Counting Patterns in Degenerate Graphs in Constant Space

Balagopal Komarath
Department of Computer Science, IIT Gandhinagar, India

Anant Kumar
Department of Computer Science, IIT Gandhinagar, India

Akash Pareek
Department of Computer Science, Université Libre de Bruxelles, Belgium

Abstract

For a fixed pattern graph, we study the algorithmic complexity of counting homomorphisms, subgraph isomorphisms, and induced subgraph isomorphisms into an n-vertex, d-degenerate host graph. Bressan (Algorithmica, 2021) introduced the notion of DAG treewidth and showed that counting homomorphisms and induced subgraphs can be performed efficiently using dynamic programming that requires polynomial space. In this work, we introduce a new graph parameter, called *DAG treedepth*, which enables efficient divide-and-conquer algorithms for counting homomorphisms in d-degenerate host graphs using only constant space.

Bera, Gishboliner, Levanzov, Seshadhri, and Shapira (SODA, 2021) showed that a pattern graph has DAG treewidth one if and only if it contains no induced cycle of length at least six. This induced minor characterization leads to linear-time and linear-space algorithms. Building on this line of work, we derive an induced-minor characterization of graphs with DAG treedepth at most two that uses only constant space.

Recently, Paul-Pena and Seshadhri (ICALP, 2025) proved that all pattern graphs on at most nine vertices can be counted in subquadratic time using polynomial space. We show that every pattern graph on at most nine vertices can be counted as an induced subgraph in $O(n^3)$ time using only constant space. Moreover, we show that patterns on at most eleven vertices can be counted in $O(n^2)$ time using polynomial space.

Finally, we present a constant-space algorithm for counting induced subgraphs that matches the running time of Bressan's algorithm. We further show that, when polynomial space is allowed, homomorphisms, subgraph isomorphisms, and induced subgraph isomorphisms can be counted faster than Bressan's algorithm. In addition, we establish several other results related to DAG treewidth and DAG treedepth that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Homomorphism Counting, Subgraph Counting, Induced subgraph Counting, Bounded degeneracy graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.28

Related Version *Full Version*: https://arxiv.org/abs/2511.04258

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 $\{(V', E') \mid (V', E') \cong H, V' \subseteq V(G), E' \subseteq E(G)\}$.
2. The number of induced subgraphs of G isomorphic to H. That is, the number of vertex subsets $S \subseteq V(G)$ such that the induced subgraph $G[S] = (S, \{\{u, v\} \in E(G) \mid u, v \in S\})$ is isomorphic to H.

3. The number of homomorphisms from H to G. That is, the number of functions $\phi : V(H) \to V(G)$ such that for every edge $\{u, v\} \in E(H)$, we have $\{\phi(u), \phi(v)\} \in E(G)$.

Counting subgraphs and homomorphisms is at the heart of many problems in combinatorics, database theory, and network analysis [15, 22, 1, 12, 13, 19]. These problems capture fundamental questions such as detecting specific patterns in large networks and 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 [14, 20].

In many real-world settings, host graphs are not arbitrary but exhibit additional structure such as sparsity or bounded degeneracy [23, 18]. 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 [28]. Studying counting problems on degenerate graphs leads to more refined algorithmic bounds that better reflect the structure of real data.

Existing work on counting patterns in bounded-degeneracy graphs has primarily focused on improving *time* complexity. By exploiting acyclic orientations of the host graph with bounded outdegree, Bressan [7] introduced the notion of *DAG treewidth* and obtained faster algorithms for counting homomorphisms and induced subgraphs via dynamic programming. This approach has led to a rich theory of (near) linear-time algorithms and structural characterizations for restricted pattern classes [5, 11, 3, 24, 25].

In contrast, the *space complexity* of pattern counting in bounded-degeneracy graphs has received little attention. Dynamic programming techniques underlying existing algorithms inherently require maintaining tables whose size grows with the width of the decomposition, and therefore do not extend naturally to the constant-space setting. This motivates the search for structural parameters that support divide-and-conquer algorithms with bounded recursion depth.

