancestor of all other sources reaching u in T . We claim that the resulting graph H' , obtained after these edge contractions, has treedepth exactly k .

Suppose, for contradiction, that H' admits an elimination tree T' of depth at most $k - 1$. Now consider using the same tree structure T' as a DAG elimination tree for H , where the contracted vertices are replaced back with their original non-source vertices. Consider any two sibling source vertices s_i and s_j in T' . For each non-source vertex $u \in R(s_i) \cap R(s_j)$, by construction, u was contracted into some source s that is an ancestor of both s_i and s_j in T . Hence, $u \in R(s)$, contradicting the assumption that $\text{dtd}(H) = k$. Therefore, such a T' of depth $k - 1$ cannot exist, and we conclude that the treedepth of H' is exactly k . $\square$

3.3 Application of DAG treedepth Till now, we have established several structural properties of DAG treedepth and an induced-minor-based characterization of DAG treedepth for counting homomorphisms in cubic time using constant space. Leveraging the framework of [CDM17b], we further extend our results to characterize the patterns that can be counted as subgraphs and induced subgraphs.

We have observed that DAGs with $\text{dtd} = 1$ are precisely those having a single source. Moreover, certain orientations, such as the star graph with all leaves as sources, may contain multiple sources but still have linear-time, constant-space algorithm. More generally, if a DAG admits an orientation where either the number of sources or non-sources is one, the counting problem remains tractable in linear time and constant space. However, for the path P_4 , an alternating edge orientation yields two sources and two non-sources, suggesting a potential boundary of tractability. This leads us to the following conjecture:

Conjecture 1 *Let d be a constant. For any constant $\epsilon > 0$, there is no $O(n^{2-\epsilon})$ time constant-space algorithm for counting $\text{hom}(P_4, H)$, where the input graph H is a d -degenerate graph.*

Assuming the above conjecture, we now give characterization of patterns appearing as subgraphs that can be counted in linear time and constant space.

Theorem 18 *Patterns that are countable as subgraphs in linear time and constant space are precisely the graphs with $\text{dtd} = 1$ and the star graphs.*

Proof A DAG with $\text{dtd} = 1$ has exactly one source vertex. Such graphs are either cliques or stars. Moreover, every graph in the spasm of a clique or a star also has $\text{dtd} = 1$. Hence, for each vertex in the spasm of a $\text{dtd} = 1$ graph, the number of homomorphisms can be counted in linear time and constant space. Consequently, the number of subgraphs of a DAG with $\text{dtd} = 1$ can also be computed in linear time and constant space.

Furthermore, there exist orientations of star graphs in which the number of sources is greater than one. However, in such cases, the number of non-sources is exactly one. Therefore, counting star subgraphs only requires computing the degree of each vertex in the host graph, which can be done in linear time and constant space. $\square$

Next to characterize patterns that can be counted in cubic time and constant space, we assume the following conjecture for general graph.

Conjecture 2 For any constant $\varepsilon > 0$, there is no $O(n^{3-\varepsilon})$ -time and constant-space algorithm for counting C_3 (triangles) in an arbitrary input graph G .

Using a similar strategy to that employed in [BPS21], we obtain the following result.

Theorem 19 There is no $O(n^{3-\varepsilon})$ -time and constant-space algorithm for counting C_6 in a bounded-degeneracy graph G with degeneracy equal to two.

Now, consider the obstruction set of graphs with $\text{dtd} = 2$. It can be verified that the spasm of every graph appearing in this obstruction set contains a triangle C_3 when reduced in a manner similar to [BPS21]. Hence, we have the following result.

Theorem 20 There is no $O(n^{3-\varepsilon})$ -time and constant-space algorithm for counting graphs with $\text{dtd} = 3$ in a bounded-degeneracy graph G with degeneracy equal to two.

Next, we show the following theorem that gives characterization for induced subgraphs.

Theorem 21 Let d be a constant. For all patterns with at most nine vertices, we can count the number of occurrences as induced subgraphs in $O(n^3)$ time and constant space for an n -vertex d -degenerate graph given as input.

Proof Assume that the pattern is connected. Consider a DAG where the number of sources is at most three. It is easy to observe that the dtd of such a DAG is bounded by three.

Now consider the case when the number of non-source vertices is at most three. If the number of non-sources is at most two, then we can construct an elimination tree by taking one source vertex as the root and another source vertex reaching the other non-source vertex, while all remaining source vertices can be attached as leaves. If the number of non-sources is three, since the graph is connected, there must exist a source vertex that reaches at least two non-source vertices. We then select one more source vertex that reaches the remaining non-source vertex. Thus, we obtain an elimination tree of height at most three.

Next, consider the case where the DAG has four source vertices. The dtd of this DAG equals four if and only if there exist unique reachable non-source vertices for every pair of source vertices. Hence, such a DAG must contain at least six non-source vertices.

Now consider the case when the number of source vertices is five. Then the number of non-source vertices is four. Let s be a source vertex with the maximum reachability set $R(s)$, so $|R(s)| \geq 2$. If there exists another source vertex s' such that $|R(s) \cup R(s')| = 4$, then we again obtain an elimination tree of height three. Otherwise, for every remaining source vertex s' , we have $|R(s) \setminus R(s')| \leq 1$. In this case, we make s the root of the elimination tree and attach one source vertex as a child corresponding to each remaining non-source vertex. Thus, we again obtain an elimination tree of height at most three.

Therefore, every DAG with at most nine vertices has $\text{dtd} \leq 3$. Since the spasm and all supergraphs of any graph with at most nine vertices also have at most nine vertices, we can count the induced subgraphs of all patterns with at most nine vertices in $O(n^3)$ time and constant space. $\square$

Using the above theorem and the fact that $\text{dtd}(P_{10}) = 3$, we can conclude that

Corollary 1 We can count subgraph P_{10} in time $O(n^3)$ using constant space.

Using Theorem 21, we get $O(n^3)$ time constant space algorithm upto nine vertices. We now show hardness result for ten vertices. Consider a DAG on ten vertices, which is one subdivision of K_4 . Using the reduction shown in the prove of Conjecture 3, we get that it reduces to counting K_4 in general graph.

Now, assuming the conjecture that no $O(n^{4-\varepsilon})$ -time and constant-space algorithm exists, for counting k_4 in general graphs. We get that there is no $O(n^{4-\varepsilon})$ -time and constant-space algorithm for counting k_4 with one subdivision in bounded degenerate graphs.

It is worth noting that efficient algorithms for counting cliques exist using combinatorial techniques or fast matrix multiplication. However, both these approaches require non-constant space. To achieve constant-space algorithms, one must rely on divide-and-conquer strategies based on tree-depth or matched tree-depth. As shown earlier, dtd is bounded above by tree-depth. For sparse patterns, matched tree-depth can lead to faster algorithms. For example, all patterns with at most eleven edges can be counted in $O(n^3)$ time and constant space. Consequently, as a corollary, the paths P_{12} and cycles C_{11} can also be counted as subgraphs in $O(n^3)$ time and constant space. However, for dense patterns and induced subgraph counting, DAG treedepth provides a more advantageous framework compared to previously known constant-space algorithms.

3.4 Constant Space computation of $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$: In this section we show that we can compute $\text{ind}(H, G)$ in time $O(n^{\lfloor \frac{k}{4} \rfloor + 2})$ and $\text{hom}(H, G), \text{sub}(H, G)$ in time $\min(O(n^{\lfloor \frac{k}{4} \rfloor + 2}), O(n^{\lfloor \frac{l}{5} \rfloor + 3}))$ using $O(1)$ -space, where $|V_H| = k$, $|E_H| = l$ and, n is the number of vertices in G . To this end, we prove the following theorem.

Theorem 22 *Let H be a pattern DAG with k vertices. Then, $\text{dtd}(H) \leq \lfloor \frac{k}{4} \rfloor + 2$.*

Proof We make a DAG elimination tree of depth at most $\lfloor \frac{k}{4} \rfloor + 2$. For this, we add source vertices that maximize the number of discovered vertices.

We incrementally construct a root-to-leaf path T_p of a DAG elimination tree. Let D be the set of discovered (processed) vertices, initialized as empty. At each step, select a source vertex s such that $|R(s) \setminus R(D)|$ is maximized. We add s as a child in T_p .

- **Case 1:** $|R(s_i) \setminus R(D)| \geq 3$

Adding such a source s_i increases the depth by one but makes at least four vertices discovered (one source and three non-sources). Since all subsequent sources are added below s_i , the reachability condition is preserved.

Now, consider the residual graph G' after deleting all sources added so far and their reachable vertices. If $|R(s_i)| = 0$, then add s_i as a child to the last vertex added in D . This increases depth by at most one. If $|R(s_i)| = 1$, say $R(s_i) = \{u\}$ then if a source s already in D reaches u , attach s_i under s . Otherwise, attach s_i under the last added vertex in D , and attach any other sources that also reach only u as its children.

In all these cases, the depth increases by at most two. We now reduce the graph to a residual subgraph where each source reaches exactly two non-sources, and each non-source is reached by at least two sources.

- **Case 2:** $|R(s_i) \setminus R(D)| = 2$ and $|P(u) \setminus D| \geq 2$

If there exists a source s such that $R(s) \cap R(s_i) = \emptyset$ for all s_i , we can add s as a child to the last source in D , and attach all sources whose reachable sets are contained in $R(s)$ under s . This increases depth by at most 2.

