Counting subgraphs via DAG tree decompositions

Marco Bressan
Sapienza University of Rome
bressan@di.uniroma1.it

Abstract
We study the problem of counting subgraphs in a graph $G$ of degeneracy $d$, a good sparsity measure for many real-world graphs. Our main tool is a decomposition for directed acyclic graphs, which we call DAG tree decomposition. This decomposition induces a width measure $s(H)$ on every undirected pattern $H$, and leads to a dynamic programming to count the copies of $H$ in $G$ in time $f(d, k) \cdot \tilde{O}(n^{s(H)})$. This is tight in the following sense: if any algorithm solves the problem in time $f(d, k) \cdot n^{o(s(H)/\ln s(H))}$ for all patterns $H$, then ETH (the Exponential Time Hypothesis) fails.

This result has multiple consequences. First, the induced or non-induced copies of any pattern $H$ on $k$ nodes (even disconnected ones) can be counted in time $f(d, k) \cdot n^{O(\alpha(H))}$ where $\alpha(H)$ is the independence number of $H$, and no algorithm can do so in $f(d, k) \cdot n^{o(\alpha(H)/\ln \alpha(H))}$ unless ETH fails. Second, by bounding $s(H)$ we rediscover and enrich classic results. For cliques minus $\epsilon$ edges we give a time bound of $f(d, k) \cdot \tilde{O}(n^{\frac{1}{4} + \epsilon^2})$, which extends the classic $f(d, k) \cdot O(n)$ bound for cliques by Chiba and Nishizeki [9]. For complete multipartite graphs, we can count non-induced copies in $f(d, k) \cdot \tilde{O}(n)$ (essentially matching an $f(d, k) \cdot O(n)$ bound by Eppstein [13]), which becomes $f(d, k) \cdot \tilde{O}(n^{\frac{1}{2} + \epsilon})$ for complete multipartite graphs plus $\epsilon$ edges. Third, we can count the induced or non-induced copies of any $H$ in $f(d, k) \cdot \tilde{O}(n^{\frac{2}{4} + \epsilon})$, beating the decades-old state-of-the-art $O(n^{\frac{2}{4} + \epsilon})$ algorithm of Nešetřil and Poljak [28] for $d < n^{0.72}$ ($\omega$ is the matrix multiplication exponent). This also gives faster subgraph counting algorithms for graphs of bounded average degree. All bounds hold for the weighted, node-colored, or edge-colored versions of counting. These results suggest our DAG tree decomposition may be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Design and analysis of algorithms

Keywords and phrases subgraph counting, tree decomposition, degeneracy

Related Version https://arxiv.org/abs/1805.02089

1 Introduction

Given a host graph $G$ on $n$ nodes and a pattern graph $H$ on $k$ nodes, we want to count the number of induced subgraphs of $G$ that are isomorphic to $H$. This problem is notoriously hard: it is naturally believed that even just detecting a $k$-clique in an $n$-node graph requires time $n^{\Omega(k)}$ [7, 8], and that counting cycles or matchings on $k$ nodes requires time $n^{\Omega(k/\ln(k))}$ [11].

Correspondingly, the fastest known induced subgraph counting algorithm [28] has running time $O(n^{\omega(k/\ln(k))})$ where $\omega$ is the matrix multiplication exponent. One way of circumventing this barrier is to refine the bounds by adding parameters (besides $n$ and $k$) that capture the structure of the input. For example, if $H$ has vertex-cover number $c$ then its non-induced copies in $G$ can be counted in time $f(k) \cdot n^{c+O(1)}$ [24, 35, 5, 11]. Or, if $H$ has treewidth $t$, then its homomorphisms into $G$ can be counted in time $f(k) \cdot O(n^{t+1})$ [19]. Finding the right parameterization can be nontrivial, but gives back better algorithms and a deeper understanding of the problem.