In this work, we introduce a new graph parameter, called *DAG treedepth* (dtd), which can be viewed as a treedepth analogue of DAG treewidth. The DAG treedepth parameter captures the reachability structure induced by an acyclic orientation of the pattern graphs. We show that this parameter precisely governs the complexity of counting homomorphisms, subgraphs, and induced subgraphs in bounded-degeneracy graphs. The algorithms use only constant space, assuming that both the degeneracy and the dtd are constants.

1.1 Our Results

We begin by introducing a new structural parameter, called the *DAG treedepth* (see Definition 19), 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. Our first main result shows that this parameter precisely governs the complexity of counting homomorphisms using only constant space.

▶ **Theorem 1.** *Let d be a fixed constant. If the DAG treedepth of a k-vertex graph H is t, then the number of homomorphisms from H to an n-vertex, d-degenerate host graph G can be counted in $O(n^t)$ time using constant space.*

We further show that the DAG-treedepth framework extends beyond homomorphism counting. In particular, it yields constant-space algorithms for counting induced subgraphs in bounded-degeneracy graphs whose running times asymptotically match the best known bounds obtained via DAG treewidth [7]. Crucially, our algorithms avoid the polynomial-space overhead inherent in dynamic programming.

▶ **Theorem 2.** *Let d be a fixed constant and let H be a k-vertex graph. Then the number of induced subgraphs of an n-vertex, d-degenerate graph G that are isomorphic to H can be counted in $O(n^{k/4+O(1)})$ time using constant space.*

We also show that our techniques yield faster constant-space algorithms for counting subgraphs when the pattern graph is sufficiently sparse, with running time depending on the number of edges rather than the number of vertices. Additionally, in Theorem 2, we show that, instead of constant space, if we allow polynomial space, we can improve the bound from $O(n^{k/4+O(1)})$ to $O(n^{k/5+O(1)})$.

▶ **Theorem 3.** *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.*

While DAG treewidth has proved effective for obtaining linear-time algorithms on bounded-degeneracy graphs, its structural complexity quickly becomes intractable beyond this regime. In particular, as we show, even deciding whether a directed acyclic graph has DAG treewidth at most two is NP-complete. Therefore, DAG treewidth does not admit induced minor characterizations even in the quadratic-time regime. In contrast, DAG treedepth admits a significantly more tractable structural theory. In particular, for every fixed constant k, it can be decided in polynomial time whether a given directed acyclic graph has DAG treedepth at most k (see Theorem 22).

Bera, Pashanasangi, and Seshadhri [4] showed that a pattern graph H has DAG treewidth one if and only if it has no induced cycle of length 6 or greater. Inspired by the induced-minor characterization of patterns admitting linear-time and linear-space algorithms [4], we develop an analogous framework for DAG treedepth. This framework enables both constant-space algorithms and structural characterizations of tractable pattern classes. In particular, we obtain a complete induced-minor characterization of patterns that admit cubic-time, constant-space algorithms.

▶ **Theorem 4.** *A pattern 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).*

More generally, we investigate the structural relationship between induced-minor–based characterizations of bounded DAG treedepth and the classical minor-based characterizations of bounded treedepth (see Lemma 29 and Lemma 30). We show that, for every fixed integer k, the class of graphs of DAG treedepth less than k admits an induced-minor characterization that is directly related to the minor obstruction set for treedepth and their supergraphs.

We further complement this structural connection with conditional lower bounds, relating the complexity of algorithms parameterized by DAG treedepth to that of algorithms parameterized by treedepth. In particular, improving the running time of constant-space algorithms for patterns of DAG treedepth three would imply a corresponding breakthrough for classical pattern-counting problems.

▶ **Theorem 5.** *For every $\varepsilon > 0$, there is no $O(n^{3-\varepsilon})$-time constant-space algorithm for counting subgraphs isomorphic to any pattern H with $dtd(H) = 3$ in bounded-degeneracy graphs of degeneracy two, unless there exists an $O(n^{3-\varepsilon})$-time constant-space algorithm for counting triangles in arbitrary graphs.*

Note that subcubic-time algorithms for triangle counting based on fast matrix multiplication are known. However, such algorithms inherently require polynomial space and therefore fall outside the constant-space setting considered here.