Now suppose that every source s_i has some overlap $R(s_i) \cap R(s_j) \neq \emptyset$ with another source s_j . Then, adding s_i removes at least four vertices from the graph: the source s_i and two non-sources shared with other sources. If any leftover source s_j now reaches only one undiscovered vertex, we can handle it as in Case 1.

In every case, we discover at least 4 vertices when we increase DAG treedepth by one. Also, for corner cases, when $|R(s)| \leq 1$, then we can manage such cases by increasing treedepth by two. Hence, the depth of the elimination tree of the DAG is bounded by $\lfloor \frac{k}{4} \rfloor + 2$. Therefore, $dtd(H) \leq \lfloor \frac{k}{4} \rfloor + 2$. $\square$

Using [Theorem 4](#) and [Theorem 22](#), we get the following result.

Theorem 23 Consider any k -node pattern graph $H = (V_H, E_H)$. Then one can compute $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$ in $O(f(k, d) \cdot n^{\lfloor \frac{k}{4} \rfloor + 2})$ time using $O(1)$ -space.

The above theorem, keeping the time complexity the same, improves the space complexity of [\[B21\]](#) from polynomial to a constant for counting all patterns of size k .

Theorem 24 Let H be a DAG with l edges. Then, $dtd(H) \leq \lfloor \frac{l}{5} \rfloor + 3$.

Proof We follow a similar strategy to that in [Theorem 22](#). The key idea is to show that for each unit increase in depth, we can safely eliminate at least 5 edges from the graph. The additive constant accounts for small residual cases.

As discussed before, after handling sources with low reachability (e.g., reaching at most one non-source), we reduce the graph to a simpler form where:

- Every non-source vertex is reachable by at least two sources.
- For any two sources s_i and s_j , $R(s_i) \not\subseteq R(s_j)$.
- For any two non-sources u and v , $P(u) \not\subseteq P(v)$.

When we add a source vertex s to the elimination path T_p , we eliminate all edges adjacent to its reachable non-sources $u \in R(s)$. Thus, the number of eliminated edges is at least $\sum_{u \in R(s)} \deg(u)$.

- **Case 1:** $|R(s_i) \setminus R(D)| \geq 3$

Suppose s_i reaches distinct non-sources u_1, u_2, u_3 , and for each pair $u_j \neq u_k$, the sets $P(u_j) \setminus P(u_k)$ are non-empty as $P(u_j) \not\subseteq P(u_k)$. This implies that each non-source u_i contributes at least 2 edges, where one edge is between the source vertex s_i and u_i and one edge between u_i and some source s . Note that such source exist because $|P(u_i)| \geq 2$. Also, if $|P(u_i)| = 2$, then $P(u_i) \cap P(u_j) = \{s_i\}$ otherwise, $P(u_i) \subseteq P(u_j)$. Therefore, we eliminate at least 6 edges by adding s_i , increasing the depth by 1.

- **Case 2:** $|R(s_i) \setminus R(D)| = 2$

- **Case A:** There exists a non-source $u \in R(s_i)$ with $|P(u)| \geq 3$

Let s_i reach u_1 and u_2 , and assume $|P(u_1) \setminus P(u_2)| \geq 1$ otherwise $P(u_1) \subseteq P(u_2)$. Let $|P(u_2)| \geq 3$ and $|P(u_1)| \geq 2$. So, $|P(u_1) \cup P(u_2)|$ is at least 4. Now, if $|P(u_1) \cup P(u_2)|$ is at least five then we can say that adding only one vertex s_i , we eliminate at least five edges. Now, consider the case $|P(u_1) \cup P(u_2)| = 4$. It is only possible when $|P(u_1)| = 2$. So, consider source $s \in P(u_1)$ other than s_i . We can check that $|R(s) \setminus R(D)| = 1$, we can eliminate source s after adding s_i to the path and increasing depth one. s can be handled as described above using the additive factor two when source is reaching at most one undiscovered non-sources. Thus, in this step also, we increased DAG treedepth by one and eliminated five edges.

- **Case B:** Every non-source u has $|P(u)| = 2$

In this scenario, the residual graph consists only of cycles or disjoint cycles. For a cycle with l' edges, its DAG treedepth is at most $\log(\lceil l'/2 \rceil)$. Since $\frac{l'}{5} + 1 \geq \log(\lceil l'/2 \rceil)$ for all l' , the depth bound still holds.

Thus, in all cases, we can safely remove at least 5 edges per unit increase in depth (with an additive constant), which gives us that $\text{dtd}(H) \leq \lfloor \frac{l}{5} \rfloor + 3$. □

Again, using [Theorem 4](#) and [Theorem 24](#), we have the following result.

Theorem 25 *Consider any l -edge pattern graph $H = (V_H, E_H)$. Then one can compute $\text{Hom}(H, G)$ and $\text{Sub}(H, G)$ in $O(f(k, d) \cdot n^{\lfloor \frac{l}{5} \rfloor + 3})$ time using $O(1)$ -space.*

Remark 3 *From [Theorem 23](#) and [Theorem 25](#), we can compute $\text{hom}(H, G)$ and $\text{sub}(H, G)$ in time $\min(O(f(k, d) \cdot n^{\frac{k}{4} + 2}), O(f(k, d) \cdot n^{\frac{l}{5} + 3}))$ using $O(1)$ -space.*

In [Theorem 24](#), we observed that in Case 1, we can achieve a bound of $\frac{\ell}{6} + 2$ on the DAG treedepth. For the remaining cases, we can construct an undirected graph G as described in [Lemma 12](#), where the vertex set consists of non-source vertices from the residual DAG G' . Importantly, the number of edges in G is equal to the number of source vertices in G' , and the number of edges in G' is exactly twice the number of its source vertices.

Now, suppose there exists a general upper bound on treedepth for undirected graphs in terms of the number of edges, i.e., $td(G) \leq \frac{m}{\alpha}$. Then, applying this to the graph G , we obtain: $\text{dtd}(G') \leq \frac{m}{2\alpha}$. This leads to the following conditional corollary:

Corollary 2 *If for any undirected graph G with m edges, $td(G) \leq \frac{m}{3}$, then for any pattern DAG H with ℓ edges, $\text{dtd}(H) \leq \frac{\ell}{6} + 2$.*

Next, we deal with DAG treewidth. Since treewidth is a well-studied graph parameter and efficient approximation algorithms exist for computing it [[B25](#)], it is reasonable to bound other parameters in terms of treewidth. Therefore in the next section, we first define DAG treewidth [[B21](#)] and obtain many relationships between the DAG treewidth and the treewidth of graphs in $\mathcal{G}$.

4 DAG Treewidth

In this section, we first establish several fundamental properties of DAG treewidth. Building upon these we prove that for any DAG H with k vertices, $\text{dtw}(H) \leq \frac{k}{5} + 3$, and leverage this bound to derive improved runtime for computing $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$. We further prove a conjecture recently posed by [[PPS24](#)] on 10-vertex pattern graph H . Along the way, we obtain several additional structural bounds which may be of independent interest. To this end, we first define the DAG treewidth.

Definition 17 (*DAG Tree Decomposition*) *Let $\vec{H}$ be a DAG with a set of sources S . A DAG tree decomposition of $\vec{H}$ is a rooted tree $T = (\mathcal{B}, E)$, where each node $B \in \mathcal{B}$ is a bag consisting of a subset of sources from S , i.e., $B \subseteq S$, and the following properties are satisfied:*

- **Coverage:** Every source appears in at least one bag, i.e., $\bigcup_{B \in \mathcal{B}} B = S$.

- **Reachability Intersection Property:** For any three bags $B, B_1, B_2 \in \mathcal{B}$, if B lies on the unique path between B_1 and B_2 in the tree T , then

$$R_H(B_1) \cap R_H(B_2) \subseteq R_H(B).$$

Definition 18 (DAG Treewidth) Let the DAG width of a DAG tree decomposition $T = (\mathcal{B}, E)$ be defined as the size of the largest bag in T , i.e., the maximum number of sources in any bag. Then, the DAG treewidth of a directed acyclic graph $\vec{H}$ is the minimum DAG width over all possible DAG tree decompositions of $\vec{H}$. For an undirected graph H , the DAG treewidth of H is the maximum of the DAG treewidth of all acyclic orientations of $\vec{H}$. We denote the DAG treewidth of a graph $\vec{H}$ by $\text{dtw}(\vec{H})$.

Observation 2 Let $\vec{H}$ be a directed acyclic graph (DAG) with a fixed orientation. Suppose there are two sources u and v such that there exists a vertex $w \in R_H(u) \cap R_H(v)$, and $w \notin R_H(s_i)$ for any other source $s_i \neq u, v$. Then, it is easy to see that in any DAG tree decomposition of $\vec{H}$, the sources u and v must appear either in the same bag or in adjacent bags.

Observation 3 As edge contractions do not increase treewidth, we have $\text{tw}(G_S) \leq \text{tw}(\text{BIP}(\vec{H}))$.

Let $T = (\mathcal{B}, E)$ be a DAG tree decomposition. We denote by B_{leaf} a leaf node of the tree T . For any node $B \in \mathcal{B}$, we denote its parent by $\text{PARENT}(B)$. We denote the root of the tree T by $\text{ROOT}(T)$. For any two nodes $B_1, B_2 \in \mathcal{B}$, the unique path between B_1 and B_2 in T is defined as $T(B_1, B_2)$.