In this paper we focus on the case where $G$ is sparse, which is often true in practice. The main measure of sparsity we adopt is degeneracy: the smallest integer $d$ such that every subgraph of $G$ has minimum degree bounded by $d$. We thus parameterize our bounds by $n, k, d$, which is natural for many reasons. First, in many applications $G$ is a social network, and in social networks $d$ is often small (see e.g. [15]); in fact, generative models...
like preferential attachment \cite{3} stipulate \( d = O(1) \). Second, it is known that cliques and complete bipartite graphs can be counted in time \( f(d, k) \cdot O(n) \) \cite{9, 13}. These bounds could be special cases of a more general result (they are). Third, many practical pattern counting algorithms are based on low-degree orientations of \( G \) \cite{36, 22, 30}, similarly to the present work. A principled, deeper explanation of their effectiveness might help.

We note that most subgraph counting bounds in the literature are parameterized by the structure of \( H \) (e.g. by its vertex-cover number or its treewidth) rather than by the structure of \( G \). In addition, bounds parameterized by the structure of \( G \) that hold for all \( H \) are quite restrictive, as they demand that \( G \) is planar or has bounded maximum degree \( \Delta \). A few bounds parameterized by the degeneracy \( d \) of \( G \) exist, but they come from specialized algorithms for specific patterns (cliques or complete bipartite graphs). No general bound parameterized by \( d \) is known, nor it is clear how to harness degeneracy in general.

\subsection{Results}

We present novel bounds for counting homomorphisms, non-induced copies, and induced copies of a \( k \)-node pattern graph \( H \) in an \( n \)-node graph \( G \), parameterized by \( n, k \) and \( d \) where \( d \) is the degeneracy of \( G \). As a first step, we transform \( G \) into a dag (directed acyclic graph) with maximum outdegree \( d \) via the well-known degeneracy orientation (see below), and reduce to the problem of counting the homomorphisms of \( k \)-node dags in \( G \). We then introduce our main tool: a tree decomposition for dags, that we call \emph{dag tree decomposition}, designed to exploit the degeneracy orientation of \( G \) algorithmically. Such a decomposition allows one to count homomorphisms naturally via dynamic programming, exactly like the standard tree decomposition of a graph. On any undirected pattern graph \( H \) the dag decomposition induces a \emph{dag treewidth} \( s(H) \) that captures the cost of counting \( H \) via the dynamic programming. Let indeed \( \text{hom}(H, G) \), \( \text{sub}(H, G) \), and \( \text{ind}(H, G) \) denote respectively the number of homomorphisms, copies, and induced copies of \( H \) in \( G \). We prove:

\begin{itemize}
  \item \textbf{Theorem 1.} For any \( k \)-node pattern \( H \) one can compute \( \text{hom}(H, G) \), \( \text{sub}(H, G) \), and \( \text{ind}(H, G) \) in time \( f(d, k) \cdot \tilde{O}(n^{s(H)}) \). Moreover, if there exists an algorithm that computes \( \text{sub}(H, G) \) or \( \text{ind}(H, G) \) in time \( f(d, k) \cdot n^{o(s(k)/\ln s(k))} \) for all \( H \), then the Exponential Time Hypothesis \cite{21} fails.
\end{itemize}

We show that \( s(H) = \Theta(\alpha(H)) \), where \( \alpha(H) \) is the independence number of \( H \). Thus, the cost of counting subgraphs in graphs of bounded degeneracy is driven by the size of its largest independent set, at least for upper bounds.\footnote{Note that the lower bound is not guaranteed if we restrict the choice of \( H \). Thus there may be families of patterns with large \( \alpha(H) \) whose patterns can be counted much faster than \( n^{o(H)} \). This is certainly true for non-induced counting – see Theorem \cite{3} in social graph mining \cite{31, 33, 32}. Next, we prove:}

With our dag decomposition in place, it is relatively easy to rediscover and generalize old bounds. First, we show:

\begin{itemize}
  \item \textbf{Theorem 2.} If \( H \) is the clique minus \( \epsilon \) edges, then one can compute \( \text{hom}(H, G) \), \( \text{sub}(H, G) \), and \( \text{ind}(H, G) \) in time \( f(d, k) \cdot \tilde{O}(n^{\frac{1}{2}+\sqrt{\epsilon}}) \).
\end{itemize}