Several graph patterns can be counted efficiently when the host graph has bounded degeneracy. Bressan established efficient counting results for certain special classes of patterns, including K_n, $K_n - \{e\}$, and K_{k_1,k_2}. In [4], the authors showed that all patterns on at most five vertices can be counted in nearly linear time using linear space. More recently, Paul-Pena and Seshadhri [26] proved that all patterns on at most nine vertices can be counted in subquadratic time using polynomial space. To this end, we show the following:

▶ **Theorem 6.** *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 further show that the bound of nine vertices in the above theorem is tight by establishing a conditional lower bound for a specific pattern on ten vertices.

Paul-Pena and Seshadhri [26] also identified a specific ten-vertex pattern, denoted $\mathcal{H}_\Delta$, and conjectured that it does not admit a sub-quadratic time polynomial space algorithm.

▶ **Conjecture 7** (Conjecture 1.8 of [26])**.** *For any $\varepsilon > 0$, there is no $O(m^{2-\varepsilon})$-time* combinatorial *algorithm for computing* $\mathrm{sub}(\mathcal{H}_\Delta, G)$.

We resolve this conjecture affirmatively in the combinatorial setting by assuming that counting K_4-copies needs (for combinatorial algorithms) quadratic time in the number of edges.

Beyond resolving this conjecture, we provide a structural explanation for the observed nine-vertex barrier. We show that every pattern graph on at most eleven vertices admits an acyclic orientation whose DAG treewidth is at most two. We also show that this bound is tight by providing a graph on twelve vertices whose DAG treewidth is three.

▶ **Theorem 8.** *Let d be a constant. For every pattern graph H with at most eleven vertices, the number of induced subgraphs of an n-vertex d-degenerate graph G that are isomorphic to H can be counted in $O(n^2)$ time using polynomial space.*

We further establish a relation between DAG treedepth and treedepth of the pattern graph, as well as DAG treewidth of the pattern graph. In addition, we show many properties and bounds related to DAG treewidth and DAG treedepth, 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, along with complexity dichotomies. A recent dichotomy hierarchy precisely characterizes linear-time subgraph counting in such graphs [25]. In the dense graph regime, Bressan et al. [5] classified pattern counting complexity using DAG-treewidth, providing a sharp dichotomy for somewhere dense graph classes [8]. At the same time, related work has addressed the complexity of counting homomorphic cycles in degenerate graphs [17], and pattern counting in directed graphs parameterized by outdegree [10, 11]. Recent studies have also considered counting patterns when the host graph is sparse [20]. For approximate counting, several works have proposed efficient approximation algorithms for subgraph

counting problems, including a general approximation framework for sparse graphs [21], and fast approximation algorithms specifically for triangle counting in streaming and distributed settings [6]. Beyond graphs, subhypergraph counting has recently gained traction. In [9], the authors extended the homomorphism basis framework to hypergraphs, while [27] 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 the necessary preliminaries, introducing the definitions, notation, and background used throughout the paper. In Section 3, we define DAG treedepth and prove Theorem 1. We then identify the obstructions for DAG treedepth 0, 1, and 2 in Section 3.1. Subsequently, in Section 3.2, we establish conditional lower bounds for counting patterns in cubic time using constant space.

In the full version of the paper, Appendix D presents constant-space algorithms for computing $\hom(H, G)$, $\mathrm{sub}(H, G)$, and $\mathrm{ind}(H, G)$. We then shift our focus to DAG treewidth in Appendix E. In Appendix E.1, we present key properties of DAG treewidth, which are subsequently used in Appendix E.2 to design faster algorithms for computing $\hom(H, G)$, $\mathrm{sub}(H, G)$, and $\mathrm{ind}(H, G)$, along with several additional results. Finally, in Appendix F, we establish relationships between DAG treedepth and treedepth, as well as between DAG treewidth and treewidth. Proofs omitted from the main text are deferred to the full version.