4.1 Properties of DAG Treewidth: In this section, we establish several important properties of treewidth and DAG treewidth ($\text{dtw}(\vec{H})$), along with important relationships between them. By definition, the DAG treewidth of an undirected graph is defined as the maximum DAG treewidth taken over all its acyclic orientations. Consequently, results derived for DAGs can be naturally extended to undirected graphs by considering all possible acyclic orientations. Therefore, any property proven for a fixed acyclic orientation of a DAG applies to the corresponding undirected graph as well. However, in [Theorem 27](#) (which contracts a single edge, reducing the number of vertices by one), [Lemma 10](#) (which compares treewidth of undirected graphs), and [Lemma 11](#) (which introduces a new vertex on every edge (known as a subdivision vertex)), it is necessary to account for all acyclic orientations. For this reason, we present the proofs of these lemmas in the undirected setting.

Treewidth is a well-studied structural parameter that plays a central role in the design of dynamic programming algorithms for many NP-complete problems on graphs of bounded treewidth. In particular, for a pattern graph H with treewidth t , the number of homomorphisms from H to a host graph G with $n = |V(G)|$ can be computed in time $n^{O(t)}$.

Bressan [B21] introduced DAG treewidth to enable near-linear time algorithms for counting homomorphisms from certain graphs, such as cliques and complete bipartite graphs, whose treewidth is unbounded, but whose DAG treewidth is just 1. This highlights DAG treewidth as a powerful and relevant parameter for homomorphism counting. But here we derive a hardness result for DAG treewidth. To this end we show the following lemma.

Lemma 7 Let H be a DAG with the set of sources S and non-sources T . For each source $s \in S$, let $R(s) \subseteq T$ denote the set of vertices reachable from s . Construct a hypergraph $\mathcal{H}$ whose vertex set is T and whose hyperedge set is $\mathcal{E} = \{R(s) \mid s \in S\}$. Then the DAG treewidth of H equals the generalized hypertree width of $\mathcal{H}$, i.e.,

$$\text{dtw}(H) = \text{ghw}(\mathcal{H}).$$

Proof ($dtd \Rightarrow ghd$) Let $(T_\tau, \{\beta(t)\}_{t \in V(T_\tau)})$ be a DAG tree decomposition of H . For each node t , construct a GHD node with $\chi(t) = \beta(t) \cap T$ and $\lambda(t) = \{R(s) : s \in \beta(t) \cap S\}$. The coverage and connectedness conditions of a GHD follow directly from those of the DTD. Since each source s corresponds to a single hyperedge $R(s)$, the width (measured by the number of sources/hyperedges) is preserved. Hence, $ghw(\mathcal{H}) \leq dtw(H)$.

($ghd \Rightarrow dtd$) Conversely, let $(T_\tau, \{(\chi(t), \lambda(t))\}_{t \in V(T_\tau)})$ be a generalized hypertree decomposition of $\mathcal{H}$. We obtain a directed tree decomposition of G by replacing each hyperedge $R(s) \in \lambda(t)$ with its corresponding source s . Formally, define $\beta(t) = \chi(t) \cup \{s \in S : R(s) \in \lambda(t)\}$. The coverage and connectedness properties of a DTD follow from those of the GHD, and the width is again preserved. Hence, $dtw(H) \leq ghw(\mathcal{H})$. Combining both directions yields $dtw(H) = ghw(\mathcal{H})$. $\square$

To obtain hardness result for DAG treewidth, we use the result of [FGP18], which shows that $ghw(\mathcal{H} \leq 2)$ is NP-complete. To this end we have the following result.

Theorem 26 *Let H be a DAG, then deciding whether $dtw(H) \leq 2$ is NP-complete.*

Proof The prove follows using Lemma 7 and the fact that deciding whether $ghw(\mathcal{H} \leq 2)$ is NP-complete. $\square$

Next, as the relationship between treewidth and DAG treewidth remains relatively unexplored, we aim to establish a more concrete connection between these two parameters. Leveraging the construction $\mathcal{G}$ defined in Section 2.1, we prove that there exists graphs $G_1, G_2 \in \mathcal{G}$ such that the following bounds hold: $\frac{dtw(G_1)+1}{2} \leq dtw(\vec{H}) \leq dtw(G_2) + 1$. To this end in Lemma 8 we show that $dtw(\vec{H}) \leq tw(G_S) + 1$ and in Lemma 9 we show that $dtw(\vec{H}) \geq \frac{tw(G_S)+1}{2}$.

Lemma 8 *For a given DAG $\vec{H}$ there exist a $G_S \in \mathcal{G}$ such that $dtw(\vec{H}) \leq tw(G_S) + 1$.*

Proof T be a tree decomposition of the graph G_S . From the construction described in Section 2.1, we know that $V(G_S) = S$, where S is the set of sources in the DAG $\vec{H}$. By the definition of a tree decomposition, each bag in T is a subset of the vertex set $V(G_S) = S$, and the union of all bags covers S . Thus, T satisfies the *coverage property* of the DAG-tree decomposition (DTD) (see Definition 17).

To complete the proof of the lemma, it remains to show that T also satisfies the *reachability intersection property*. Specifically, for all bags $B, B_1, B_2 \in \mathcal{B}$, if B lies on the unique path between B_1 and B_2 in the tree T , then we must have:

$$R_H(B_1) \cap R_H(B_2) \subseteq R_H(B).$$

To prove this, we define: $Y_1 = B_1 \setminus B_2$ and $Y_2 = B_2 \setminus B_1$. Since T is a tree decomposition, the *connectivity property* implies that for any vertex $v \in B_1 \cap B_2$, all bags containing v must lie along the path between B_1 and B_2 , which includes B . Therefore, $B_1 \cap B_2 \subseteq B$, and consequently:

$$R_H(B_1 \cap B_2) \subseteq R_H(B).$$

Thus, to prove the reachability intersection condition, it is sufficient to consider the contributions from Y_1 and Y_2 . This leads us to two cases:

1. $R_H(Y_1) \cap R_H(Y_2) = \emptyset$.
2. $R_H(Y_1) \cap R_H(Y_2) \neq \emptyset$.

For both cases, we want to show that the reachability condition holds.

Case 1: $R_H(Y_1) \cap R_H(Y_2) = \emptyset$

We know that $R_H(B_1) = R_H(Y_1 \cup (B_1 \cap B_2))$, and $R_H(B_2) = R_H(Y_2 \cup (B_1 \cap B_2))$. Taking the intersection we have,

$$R_H(B_1) \cap R_H(B_2) = (R_H(Y_1) \cap R_H(Y_2)) \cup R_H(B_1 \cap B_2).$$

Since $R_H(Y_1) \cap R_H(Y_2) = \emptyset$ by assumption, the intersection simplifies to:

$$R_H(B_1) \cap R_H(B_2) = R_H(B_1 \cap B_2).$$

From the connectivity property of tree decompositions, $B_1 \cap B_2 \subseteq B$, so:

$$R_H(B_1 \cap B_2) \subseteq R_H(B).$$

Thus, the reachability condition holds in this case.

Case 2: $R_H(Y_1) \cap R_H(Y_2) \neq \emptyset$

This is the more involved case where the reachability sets of Y_1 and Y_2 overlap. Our goal remains to show that every vertex in $R_H(B_1) \cap R_H(B_2)$ is also in $R_H(B)$.

Suppose there exist sources $u \in Y_1$ and $v \in Y_2$ such that $R_H(u) \cap R_H(v) \neq \emptyset$. We consider two subcases based on how the common reachable non-source vertices are connected in the graph G_S .

(A) *There exists a non-source vertex that is reachable only from u and v :*

From the construction of the graph G_S , we know that in such a case, the edges connecting this non-source vertex to u and v would be contracted. This results in an edge $\{u, v\} \in E(G_S)$, implying that T , the tree decomposition of G_S , contains a bag with both u and v .

(B) *The common reachable non-source vertices are also reachable from some other set of sources:*

If, for any such non-source vertex in $R_H(u) \cap R_H(v)$, the edge contractions in G_S were performed using u and v , then again $\{u, v\} \in E(G_S)$, and T contains a bag with both u and v .

However, consider the case where edge contractions were performed using a different set of sources, say, a set $\{S'\}$ such that each non-source vertex in $R_H(u) \cap R_H(v)$ is reachable from some source in S' . These sources in S' are then connected via contractions to the non-sources and become common neighbors of both u and v in G_S . As a result, the tree decomposition T must include intermediate bags containing these shared neighbors, which lie on the path between u and v in T .

Since u and v were chosen arbitrarily from Y_1 and Y_2 , this argument extends to show that all shared reachable vertices in $R_H(Y_1) \cap R_H(Y_2)$ must be included in the reachability of some intermediate bag B on the path between B_1 and B_2 . Therefore,

$$R_H(B_1) \cap R_H(B_2) \subseteq R_H(B),$$

for all such bags B on the path from B_1 to B_2 in T .

This shows that T satisfies both the coverage and the reachability intersection properties required for a valid DAG-tree decomposition. Finally, by definition, the DAG-treewidth of $\vec{H}$ is the maximum

size of any bag in the DAG-tree decomposition. Since the treewidth of G_S is one less than the size of its largest bag, we conclude that $\text{dtw}(\vec{H}) \leq \text{tw}(G_S) + 1$. □

Using [Observation 3](#) and [Lemma 8](#) we get the following remark:

Remark 4 For a given DAG $\vec{H}$ and corresponding $\text{BIP}(\vec{H})$, $\text{dtw}(H) \leq \text{tw}(\text{BIP}(\vec{H})) + 1$.

Remark 5 The above bound is tight. For example, consider the cycle graph C_5 . For any acyclic orientation of C_5 , the DAG treewidth is $\text{dtw}(C_5) = 1$, while the treewidth is $\text{tw}(C_5) = 2$.

For a more general case, consider the graph G obtained by subdividing every edge of the complete bipartite graph $K_{n,n}$ exactly once. Use an orientation where the original vertices have out-degree only (i.e., all original vertices are sources), and the newly added subdivision vertices are non-sources.

In this construction, each vertex in one part of the bipartite graph has a unique reachable non-source shared with every vertex in the opposite part. By [Observation 2](#), all sources from one side must appear together (or in adjacent bags) in any DAG tree decomposition. Hence, any bag must contain at least n source vertices, out of the total $2n$ sources. This gives $\text{dtw}(G) \geq n$.

Moreover, we can construct a valid DAG tree decomposition by putting all sources from one side in the root bag and attaching one leaf bag for each source from the other side. This shows that $\text{dtw}(G) = n$, matching the lower bound.

Lemma 9 For a given DAG $\vec{H}$ and a corresponding set of graphs $\mathcal{G}$, there exist $G_S \in \mathcal{G}$ such that $\text{dtw}(\vec{H}) \geq \frac{\text{tw}(G_S) + 1}{2}$.

Proof Let $\vec{H}$ be a fixed acyclic orientation, and let $T = (\mathcal{B}, E)$ be a DAG tree decomposition (DTD) of $\vec{H}$. Let G_S be the graph obtained from $\vec{H}$ using the construction described in [Section 2.1](#). Note that the vertex set of G_S consists of the source vertices of $\vec{H}$. Since each bag in T contains only source vertices and the union of all bags equals $V(G_S)$, the first condition of a tree decomposition for G_S is satisfied.

To complete the construction of G_S , we perform a sequence of edge contractions on $\text{BIP}(\vec{H})$, the bipartite graph built from $\vec{H}$ as described earlier. The contraction process is guided by the structure of T , proceeding in a bottom-up fashion.

For each leaf node $u \in B_{\text{leaf}}$ in T and its parent $v = \text{PARENT}(u)$, if $R(u) \cap R(v) \neq \emptyset$, we contract the non-source vertices in this intersection with v . In the next step, we consider the grandparent bag $B = \text{PARENT}(v)$ and perform edge contractions for non-source vertices in $(R(B) \cap R(v)) \setminus (R(B) \cap R(u))$, as contractions involving vertices in $R(B) \cap R(u)$ have already been completed. This recursive process continues until all contractions are completed. We now claim the following:

Claim 1 If u and v belong to the same bag or to adjacent bags in T , then $\{u, v\} \in E(G_S)$.

Proof Suppose, for contradiction, that $\{u, v\} \in E(G_S)$, but u and v do not appear in the same or adjacent bags in T . Let B_1 and B_2 be bags containing u and v , respectively, and consider the unique path $T(B_1, B_2)$. Let $B \in T(B_1, B_2)$ be a bag that contains neither u nor v . Since T is a valid DAG tree decomposition, it satisfies the reachability condition, implying that there exists a minimal set of source vertices $\{u_1, \dots, u_k\} \subseteq B$ such that:

$$R(u) \cap R(v) \subseteq \bigcup_{i=1}^k (R(u) \cap R(u_i)).$$

Due to edge contraction, each u_i contracts with the non-source vertices in its reachability set. Consequently, $\{u, u_i\}$ and $\{v, u_i\}$ are edges in G_S , implying that $\{u, v\} \notin E(G_S)$, a contradiction. $\square$

Next, we verify the *connectivity condition* of the tree decomposition. Suppose a source vertex u appears in two bags B_1 and B_2 , but not in any bag along the path $T(B_1, B_2) \setminus \{B_1, B_2\}$. From the reachability property of T , we know:

$$R(B_1) \cap R(B_2) \subseteq R(B),$$

for any intermediate bag B . Since $R(u) \subseteq R(B_1) \cap R(B_2)$, we must have $R(u) \subseteq R(B)$ as well. Therefore, there exists a minimal set of source vertices $\{u_1, \dots, u_k\}$ such that:

$$R(u) \subseteq \bigcup_{i=1}^k R(u_i).$$

This implies that for every non-source vertex $w \in R(u)$, there exists some $u_i \neq u$ such that $w \in R(u_i)$, allowing the contraction of w with u_i . As a result, all of u 's neighbors in G_S are contained in $\{u_1, \dots, u_k\}$, and so u can safely be removed from B_2 , maintaining the connectivity condition.

We now verify the *second condition* of the tree decomposition for G_S , namely, that for every edge $\{u, v\} \in E(G_S)$, there exists a bag that contains both u and v . From [Claim 1](#), such edges arise only when u and v are in the same or adjacent bags in T . We construct a new tree decomposition T' from T by adding for each bag $B \in T$ its parent $\text{PARENT}(B)$ as an additional bag in T' . Let $\text{ROOT}(T') = \text{ROOT}(T)$, this construction guarantees that for every edge $\{u, v\} \in E(G_S)$, there exists a bag in T' containing both vertices.

Let k be the maximum bag size in T . Then each new bag in T' has size at most $2k$, and hence $\text{tw}(G_S) \leq 2k - 1$. Therefore, $k \geq \frac{\text{tw}(G_S) + 1}{2}$. Since k is the width of the DAG-tree decomposition T , this completes the proof. $\square$

Remark 6 The above bound is tight. Consider the graph G obtained by subdividing every edge of K_5 . We orient all original vertices to point outward, making them sources. For each pair of original vertices, there exists a unique non-source vertex reachable from both the original vertices. By [Observation 2](#), such sources must appear together in the same or adjacent bags of any valid DAG tree decomposition. Therefore, $\text{dtw}(G) = 3$, while $\text{tw}(G) = 5$.

The following lemma shows that $\text{dtw}(\vec{H}) \leq \text{tw}(H) + 1$. This bound suggests that the DAG treewidth of a graph is never significantly worse than its treewidth.

Lemma 10 For any undirected graph H , $\vec{H}$ is any acyclic orientation of H . Then, $\text{dtw}(\vec{H}) \leq \text{tw}(H) + 1$.

Proof Let T be a tree decomposition of H where $\text{tw}(H) = k$. Fix any arbitrary acyclic orientation of $\vec{H}$. Observe that every source in $\vec{H}$ must appear in some bag of T , since all vertices are covered by the decomposition.

Now, consider any pair of source vertices s_i and s_j such that their reachability sets intersect, i.e., $R(s_i) \cap R(s_j) \neq \emptyset$. Suppose $s_i \in B_1$ and $s_j \in B_2$, where B_1 and B_2 are bags in T . Let $T(B_1, B_2)$ denote the unique path between B_1 and B_2 in the tree decomposition. If $R(s_i) \cap R(s_j) \subseteq R(B)$ for all intermediate bags $B \in T(B_1, B_2)$, then the reachability condition is already satisfied. Otherwise, let $u \in R(s_i) \cap R(s_j)$ be a vertex that is not contained in any such intermediate bag.

Since $u \in R(s_i) \cap R(s_j)$, there exists a directed path from s_i and s_j to u , consisting only of non-source vertices:

$$s_i \rightarrow u_1 \rightarrow \cdots \rightarrow u \leftarrow u'_1 \leftarrow \cdots \leftarrow u'_l \leftarrow s_j.$$

Therefore, there is a corresponding path in the tree decomposition:

$$B_1 - B'_1 - B'_2 - \cdots - B'_l - B_2$$

where each bag B'_i must contain at least one vertex from the above path, by the property of the tree decomposition. In each such bag B'_i , we replace one non-source vertex (e.g., u_k or u'_k) with the source vertex s_i . This replacement does not increase the size of the bag. By applying this process iteratively to all these source pairs s_i, s_j , we can ensure that the reachability condition for a DAG tree decomposition is satisfied throughout the tree. Thus, $dtw(\vec{H}) \leq tw(H) + 1$. □

The treewidth (tw) of a graph remains invariant under any number of edge subdivisions. In many graph classes with bounded treewidth, the number of vertices can far exceed the treewidth. It is also known that cliques and quasi-cliques typically exhibit high treewidth but low DAG treewidth. However, even a single edge subdivision in such graphs can cause a significant increase in DAG treewidth. For example, $tw(K_n) = n - 1$ and $dtw(K_n) = 1$ but with only one subdivision we get that $tw(K_{n_{sub}}) = n - 1$ whereas $dtw(K_{n_{sub}}) = \lceil \frac{n}{2} \rceil$.

To account for this behavior, [Lemma 11](#) establishes an upper bound on the DAG treewidth following a single subdivision per edge. This result is especially useful for analyzing the DAG treewidth of cliques and quasi-cliques under limited subdivision. By combining [Lemma 11](#) with [Lemma 10](#), one can obtain a sharper upper bound on DAG treewidth in such settings.

Lemma 11 *Let H be an undirected graph with n vertices. Let G_{sub} be a graph obtained by subdividing each edge of H at most once. Then, $dtw(G_{sub}) \leq \lceil \frac{n}{2} \rceil$.*

Proof Let O_i be arbitrary but fixed acyclic orientation of G_{sub} and let $S = \{s_1, s_2, \dots, s_k\}$ be the set of source vertices in G_{sub} . These source vertices may be original vertices of H or subdivision vertices introduced in G_{sub} .

If $|S| \leq n$, then we can partition the source set S into two bags such that each bag contains at most $\lceil \frac{n}{2} \rceil$ vertices.