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 [29]. We use the following standard notations for some well-known graphs: e or (uv) for edges in a graph, P_k for k-vertex paths, C_k for k-cycles, K_k for k-cliques, $K_k - e$ for a 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 $v_1, \ldots, v_k$ 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 9** (Graph Homomorphism). *Given two graphs G and H, a graph homomorphism from G to H is a map $\phi : V(G) \to V(H)$ 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 10** (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 11** (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* $\mathrm{tw}(G)$, *is the minimum width over all possible tree decompositions of G.*

▶ **Definition 12** (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 13** (Treedepth). *The* treedepth *of a graph G, denoted as* $\mathsf{td}(G)$, *is the minimum depth among all possible elimination trees of G.*

▶ **Definition 14** (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 15** (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 $\mathrm{hom}(\mathrm{H}, \mathrm{G})$, number of subgraphs of H in G by $\mathrm{sub}(H, G)$, and number of induced subgraph of H in G by $\mathrm{ind}(H, G)$.

▶ **Definition 16** (Spasm). *Let H be a graph. If H is a clique, then* $\mathsf{Spasm}(H) = \{H\}$. *Otherwise, let $\mathcal{I}$ denote the set of all independent sets of H of size at least two. For any $I \in \mathcal{I}$, let* $\mathsf{Merge}(H, I)$ *be the graph obtained by merging all vertices in I into a single vertex. Then,* $\mathsf{Spasm}(H) = \{H\} \cup \bigcup_{I \in \mathcal{I}} \mathsf{Spasm}(\mathsf{Merge}(H, I))$.

Note that for any fixed pattern graph H, the number of subgraph isomorphisms from H to a host graph G can be expressed as a linear combination of homomorphism counts from graphs in $\mathsf{Spasm}(H)$ to G. More precisely, there exist constants $\{\alpha_{H'}\}_{H' \in \mathsf{Spasm}(H)}$, such that $\mathrm{sub}(H, G) = \sum_{H' \in \mathsf{Spasm}(H)} \alpha_{H'} \mathrm{hom}(H', G)$. This identity is a key tool in the literature for counting subgraphs (see, for example, [2, 12])

▶ **Definition 17** (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 $(u \to v)$ or $(v \to u)$, 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 in-degree 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 $\mathrm{R}_H(s)$ denote the set of all vertices reachable from s in $\vec{H}$. More generally, for a set of sources $B = \{s_1, s_2, \ldots, s_\ell\} \subseteq S$, we define, $\mathrm{R}_H(B) = \bigcup_{i=1}^{\ell} \mathrm{R}_H(s_i)$. When the underlying graph is clear from the context, we define the set of reachable vertices from s as $\mathrm{R}(s)$.

▶ **Definition 18** (Subdivision Vertex). *Let $G = (V, E)$ be an undirected graph and let $\{u, v\} \in E$. The graph obtained after subdividing the edge $\{u, v\}$ is the graph $G' = (V \cup \{w\}, (E \setminus \{\{u, v\}\}) \cup \{\{u, w\}, \{w, v\}\})$, where w is a new vertex not in V. The vertex w is called a* subdivision vertex.

In [7], Bressan described a way to construct a bipartite graph $\textsc{Bip}(\vec{H})$ 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 $\textsc{Bip}(\vec{H})$

Given a directed acyclic graph $\vec{H}$ with a fixed acyclic orientation, Bressan [7] constructed a bipartite graph $\textsc{Bip}(\vec{H})$ as follows:

- The vertex set of $\textsc{Bip}(\vec{H})$ denoted as $V(\textsc{Bip}(\vec{H})) = (S, V(H) \setminus S)$, where S is the set of sources in $\vec{H}$.
- An edge $\{u, v\} \in E(\textsc{Bip}(\vec{H}))$ if and only if $v \in \mathrm{R}_H(u)$, i.e., v is reachable from the source vertex u in $\vec{H}$.

Here, every edge in $\textsc{Bip}(\vec{H})$ has one of its endpoints as a source vertex and another endpoint as a non-source vertex. Now, we perform edge contraction. Note that one non-source vertex may be reachable from several sources and therefore have several options for edge contraction. We sequentially contract exactly one edge for each non-source and get the resultant graph. Note that the vertices of the resultant graph are exactly the set of source vertices. The resulting graph may differ depending on the choice of edge contraction. 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.