Now consider the case when $|S| > n$. This means some of the subdivision vertices in G_{sub} are also source vertices. We construct a DAG tree decomposition T as follows: Initialize T with two bags, B_1 and B_2 . We also maintain two corresponding lists $SEEN(B_1)$ and $SEEN(B_2)$, which keep track of the vertices from H added so far.

Step 1 Identify all source vertices that are also original vertices of H . Add them alternately to B_1 and B_2 . Each time we add such a vertex to a bag, we also add it to the corresponding $SEEN$ list. Note that if a vertex v from H is a source in G_{sub} , then none of the subdivision vertices adjacent to v can be a source.

Step 2 Handle source vertices that are subdivision vertices. For a subdivision vertex uv , which comes from subdividing edge $\{u, v\}$ in H :

- **Step A:** If both u and v are not present in either $SEEN(B_1)$ or $SEEN(B_2)$, add uv to both B_1 and B_2 . Also, add u and v to both $SEEN(B_1)$ and $SEEN(B_2)$.
- **Step B:** Now look for a source vertex vv_i where v_i is not yet in either $SEEN(B_1)$ or $SEEN(B_2)$. Add vv_i to the smaller of B_1 or B_2 and update the corresponding $SEEN$ list by adding v_i . Continue

this process: alternate between B_1 and B_2 , adding new source vertices and updating SEEN lists accordingly.

If you reach a subdivision vertex $v_p v_q$ such that no further extension is possible (i.e., there's no next source $v_q v_r$ with r not in SEEN), then go back to Step A and repeat the process with a fresh $v_s v_t$ where both endpoints are still unseen.

Through these procedures, we ensure that (almost) all vertices of H are eventually added to $\text{SEEN}(B_1)$ or $\text{SEEN}(B_2)$, except potentially the vertex v_q where Step B terminates. Moreover, no further source $v_q v_r$ exists with r outside both SEEN lists.

Step 3 Now handle the remaining subdivision source vertices $v_s v_t$ where v_s is already in some SEEN list and v_t is not. Add $v_s v_t$ to the smaller of B_1 or B_2 and update the respective SEEN list by including v_t . Repeat this until all vertices from H are in one of the SEEN lists.

After Steps 1–3, we claim that both bags have size at most $\lceil \frac{n}{2} \rceil$. Why? Because each time we add a source vertex, we are effectively marking a new vertex from H as SEEN. In Step 2A, while a source vertex is added to both bags, it causes two H vertices to be added to each SEEN list. Additionally, we always ensure the bag sizes differ by at most 1, so the maximum size of each bag remains within $\lceil \frac{n}{2} \rceil$.

Now, there may still be some remaining source vertices $u_l v_l$ such that both u_l and v_l have already been added to the SEEN lists. For these vertices, we make the following claim:

Claim 2 *Each remaining such source vertex $u_l v_l$ has both endpoints in the same SEEN list, either $\text{SEEN}(B_1)$ or $\text{SEEN}(B_2)$.*

Proof Suppose u_l is in $\text{SEEN}(B_1)$ and v_l is in $\text{SEEN}(B_2)$. Then u_l must have been added through Step 2 or 3. If it was added in Step 2 (except at the stopping point), then it must have been added to both SEEN lists. If u_l were the terminal vertex of Step 2B, then we would still be able to process $u_l v_l$ in the next step, contradicting the assumption that it is a leftover. Similarly, if both u_l and v_l were previously unseen, they would have been picked up by Step 2A. So, either both endpoints were added together, or they ended up in the same SEEN list through the process. $\square$

Step 4 For all such remaining source vertices $u_l v_l$, attach a *leaf bag* to the corresponding bag B_i (either B_1 or B_2), containing just the vertex $u_l v_l$. We attach it to B_i such that both u_l and v_l are in $\text{SEEN}(B_i)$.

Now, we argue that the DAG tree decomposition T is valid. All sources are included in some bag. For the reachability condition, observe that for a subdivision vertex uv , we have $R_T(uv) \subseteq R_T(u) \cap R_T(v)$. (Here, we are slightly abusing notation since u and v may not be sources themselves.)

All subdivision sources added in Step 1–3 are in either the same or adjacent bags, so reachability is preserved trivially. For those added in Step 4, the claim ensures both u_l and v_l are in the same SEEN list, hence their reachability is covered by the parent bag they are attached to. Thus, the decomposition is valid, and each bag has size at most $\lceil \frac{n}{2} \rceil$. Therefore, $\text{dtw}(G_{\text{sub}}) \leq \lceil \frac{n}{2} \rceil$. $\square$

Remark 7 *The above bound is tight. Consider the graph $H = K_4$, and let G_{sub} be the graph obtained by subdividing each edge of H exactly once. If we orient each edge from the original vertex toward its corresponding subdivision vertex, then we can verify that $\text{dtw}(G_{\text{sub}}) = 2$.*

Observation 4 *Unlike treewidth (tw), the DAG treewidth (dtw) is not closed under minors. For example, the complete graph K_6 has only one source in any acyclic orientation, which implies $\text{dtw}(K_6) = 1$. However, the cycle graph C_6 , which is a subgraph of K_6 , satisfies $\text{dtw}(C_6) = 2$. This shows that dtw is not even closed under subgraphs.*

For a non-source vertex u in a DAG $\vec{H}$, we define the set $P_H(u)$ as the set of source vertices from which u is reachable. That is, $P_H(u) = \{s \in S \mid u \in R_H(s)\}$, where S is the set of sources in $\vec{H}$ and $R_H(s)$ denotes the set of vertices reachable from source s . Let I be an induced minor of a graph H , then we show the following theorem.

Theorem 27 *Let H be an undirected graph, and let I be an induced minor of H . Then, $dtw(H) \geq dtw(I)$.*

Proof To obtain this result, it suffices to prove the following two claims.

1. If I is an induced subgraph of H then $dtw(H) \geq dtw(I)$.
2. Let H be an undirected graph with $dtw(H) = k$. Let H' be the undirected graph obtained by contracting a single edge in H . Then, $dtw(H') \leq k$.

Proof of Claim 1: Let $dtw(I) = k$. By definition, there exists an acyclic orientation O_1 of I such that $dtw(I) = k$ under this orientation. We now construct an acyclic orientation of H that extends O_1 and preserves the treewidth. Let $I' \subseteq H$ be an induced subgraph isomorphic to I , with an isomorphism $f: V(I) \rightarrow V(I')$. Fix the orientation of I' in H according to O_1 .

Let $V(H) = V(I') \cup (V(H) \setminus V(I'))$. Partition the remaining vertices based on their distance from I' in H . Define layers $D_1, D_2, \dots, D_j$, where

$$D_i = \{v \in V(H) \setminus V(I') \mid \text{dist}_H(v, V(I')) = i\}.$$

Now, define a topological ordering on the vertices of H as:

$$V(I'), D_1, D_2, \dots, D_j.$$

Orient the edges from each D_i to D_{i+1} in the forward direction (i.e., from smaller to larger layers), and similarly orient the edges from $V(I')$ to D_1 forward. For edges within any D_i , choose an arbitrary acyclic orientation.

Note that after this orientation, we have $S(I) = S(\vec{H}) \cap I'$. Also, for any non-source u in I , we have $P_I(u) = P_{\vec{H}}(u)$ i.e., there is no source reaching u other than the source vertices in I .

Now, suppose for contradiction that $dtw(H) < k$ under this orientation. Let T be a valid DAG-tree decomposition of H with DAG treewidth less than k . Then, the restriction of T to the subgraph I' forms a valid DAG-tree decomposition of I' , and hence of I (by isomorphism), contradicting the assumption that $dtw(I) = k$. Therefore, $dtw(H) \geq dtw(I)$.

Proof of Claim 2: The DAG treewidth of an undirected graph H is the maximum of the DAG treewidth of DAG $\vec{H}$. We want to show $dtw(H') \leq k$, where H' is the undirected graph obtained after contraction of an edge $\{u, v\}$. So, for all acyclic orientations of H' , we have to show that the DAG treewidth of H' is bounded by k . We pick an arbitrary but fixed acyclic orientation of H' . We copy the same orientation in H . Note that $\{u, v\} \notin E(H')$. So, the orientation is not known. We first give (u, v) orientation to $\{u, v\}$. Note that the orientation of H is acyclic. Let w be the vertex in H' obtained after contracting $\{u, v\}$. Let T be a DAG tree decomposition of $\vec{H}$ of width at most k . We will now construct DAG tree decomposition T' of $\vec{H}'$.

- **Case 1:** Both u and v are non-source vertices.

In this case, we can assume without loss of generality that $P_H(u) \subseteq P_H(v)$, i.e., all source vertices reaching u also reach v . Note that the set of source vertices remains unchanged after contraction, i.e., $S(\vec{H}') = S(\vec{H})$. We consider the same DAG tree decomposition T of $\vec{H}$. We only have to

check whether the source vertices reaching w are connected in T or not. However, one can see that $P_H(v) = P_{H'}(w)$. So, the source vertices in $P_{H'}(w)$ in T is connected.

- **Case 2:** u is a source vertex.

We consider two subcases depending on whether the new vertex w is a source in $\vec{H}'$ or not.

- **Case A :** w is a source vertex.

This happens only when $P_H(v) = \{u\}$, meaning v was reachable only from u . In this case, $R_H(u) = R_{H'}(w) \setminus \{v\}$. We can construct a DAG tree decomposition T' of $\vec{H}'$ by simply replacing u with w in all the bags of T where u appears. Since no new vertices are introduced and bag sizes remain unchanged, $\text{dtw}(\vec{H}') \leq k$.

- **Case B:** w is a non-source vertex.

Here, the contraction removes u from the set of sources, so $S(\vec{H}') = S(\vec{H}) \setminus \{u\}$. v must be reachable by some source vertex s other than u . Moreover,

$$P_{H'}(w) = P_H(v) \setminus \{u\},$$

and for each source $s \in P_{H'}(w)$,

$$R_{H'}(s) = (R_H(s) \cup R_H(u) \cup \{w\}) \setminus \{v\}.$$

For every bag B where u appears and is the only source from $P_H(v)$, we can replace u with another source from $P_H(v)$ present in an adjacent bag. Also, for any s such that $v \in R(s)$, $R(u) \subseteq R(s)$. Thus, we obtain a decomposition of the DAG tree T' with a maximum bag size k .

Since these modifications only replace one source with another and do not increase the size of any bag, the resulting DAG tree decomposition T' of $\vec{H}'$ has the same width as T , i.e., $\text{dtw}(\vec{H}') \leq k$.

□

4.2 Fast Running Time for $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$: In [BPS21], the authors established that for any pattern H with at most five vertices, one can count H -homomorphisms in bounded-degeneracy graphs in $O(m \log m)$ time. More recently, [PPS24] presented a subquadratic algorithm for counting subgraphs of patterns with up to nine vertices, achieving a running time of $\tilde{O}(n^{5/3})$. They further conjectured that no subquadratic algorithm exists for computing the number of subgraphs for all 10-vertex patterns H .

Building upon these works ([BPS21, PPS24]) we extend the results to patterns with up to 11 vertices and show that they have DAG treewidth 2. Further, we answer the conjecture posed by [PPS24] in the affirmative.

Further in [BPS21], the authors also proved that for a DAG H with k vertices, $\text{dtw}(H) \leq \frac{k}{4} + 2$, and using this obtained upper bounds to compute $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$ (Theorem 28). In this section we improve the bound of $\frac{k}{4} + 2$ to $\frac{k}{5} + 3$ and as a consequence get fast running time for $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$ (Theorem 32).

We now recall the definition of DAG treewidth introduced by Bressan [B21], along with a key theorem that will help us in deriving the results presented in this section.

Definition 19 (Similar to Definition 6 of [B21]) For an undirected pattern graph H , the DAG treewidth of H is $\tau(H) = \tau_3(H)$, where:

$$\begin{aligned}\tau_1(H) &= \max \{ \tau(P) \mid P \in \Sigma(H) \} \\ \tau_2(H) &= \max \{ \tau_1(H/\theta) \mid \theta \in \Theta(H) \} \\ \tau_3(H) &= \max \{ \tau_2(H') \mid H' \in D(H) \}\end{aligned}$$

Here:

- $\Sigma(H)$ is the set of all acyclic orientations of H ,
- $\Theta(H)$ is the set of all equivalence relations (partitions) over V_H , and H/θ is the pattern obtained by identifying equivalent nodes in H according to θ , removing loops and multiple edges,
- $D(H)$ is the set of all supergraphs of H on the same vertex set V_H .

We now state the following important theorem from [B21].

Theorem 28 (Theorem 9 of [B21]) Consider any k -node pattern graph $H = (V_H, E_H)$, and let $f_T(k)$ be an upper bound on the time needed to compute a dag tree decomposition of minimum width on $2^{O(k \log k)}$ bags for any k -node dag. Then one can compute:

- $\text{hom}(H, G)$ in time $2^{O(k \log k)} \cdot O \left(f_T(k) + d^{k-\tau_1(H)} n^{\tau_1(H)} \log n \right)$,
- $\text{sub}(H, G)$ in time $2^{O(k \log k)} \cdot O \left(f_T(k) + d^{k-\tau_2(H)} n^{\tau_2(H)} \log n \right)$,
- $\text{ind}(H, G)$ in time $2^{O(k^2)} \cdot O \left(f_T(k) + d^{k-\tau_3(H)} n^{\tau_3(H)} \log n \right)$.

The theorem still holds if we replace τ_1, τ_2, τ_3 with upper bounds, and $f_T(k)$ with the time needed to compute a dag tree decomposition on $2^{O(k \log k)}$ bags that satisfies those upper bounds.

Given the above preliminaries, we first show that pattern graphs with up to 11 vertices have DAG treewidth 2.

Theorem 29 For any pattern graph H , if $|V(H)| \leq 11$, then $\text{dtw}(\vec{H}) \leq 2$.

Proof If the number of sources is at most 4, then we can split the sources into two bags, each of size at most 2. So, we are interested in the case when the number of sources is at least 5. Also, we assume that $R(s)$ is not contained in $R(s_i)$ for any source s_i , otherwise we can attach a leaf bag containing s to a parent bag containing s_i . Further, we can assume that $|R(s)| \geq 2$, otherwise, it is easy to check that we can get a DAG tree decomposition without increasing the size of the bag.

- **Case 1:** Assume that the number of sources is 5 and the number of non-sources is 6.

We pick a source s such that $|R(s)|$ is minimum.

- **Case A:** If $|R(s)| \leq 2$ then we can take two sources s_i, s_j such that $R(s) \subseteq R(s_i) \cup R(s_j)$. Thus, we can put s in leaf bag B and put $B_1 = \{s_i, s_j\}$ adjacent to B . We put remaining two source vertices in B_2 adjacent to B_1 . It is easy to verify the reachability condition, as we only have to argue about the reachability intersection of s and $R(B_2)$. However, $R(s) \subseteq R(B_1)$. Similarly,

we can argue that whenever there exist sources s such that there exists a pair s_i, s_j , then $R(s) \cap R(s_l) \subseteq R(s_i) \cup R(s_j)$ for any source s_l . Then we can get DAG tree decomposition of bag size at most two.

Now consider the case when $|R(s)| \geq 3$ and there does not exist $\{s_i, s_j\}$ such that $R(s) \subseteq R(s_i) \cup R(s_j)$.

– **Case B:** $|R(s)| = 3$

We know that for any s_i , $|R(s_i) \cap R(s)| \leq 1$, otherwise $R(s) \subseteq R(s_i) \cup R(s_j)$. Therefore, $|R(s_i) \cup R(s)| \geq 5$. If $|R(s_i) \cup R(s)| = 6$ then we are done.

If there is no such s_i then we know that for each source s , $|R(s)| = 3$. So, we consider the case when $|R(s)| = 3$ and $|R(s_i) \cap R(s_j)| \leq 1$ for every pair of source vertices s_i and s_j . However, we require at least seven non-source vertices even to satisfy the condition when the number of sources is four. Thus, it is not possible.

– **Case C:** $|R(s)| = 4$

We know that for any s_i , $|R(s_i) \cap R(s)| \leq 2$. But then $|R(s_i) \cup R(s)| \geq 6$. So we are done.

• **Case 2:** Assume that the number of sources is 6 and the number of non-sources is 5.

– **Case A:** If there exists s_i such that $|R(s_i)| \geq 3$.

If $\exists s_j$ such that $|R(s_i) \cup R(s_j)| = 5$, then we are done. So, $|R(s) \setminus R(s_i)| \leq 1$.

For a non-source x such that $x \notin R(s_i)$, and $|P(x)|$ is at least three (number of sources reaching x), there exists x because there are only two non-sources that are not reachable by s , and there are a total 5 source vertices other than s . We pick a source s , s.t. $x \in R(s)$. We make bag $B_1 = \{s_i, s\}$. The source vertices that are reaching x can be kept as leaf bags of B_1 . The source vertices that are not reaching x is at most two. We keep it in a bag B_2 . This gives a valid DAG tree decomposition.

– **Case B:** $|R(s_i)| \leq 2$

If $|R(s_i)| \leq 2$ then pick two sources s_1 and s_2 such that $|R(s_1) \cap R(s_2)| = 1$. Put s_1 and s_2 in bag B_1 . Now take two other sources s_3 and s_4 such that $|R(s_1) \cap R(s_3)| = 1$ and $|R(s_2) \cap R(s_4)| = 1$ and put s_3 and s_4 in bag B_2 . Now, for the remaining sources s_5 and s_6 , put them in bag B_3 . Now we make DAG tree decomposition where the edges are between $B_1 - B_2 - B_3$.

• **Case 3:** When the number of sources is 7, then the number of non-sources is 4.

If $\exists s$ such that $|R(s)| \geq 3$ then we are done. Therefore $|R(s)| \leq 2$. Now, if $|R(s)| = 1$, then it can be easily handled. So we only consider the case when $|R(s)| = 2$. But in this case, it is not possible to have 7 sources such that $R(s_i) \not\subseteq R(s_j)$. Therefore, we have $|R(s_i) \cup R(s_j)| = 4$. Hence, we are done.

The remaining cases are also similar to the prove. □

As mentioned earlier, [BPS21] first showed that any pattern H with atmost 5 vertices and later [PPS24] showed that any pattern H with 9 vertices, subgraph counting can be performed in time $O(m \log m)$ and $\tilde{O}(n^{5/3})$, respectively. We now extend these results to patterns on up to 11 vertices and show that all such patterns have DAG treewidth at most 2. By applying [Theorem 29](#) together with [Theorem 28](#), we derive the following improved bounds.

Theorem 30 For any pattern graph H we can compute:

- $\text{hom}(H, G)$ in time $O(d^{k-2}n^2 \log n)$,
- $\text{sub}(H, G)$ in time $O(d^{k-2}n^2 \log n)$,
- $\text{ind}(H, G)$ in time $O(d^{k-2}n^2 \log n)$.