2.2 Model of Computation

The model of computation that we consider is the unit-cost RAM model. In particular, we can store labels of vertices and edges of the host graph in a constant number of words[1]. In this model, algorithms based on fast matrix multiplication and/or treewidth mentioned above use polynomial space. However, the brute-force search algorithm uses only constant space as it only needs to store k vertex labels at a time (Recall that we regard k as a constant).

3 DAG Treedepth

In [7], Bressan introduced DAG treewidth (dtw), a structural parameter analogous to treewidth, and used it to design efficient algorithms for computing $\hom(H, G)$, $\mathrm{sub}(H, G)$, and $\mathrm{ind}(H, G)$. 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, 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 prove Theorem 1. In Section 3.1, we identify the obstruction sets for DAG treedepth 0, 1, and 2. In Section 3.2 we present conditional lower bounds for pattern counting and a few results on small pattern counting.

[1] In the TM model or the log-cost RAM model, storing labels of vertices would take $O(\log n)$ space.

In the full version of the paper, we compute $\hom(H, G)$, $\mathrm{sub}(H, G)$, and $\mathrm{ind}(H, G)$ using constant space, improving the space complexity of the fast algorithms of [7] from polynomial to constant. We also provide upper bounds using the number of vertices and the number of edges of the pattern graph H.

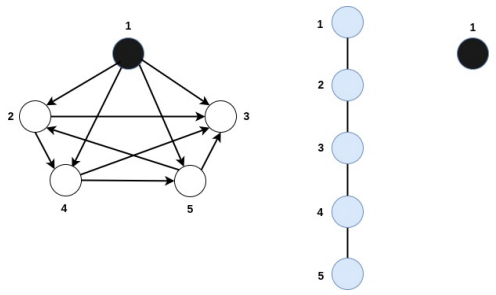
▶ **Definition 19.** *Let $\vec{H}$ be any DAG. A* DAG elimination forest *of $\vec{H}$ is defined as a collection of rooted trees, constructed recursively as follows:*

- *If $\vec{H}$ is empty, the forest is empty.*
- *If the underlying undirected graph of $\vec{H}$ is disconnected, then construct DAG elimination trees for each connected component, and their union is a DAG elimination forest for $\vec{H}$.*
- *Otherwise, $\vec{H}$ has at least 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 $\vec{H}$. The root of the tree is s, and its sub-trees are the trees in the DAG elimination forest for the remaining DAG.*

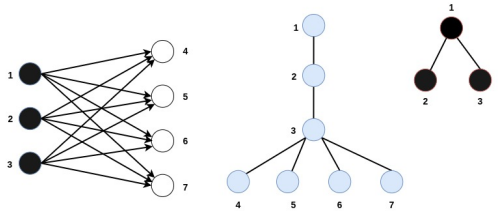
The DAG treedepth *of a DAG $\vec{H}$ is defined as the minimum depth of any elimination forest constructed in the above fashion. For an undirected graph H, its DAG treedepth is defined as the maximum DAG treedepth taken over all acyclic orientations of $\vec{H}$.*

▶ Remark 20. 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 DAG's sources.

▶ **Observation 21.** *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 1.** *Let d be a fixed constant. If the DAG treedepth of a k-vertex graph H is t, then the number of homomorphisms from H to an n-vertex, d-degenerate host graph G can be counted in $O(n^t)$ time using constant space.*

Proof. See Algorithm 1. The required count is obtained by `COUNT-HOM(G, T, r, H, {})`, where T is the DAG elimination tree of depth t and r is the root of 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 ← 0
2:  for all u ∈ V(G) do
3:      p ← 0
4:      for all σ' extending σ to {v} ∪ R(v) such that σ'(v) = u do
5:          p ← 1
6:          for all children w of v in T do
7:              p ← p × COUNT-HOM(G, T, w, H, σ')
8:          end for
9:      end for
10:     c ← c + p
11: end for
12: return c
```

We claim that `COUNT-HOM(G, T, v, H, σ)` 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 possible mappings v. Any non-source reachable from v is already mapped in σ or is 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 these 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, 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. The length of any simple path from v in H is bounded by k. The number of iterations of the 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 $O(.)$ 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. ◀

Having established that bounded DAG treedepth enables constant-time homomorphism counting, a natural algorithmic question arises: given a directed graph G, how can one determine its DAG treedepth? In particular, since our main result assumes the parameter as part of the input, it is essential to understand the computational complexity of detecting and computing this parameter. We therefore study the problem of deciding whether a graph has DAG treedepth at most k, and of constructing a corresponding decomposition when it exists.

▶ **Theorem 22.** *Given a DAG H, it can be verified in $O(g(k)n^{f(k)})$ time whether $dtd(H) \leq k$ or not, where $f(k)$ and $g(k)$ are some computable functions.*

Proof. We can design an XP algorithm to check whether, for a given DAG H, $\mathrm{dtd}(H) \leq k$. 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 $H'_1, H'_2, \ldots, H'_r$, 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.

The 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 $\mathrm{dtd}(H) \leq k$.

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 $O(g(k)n^{f(k)})$. Hence, the algorithm runs in XP time with respect to parameter k. ◀

Next, we analyze the structure of the DAG elimination tree and derive a structural constraint on the sources. Intuitively, if two sources have a unique common reachable vertex, then any elimination tree must reflect this dependency by placing them along a common root-to-leaf path. The following lemma formalizes this observation.

▶ **Lemma 23.** *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 $w \in \mathrm{R}_H(u) \cap \mathrm{R}_H(v)$, and $w \notin \mathrm{R}_H(s)$ 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.*

▶ Remark 24. 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 25.** *If I is an induced minor of H, then $dtd(H) \geq dtd(I)$.*

3.1 DAG Treedepth Obstruction

In this section, we obtain a complete induced-minor characterization of patterns with DAG treedepth at most 2. This helps in identifying exactly those patterns that admit cubic-time, constant-space algorithms. In addition, we show that, for every fixed integer k, the class of graphs of DAG treedepth less than k admits an induced-minor characterization that is directly related to the minor obstruction set for treedepth and their supergraphs.

▶ **Definition 26** (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, $u \in R(\textsc{Parent}(s)) \cap R(T(s))$ and $u \notin R(\text{ancestor of } \textsc{Parent}(s))$.*

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 27.** *Let H be a DAG with $dtd(H) = d$. Then, there exists a reduced DAG elimination tree of H with maximum depth d.*

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

Require: A directed acyclic graph $H = (V, E)$ and an integer $k \geq 1$
Ensure: Returns **true** if $\mathrm{dtd}(H) \leq k$, 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, ..., H'_r} be the connected components of H'
13:            valid ← true
14:            for each component H'_i do
15:                if CheckDTD(H'_i, k − 1) is false then
16:                    valid ← false
17:                    break
18:                end if
19:            end for
20:            if valid then
21:                return true
22:            end if
23:        else
24:            if 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
```

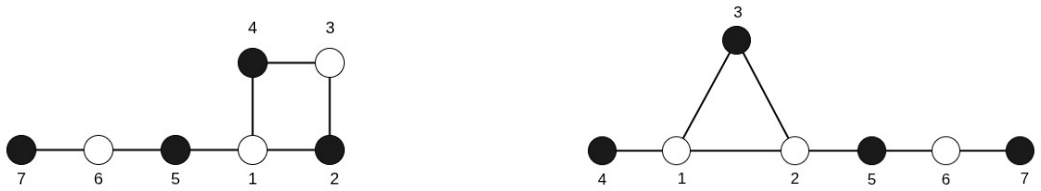
Next, we derive the induced minor obstructions for DAG treedepth 0, 1, and 2.

▶ **Theorem 28.** *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.*

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.



(a) H_1. **(b)** H_2.

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

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 the root and all other sources as children 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, 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 r the root of a child. There are two cases: 1) the root r has only one child s in the elimination tree T, 2) the root r has more than one child.