Proof From [Theorem 29](#), we know that all the patterns with eleven vertices has dtw at most two.

Given an undirected graph G , we know that the number of vertices of $\text{Spasm}(G)$ is at most the number of vertices in G . Therefore $\tau_2(H) \leq 2$. Similarly, the number of vertices in G is bounded for every supergraph of $\text{Spasm}(G)$. Therefore, $\tau_3(H) \leq 2$. Therefore, using [Theorem 28](#) we get the desired runtime for $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$. $\square$

In [\[PPS24\]](#), apart from the result on 9 vertices, the authors conjectured the following:

Conjecture 3 (Conjecture 1.8 of [\[PPS24\]](#)) There is no $o(m^2)$ algorithm for computing $\text{sub}(\mathcal{H}_\Delta, G)$.

In the above conjecture, $\mathcal{H}_\Delta$ is the same as defined in [\[PPS24\]](#) (See [Figure 3](#)). Thus, [Conjecture 3](#) implies that there are no subquadratic algorithms for computing the number of subgraphs of all 10-vertex patterns H . We now answer this conjecture in the affirmative by showing that no combinatorial algorithm can compute $\text{sub}(H, G)$ in $o(m^2)$ time, when $|V(H)| = 10$.

Proof of [Conjecture 3](#) Our proof relies on the assumption from [\[KKMS23\]](#), which states that there is no combinatorial algorithm of time $o(m^2)$ to count copies of K_4 ([\[KKMS23\]](#)).

We use a strategy similar to that in [\[BPS21\]](#) to prove the hardness of counting subdivision of K_4 patterns.

Let H be a graph on h vertices. We define a coloring of a graph G as $c : V(G) \rightarrow [h]$. A mapping $\phi : V(H) \rightarrow V(G)$ is called a *colorful homomorphism* if

$$\{c(\phi(v)) : v \in V(H)\} = [h].$$

In other words, a colorful homomorphism maps each vertex of H to a distinct color class in G , thereby ensuring injectivity. It is a known fact that counting injective colorful homomorphisms reduces to counting homomorphisms $\text{Hom}(H, G)$ such that the reduction preserves degeneracy (see Lemma 3.1 [\[BPS21\]](#)). Also, similar to Lemma 3.2 of [\[BPS21\]](#), if one can detect colorful K_4 in a 4 vertex colored graph with m edges in time $f(m)$, then it is possible to detect (uncolored) K_4 in a graph with m edges in time $\tilde{O}(f(m))$ via color-coding.

Using all these known facts, we now describe the construction used for this reduction: Given a graph G and a coloring $c : V(G) \rightarrow [4]$, we construct a new graph G' with a coloring $c' : V(G') \rightarrow [10]$ in time $O(|V(G)| + |E(G)|)$, such that

$$|V(G')|, |E(G')| = \mathcal{O}(|V(G)| + |E(G)|).$$

We now claim that the number of colored subdivision K_4 patterns in G' is equal to the number of K_4 subgraphs in G . Furthermore, G' is 2-degenerate.

The construction is as follows: Since G is colored with 4 colors, we assign one of 6 new colors to each subdivision vertex in G' . For every edge $\{x, y\} \in E(G)$, if $c(x) \neq c(y)$, we subdivide the edge by introducing a new vertex with color $(c(x), c(y))$. If $c(x) = c(y)$, the edge is deleted. Note that the original vertices in $V(G)$ retain colors from $\{1, 2, 3, 4\}$, while subdivision vertices are colored with

the remaining 6 colors. Clearly, $|V(G')| \leq |V(G)| + |E(G)|$ and $|E(G')| \leq 2|E(G)|$ both remain in $O(|V(G)| + |E(G)|)$.

One can observe that the original vertices $V(G)$ form an independent set in G' , and all subdivision vertices have degree exactly 2 in G' . After removing the subdivision vertices, the resulting graph is empty. Hence, G' is 2-degenerate. Also, from the construction, it is straightforward to verify that the number of colorful K_4 copies in G equals the number of colorful subdivision K_4 copies in G' .

Now, suppose that homomorphism counting for subdivision K_4 can be performed in $o((n^2))$ time on 2-degenerate n -vertex graphs. Then, one can also count colored injective homomorphisms of subdivision K_4 in $o((n^2))$ time on 2-degenerate, k -vertex-colored, n -vertex graphs.

Using the above construction, one can then detect colorful K_4 in 4-vertex-colored, m -edge graphs in $o(m^2)$ time. Consequently, this yields an $\tilde{o}(m^2)$ -time algorithm for detecting (uncolored) K_4 in general m -edge graphs.

However, it is known that $\text{mtw}(K_4) = 4$ and no combinatorial algorithm exists to detect K_4 in $o(m^2)$ time under standard complexity assumptions ([KKMS23]). Therefore, subdivision K_4 subgraph counting cannot be done in $o(m^2)$ time. This further implies that there is no $o(m^2)$ algorithm for computing $\text{sub}(H, G)$, when $V(H) = 10$.

□

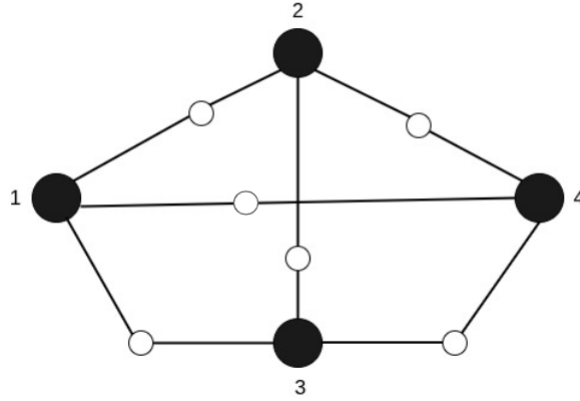


Figure 3: k_4 with one subdivision

We now derive the following lemma, which will help us to show that for a DAG with k vertices, $\text{dtw}(H) \leq \frac{k}{5} + 4$. We then do further refinement to show that $\text{dtw}(H) \leq \frac{k}{5} + 3$.

Lemma 12 *Given a DAG H with t source vertices such that the reachability of every source vertex is bounded by 2, we have $\text{dtw}(H) \leq \frac{t}{5.217} + 4$.*

Proof We construct a graph G where each vertex corresponds to a non-source vertex in H . For every source vertex s in H such that $R(s) = \{u, v\}$, we add the edge $\{u, v\}$ in G . Thus, the number of edges in G is equal to the number of source vertices in H , i.e., $|E(G)| = t$.

From the result of [KMRR05], we know that any graph with m edges has treewidth at most $\frac{m}{5.217} + 3$. Hence, $\text{tw}(G) \leq \frac{t}{5.217} + 3$, and there exists a tree decomposition T of G with maximum bag size at most $\frac{t}{5.217} + 4$.

We now use T to construct a DAG tree decomposition T' of H . For every non-source vertex u in a bag of T , we replace it by the source vertex s such that $u \in R(s)$. Since $|R(s)| = 2$ for every s , this replacement is well-defined and unique. After this replacement, for every source vertex s , either s appears in some bag, or $R(s)$ is contained in a single bag. In the latter case, we can add a leaf bag B_s

containing only s as a child of the bag that contains $R(s)$. This ensures the reachability condition of DAG tree decomposition is satisfied.

Thus, the resulting DAG tree decomposition T' has maximum bag size at most $\frac{t}{5.217} + 4$, and we conclude that $dtw(H) \leq \frac{t}{5.217} + 4$. $\square$

Theorem 31 *For any DAG H with k vertices, we have $dtw(H) \leq \frac{k}{5} + 4$.*

Proof We construct a DAG tree decomposition for H with maximum bag size at most $\frac{k}{5} + 4$.

The construction has two phases. In the first phase, we build a set of source vertices through which we can discover at least four new non-sources. In the second phase, we build a DAG tree decomposition for the rest of the sources. Finally, we combine these two to build a DAG tree decomposition for H .

First phase: Initialize B^* , a subset of sources of H , to the empty set. We greedily add sources to B^* as follows: at each step, if there is a source vertex s such that $|R(s) \setminus R(B^*)| \geq 4$. We mark s and the non-sources reachable from s as SEEN. At least five new vertices are marked by this step. We add s to B^* and remove all SEEN vertices from H . Then, we repeat this process. If there is no source vertex s such that $|R(s) \setminus R(B^*)| \geq 4$, then we stop this phase. Let the subgraph of H at the end of this phase be H' and let k' be the number of vertices in H' .