- **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 induces 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 an 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 one of its supergraphs, such that the *dtd* of the 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 obstructions in this case are P_7, H_1, and H_2.
  Now, consider the case $w \in R(s) \cap R(r)$ and $w \notin R(s_i)$, we can check that it forms an 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 a 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 the 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 exists a non-source in pairwise source reachability intersection; otherwise, we can root at s_1 and get a 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 supergraphs of *dtd* three as induced minors.
  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 that P_7 is a 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 obstructions for treedepth two are C_6, P_7, and the supergraphs of P_7 with $dtd = 3$. We can check that H_1 and H_2 are such graphs. Hence, the induced minor obstructions of tree depth two are C_6, P_7, H_1, and H_2. ◀

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

▶ **Lemma 29.** *Let H be a minor obstruction for treedepth $k-1$. Then $dtd(H_{sub}) \geq k$.*

▶ **Lemma 30.** *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.*

Using Lemma 29, we observe that the subdivision of any minor obstruction for treedepth k is an induced-minor obstruction for DAG treedepth k. Conversely, by Lemma 30, from any induced-minor obstruction for DAG treedepth k, one can obtain a minor obstruction for treedepth k via a sequence of edge contractions.

Let S denote the set of minor obstructions for treedepth k. For each $H \in S$, we construct a corresponding pattern H_{sub}. By construction, each H_{sub} is an induced-minor obstruction for DAG treedepth k. Moreover, we must also consider all supergraphs of H_{sub}. Let H^* be a supergraph of H_{sub} such that $\text{dtd}(H^*) \geq k$. Then H^* is also an induced-minor obstruction for DAG treedepth k.

Therefore, the set of induced-minor obstructions for DAG treedepth k can be derived from the set of minor obstructions for treedepth k. Since the number of minor obstructions for treedepth k is $\frac{(2^{2^{k-1}-k})(1+2^{2^{k-1}-k})}{2}$ [16], it implies that the number of induced-minor obstructions for DAG treedepth k is at least $\frac{(2^{2^{k-1}-k})(1+2^{2^{k-1}-k})}{2}$.

Giannopoulou and Thilikos [16] have listed the obstruction set of treedepth up 3. The number of graphs in the obstruction set of treedepth 3 is twelve. Consequently, to obtain the induced minor obstructions, one must consider the subdivisions of these graphs together with all of their supergraphs. Therefore, we gave induced minor obstruction till DAG treedepth at most 2.

3.2 Conditional Hardness and Small Pattern Counting

In this section, we present conditional lower bounds for counting patterns in cubic time using constant space. We then show that all patterns on at most nine vertices can be counted in cubic time and constant space. Finally, we exhibit a specific pattern on ten vertices and prove a conditional lower bound, showing that it cannot be counted in $O(n^{4-\varepsilon})$ time and constant space.

We have observed that DAGs with $\mathrm{dtd} = 1$ are precisely those having a single source. Moreover, specific orientations, such as the star graph with all leaves as sources, may contain multiple sources but still admit a linear-time, constant-space algorithm for counting homomorphisms. More generally, if a DAG admits an orientation where either the number of sources or non-sources is one, then homomorphism counting remains tractable in linear time and constant space. The set of patterns with $dtd = 1$ is cliques and star graphs. Leveraging the framework of [12], we extend our result for subgraph counting. One can observe that the spasm of cliques and star graphs is a clique or a star graph and therefore has *dtd* one. Hence, one can count the number of star graphs and cliques appearing as subgraphs 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 31.** *Let d be a constant. For any constant $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time constant-space algorithm for counting* $\hom(P_4, H)$*, where the input graph H is a d-degenerate graph.*

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

▶ **Theorem 32.** *Patterns that are countable as subgraphs in linear time and constant space are precisely the graphs with* $\mathrm{dtd} = 1$ *and the star graphs.*

Proof. A DAG with $\mathrm{dtd} = 1$ 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 $\mathrm{dtd} = 1$. Hence, for each vertex in the spasm of a $\mathrm{dtd} = 1$ graph, the number of homomorphisms can be counted in linear time and constant space. Consequently, the number of subgraphs of a DAG with $\mathrm{dtd} = 1$ 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. ◀

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

▶ **Conjecture 33.** *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.*

It is shown in [4] that the problem of counting C_6 even in degeneracy two is equivalent to counting triangles in a general graph. Thus, assuming Conjecture 33, there is no $O(n^{3-\varepsilon})$-time and constant-space algorithm for counting C_6 in a graph G with degeneracy equal to two. Further, consider the obstruction set of graphs with $\mathrm{dtd} = 2$. 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 [5]. Hence, we have the following :

▶ **Theorem 34.** *There is no $O(n^{3-\varepsilon})$-time and constant-space algorithm for counting graphs with* $\mathrm{dtd} = 3$ *in a graph G with degeneracy equal to two.*

Next, we prove that every pattern graph on at most nine vertices has DAG treedepth at most three. Consequently, by applying the framework of [12], it follows that homomorphisms, subgraphs, and induced subgraphs of all such patterns can be counted in $O(n^3)$ time using constant space.

▶ **Theorem 35.** *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 with at most three sources. 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 $\mathrm{dtd} \leq 3$. 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. ◀

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

▶ **Corollary 36.** *We can count subgraph P_{10} in time $O(n^3)$ using constant space.*

Using Theorem 35, we get an $O(n^3)$ time constant space algorithm up to nine vertices. We now present the hardness results for ten vertices. Consider a DAG on ten vertices, which is one subdivision of k_4. Using the reduction shown in the proof of Conjecture 7, we get that it reduces to counting k_4 in a 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.

References

1 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. doi:10.1109/ICDM.2015.141.

2 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, March 1997. doi:10.1007/BF02523189.

3 Suman K Bera, Lior Gishboliner, Yevgeny Levanzov, C Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *ACM Journal of the ACM (JACM)*, 69(3):1–21, 2022. doi:10.1145/3520240.

4 Suman K Bera, Noujan Pashanasangi, and C Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 38–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.38.

5 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. doi:10.1137/1.9781611976465.138.

6 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. doi:10.1145/3375395.3387665.

7 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021. doi:10.1007/S00453-021-00811-0.

8 Marco Bressan, Leslie Ann Goldberg, Kitty Meeks, and Marc Roth. Counting subgraphs in somewhere dense graphs. *SIAM Journal on Computing*, 53(5):1409–1438, 2024. doi:10.1137/22M1535668.

9 Marco Bressan, Julian Christoph Brinkmann, Holger Dell, Marc Roth, and Philip Wellnitz. The parameterised complexity of counting small sub-hypergraphs. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2002–2014. SIAM, 2026.

10 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. doi:10.1145/3564246.3585204.

11 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.

12 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017. doi:10.1145/3055399.3055502.

13 Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *Logical Methods in Computer Science*, 21, 2025. doi:10.46298/LMCS-21(2:14)2025.

14 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting h-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1-2):291–309, 2002. doi:10.1016/S0304-3975(02)00017-8.

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

16 Archontia C Giannopoulou and Dimitrios M Thilikos. Obstructions for tree-depth. *Electronic Notes in Discrete Mathematics*, 34:249–253, 2009. doi:10.1016/J.ENDM.2009.07.041.

17 Lior Gishboliner, Yevgeny Levanzov, Asaf Shapira, and Raphael Yuster. Counting homomorphic cycles in degenerate graphs. *ACM Transactions on Algorithms*, 19(1):1–22, 2023. doi:10.1145/3560820.

18 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. doi:10.1145/2736277.2741101.

19 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Conjunctive queries with free access patterns under updates. *Logical Methods in Computer Science*, 21, 2025. doi:10.46298/LMCS-21(2:23)2025.

20 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. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.40.

21 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. doi:10.1145/3717823.3718251.

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

23 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. doi:10.1145/3336191.3371773.

24 Daniel Paul-Pena and C Seshadhri. A dichotomy theorem for linear time homomorphism orbit counting in bounded degeneracy graphs. In *35th International Symposium on Algorithms and Computation*, 2024.

25 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. doi:10.1137/1.9781611978322.2.

26 Daniel Paul-Pena and C Seshadhri. Subgraph counting in subquadratic time for bounded degeneracy graphs. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, pages 124–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.124.

27 Daniel Paul-Pena and C Seshadhri. Near-linear time subhypergraph counting in bounded degeneracy hypergraphs. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3652–3687. SIAM, 2026. doi:10.1137/1.9781611978971.134.

28 C Seshadhri. Some vignettes on subgraph counting using graph orientations (invited talk). In *26th International Conference on Database Theory (ICDT 2023)*, pages 3–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICDT.2023.3.

29 Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.