Second phase: Our goal now is to construct a DAG tree decomposition T' of H' with maximum bag size at most $\frac{k'}{5} + 4$. In the graph H' , every source vertex has at most three reachable vertices, that is, $|R(s)| \leq 3$. We now apply the construction in the first phase again to build a set of sources B_1 (starting from the empty set) by adding iteratively sources s such that $|R(s) \setminus R(B_1)| = 3$ while removing all vertices reachable from s at each step as before. Let H_1 be the resulting subgraph of H' . Now, we split the proof into cases:

- $|B_1| = 0$: By [Lemma 12](#), we have $dtw(H') \leq \frac{k'}{5.217} + 4$.
- $|B_1| = \frac{k'}{5}$: We have removed at least $4(\frac{k'}{5})$ vertices while building B_1 . Therefore, at most $\frac{k'}{5}$ sources remain. Let B_2 be the set of remaining sources. The DAG tree decomposition of H' is a tree that is an edge. One vertex is the bag B_1 and the other is bag B_2 . This decomposition has maximum bag size $\frac{k'}{5}$.
- $0 < |B_1| < \frac{k'}{5}$: Let t be the number of non-source vertices in H_1 . If $t \leq \frac{k'}{5}$, we build a bag B' that includes one source for each of the t non-source vertices. Any remaining sources are attached to the bag B' as singleton leaf bags. Finally bag B_1 is attached to B' . The maximum bag size remains at most $\frac{k'}{5}$. If $t > \frac{k'}{5}$, then the number of sources in H_1 is less than $\frac{4k'}{5}$. Let G_1 be the induced subgraph of H' on $B_1 \cup R(B_1)$ and let k_1 be the number of vertices in G_1 . Then, G_1 has $\frac{k_1}{4}$ sources and $\frac{3k_1}{4}$ non-sources. Let G_2 be the graph obtained by removing $V(G_1)$ from H' . In G_2 , each source has at most two reachable vertices, and the number of sources is less than $\frac{4k'}{5} - \frac{k_1}{4}$. Applying [Lemma 12](#) gives:

$$dtw(G_2) \leq \frac{1}{5.217} \left(\frac{4k'}{5} - \frac{k_1}{4} \right) + 4.$$

We then form the decomposition of H' by adding B_1 to each bag in the decomposition of G_2 . So, the maximum bag size is:

$$dtw(H') \leq \frac{4k'}{25} - \frac{k_1}{20} + \frac{k_1}{4} + 4 = \frac{4k'}{25} + \frac{k_1}{5} + 4.$$

If $k_1 > \frac{k'}{5}$, then the number of source vertices in H' is at most $\frac{2k'}{5}$. Thus, we can get DAG tree decomposition for H' with maximum bag size $\frac{k'}{5}$. So $k_1 \leq \frac{k'}{5}$ and therefore $\frac{4k'}{25} + \frac{k_1}{5} + 4 \leq \frac{k'}{5} + 4$.

Therefore, in all cases, at the end of the second phase, we have a DAG tree decomposition for H' with maximum bag size at most $k'/5 + 4$.

Finally, by adding B^* from the first phase to every bag of the decomposition of H' obtained in the second phase, we obtain a DAG tree decomposition of H . The maximum bag size of this DAG tree decomposition is: $|B^*| + \frac{k'}{5} + 4 \leq \frac{(k-k')}{5} + \frac{k'}{5} + 4 = \frac{k}{5} + 4$. $\square$

Corollary 3 *The additive term $+4$ arises only when using [Lemma 12](#), which divides by 5.217. If we divide by 5, so in practice, using $+3$ is sufficient. Hence, $\text{dtw}(H) \leq \frac{k}{5} + 3$ for all DAGs with k vertices.*

We now use [Corollary 3](#) and [Theorem 28](#) to have the following result.

Theorem 32 *Consider any k -node pattern graph $H = (V_H, E_H)$. Then one can compute:*

- $\text{hom}(H, G)$ in time $2^{O(k \log k)} \cdot O(d^{k - \lfloor \frac{k}{5} \rfloor - 3} n^{\lfloor \frac{k}{5} \rfloor + 3} \log n)$,
- $\text{sub}(H, G)$ in time $2^{O(k \log k)} \cdot O(d^{k - \lfloor \frac{k}{5} \rfloor - 3} n^{\lfloor \frac{k}{5} \rfloor + 3} \log n)$,
- $\text{ind}(H, G)$ in time $2^{O(k^2)} \cdot O(d^{k - \lfloor \frac{k}{5} \rfloor - 3} n^{\lfloor \frac{k}{5} \rfloor + 3} \log n)$.

This improves the previous known bound for counting induced subgraphs of k -vertex pattern of [B21]. In [B21], the author improved the exponent of n from $0.791k + 2$ ([NP85]) to $0.25k + 2$. In [Corollary 3](#), we show that the exponent of n is bounded by $0.2k + 3$. Let r be the average degree of G . Since we know that $d = O(\sqrt{rn})$ as a corollary of [Theorem 32](#) we have the following theorem:

Theorem 33 *Consider any k -node pattern graph $H = (V_H, E_H)$ and let r be the average degree of G . Then one can compute:*

- $\text{hom}(H, G)$ in time $2^{O(k \log k)} \cdot O(r^{\frac{1}{2}(k - \lfloor \frac{k}{4} \rfloor - 2)} n^{\frac{1}{2}(k + \lfloor \frac{k}{4} \rfloor + 2)} \log n)$,
- $\text{sub}(H, G)$ in time $2^{O(k \log k)} \cdot O(r^{\frac{1}{2}(k - \lfloor \frac{k}{4} \rfloor - 2)} n^{\frac{1}{2}(k + \lfloor \frac{k}{4} \rfloor + 2)} \log n)$,
- $\text{ind}(H, G)$ in time $2^{O(k^2)} \cdot O(r^{\frac{1}{2}(k - \lfloor \frac{k}{4} \rfloor - 2)} n^{\frac{1}{2}(k + \lfloor \frac{k}{4} \rfloor + 2)} \log n)$.

References

- [DST02] Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting H-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002. [doi:10.1016/S0304-3975\(02\)00017-8](#).
- [Kn74] Donald E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17(12):667–673, 1974. [doi:10.1145/361604.361612](#).
- [Di68] Edsger W. Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968. [doi:10.1145/362929.362947](#).
- [GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [HPV75] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On Time versus Space and Related Problems. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 57–64. IEEE, 1975. [doi:10.1109/SFCS.1975.23](#).

- [FPT08] Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Approximating acyclicity parameters of sparse hypergraphs. *arXiv preprint arXiv:0809.3646*, 2008.
- [BGM23] Marco Bressan, Leslie Ann Goldberg, Kitty Meeks, and Marc Roth. Counting subgraphs in somewhere dense graphs. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- [GLY23] Lior Gishboliner, Yevgeny Levanzov, Asaf Shapira, and Raphael Yuster. Counting homomorphic cycles in degenerate graphs. *ACM Transactions on Algorithms*, 19(1):1–22, 2023.
- [BGLS22] Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *J. ACM*, 69(3):1–21, 2022. doi:10.1145/3520240.
- [PPS24] Daniel Paul-Pena and C. Seshadhri. Subgraph counting in subquadratic time for bounded degeneracy graphs. *arXiv preprint arXiv:2410.08376*, 2024.
- [PPS25] Daniel Paul-Pena and C. Seshadhri. A dichotomy hierarchy for linear time subgraph counting in bounded degeneracy graphs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 48–87. SIAM, 2025.
- [BLR23] Marco Bressan, Matthias Lanzinger, and Marc Roth. The complexity of pattern counting in directed graphs, parameterised by the outdegree. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 542–552, 2023.
- [BPS21] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2315–2332. SIAM, 2021.
- [BR22] Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285. IEEE, 2022.
- [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.
- [KKMS23] Balagopal Komarath, Anant Kumar, Suchismita Mishra, and Aditi Sethia. Finding and counting patterns in sparse graphs. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, pages 40–1, 2023.
- [B21] Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021.
- [KMRR05] Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Algorithms based on the treewidth of sparse graphs. In *Graph-Theoretic Concepts in Computer Science: 31st International Workshop, WG 2005*, pages 385–396. Springer, 2005.
- [AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- [CDM17] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of STOC 2017*, pages 210–223, 2017. doi:10.1145/3055399.3055502.
- [B25] Édouard Bonnet. Treewidth inapproximability and tight eth lower bound. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 2130–2135, 2025.
- [NP85] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- [CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.
- [Epp94] David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207–211, 1994.
- [BPS19] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. *arXiv preprint arXiv:1911.05896*, 2019.
- [LPSXZ25] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Efficiently finding and counting patterns with distance constraints in sparse graphs. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1965–1974, 2025.
- [GLYS22] Lior Gishboliner, Yevgeny Levanzov, Asaf Shapira, and Raphael Yuster. Counting homomorphic cycles in degenerate graphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 417–430, 2022.

- [We01] Douglas Brent West et al. *Introduction to Graph Theory*, volume 2. Prentice Hall Upper Saddle River, 2001.
- [Ses23] C. Seshadhri. Some vignettes on subgraph counting using graph orientations (invited talk). In *26th International Conference on Database Theory (ICDT 2023)*, pages 3–1, 2023.
- [FG04] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.
- [Lov12] László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Society, 2012.
- [ANRD15] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*, pages 1–10. IEEE, 2015.
- [CDM17b] 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.
- [DK25] Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *Logical Methods in Computer Science*, 21, 2025.
- [KNOZ25] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Conjunctive queries with free access patterns under updates. *Logical Methods in Computer Science*, 21, 2025.
- [PS20] Noujan Pashanasangi and C. Seshadhri. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 447–455, 2020.
- [JSP15] Madhav Jha, C. Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web*, pages 495–505, 2015.
- [PPS22] Daniel Paul-Pena and C. Seshadhri. A dichotomy theorem for linear time homomorphism orbit counting in bounded degeneracy graphs. *arXiv preprint arXiv:2211.08605*, 2022.
- [BBDRW25] Marco Bressan, Julian Brinkmann, Holger Dell, Marc Roth, and Philip Wellnitz. The Complexity of Counting Small Sub-Hypergraphs. *arXiv preprint arXiv:2506.14081*, 2025.
- [PPS25b] Daniel Paul-Pena and C. Seshadhri. Near-linear time subhypergraph counting in bounded degeneracy hypergraphs. *arXiv preprint arXiv:2510.16330*, 2025.
- [FGP18] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 17–32, 2018.