This generalizes the well-known \( O(nd^{k-1}) \) bound for counting cliques \cite{3}, possibly losing a \( \tilde{O}(f(k)) \) factor. We remark that counting quasi-cliques is of interest in social graph mining \cite{31, 33, 32}. Next, we prove:
Theorem 3. If $H$ is a complete multipartite graph, then one can compute $\text{hom}(H, G)$ and $\text{sub}(H, G)$ in time $f(d, k) \cdot O(n)$. If $H$ is a complete multipartite graph plus $\epsilon$ edges, then one can compute $\text{sub}(H, G)$ in time $f(d, k) \cdot \widetilde{O}(n^{\frac{1}{2} + \frac{\epsilon}{2}})$.

This generalizes the $f(d, k) \cdot O(n)$ bound to count non-induced complete bipartite graphs by Eppstein [13]. Both Theorem 2 and 3 follow easily from Theorem 4 by bounding $s(H)$.

We then prove a general bound for every $k$-node pattern (including disconnected ones):

Theorem 4. For any $k$-node pattern $H$ one can compute $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$ in time $f(d, k) \cdot \widetilde{O}(n^{\frac{1}{2} + \frac{\epsilon}{2} + \frac{3}{2}})$.

This should be compared to the $O(n^{\frac{1}{2} + \frac{\epsilon}{2}})$ algorithm of [28], which is the current state of the art for induced subgraph counting. If we let $d$ depend on $n$, then we can show our algorithm is faster as long as $d < n^{0.721}$ assuming the current value $\omega \approx 2.373$ [25], and in any case for $d < n^\delta \approx n^{0.556}$ since $\omega \geq 2$.

As a consequence of Theorem 4, we obtain faster algorithms for graphs with small average degree $\delta$. This comes from the fact that $d = O((\delta n)^{1/2})$—see e.g. [9]. Our algorithm is faster than [28] when $\delta < n^{0.442}$ assuming $\omega \approx 2.373$, and in any case when $\delta < n^{0.112}$. In particular, if $\delta = O(1)$ then we can count any pattern in time $f(k) \cdot \widetilde{O}(n^{\frac{1}{2} + \frac{1}{2}})$. To the best of our knowledge, this is the first general algorithm faster than [28] for sparse graphs.

We conclude with a bound parameterized by the maximum degree $\Delta$ of $G$.

Theorem 5. For any $k$-node pattern $H$ one can compute $\text{hom}(H, G)$, $\text{sub}(H, G)$ and $\text{ind}(H, G)$ in time $f(k) \cdot O(\Delta^{k-1} n)$.

This improves the dependence on $\Delta$ of a recent $\widetilde{O}((7\Delta)^{2k} n)$ bound for computing $\text{ind}(H, G)$ by Patel et al. [29], and does so with a considerably simpler algorithm.

We note that all our results hold for the colored versions of the problem (count only copies of $H$ with prescribed vertex and/or edge colors) as well as the weighted versions of the problem (compute the total weight of copies of $H$ in $G$ where $G$ has weights on nodes or edges). This follows immediately by adapting our homomorphism counting algorithms.

1.2 Preliminaries and notation

The host graph $G = (V, E)$ and the pattern graph $H = (V_H, E_H)$ are simple undirected graphs. For any subset $V' \subseteq V$ we denote by $G[V']$ the subgraph $(V', E \cap (V' \times V'))$ induced by $V'$; the same notation applies to $H$. A homomorphism from $H$ to $G$ is a map $\phi : V_H \to V$ such that $\{u, u'\} \in E_H$ implies $\{\phi(u), \phi(u')\} \in E$. We usually write $\phi : H \to G$ to make clear the edges that $\phi$ must preserve. When $H$ and $G$ are oriented, $\phi$ must preserve the direction of the arcs. If $\phi$ is injective then we have an injective homomorphism. We denote by $\text{hom}(H, G)$ and $\text{inj}(H, G)$ the number of homomorphisms and injective homomorphisms from $H$ to $G$. We denote by $\psi$ a map that is not necessarily a homomorphism. The symbol $\simeq$ denotes isomorphism. For a subgraph $F \subseteq G$, if $F \simeq H$ then $F$ is a copy of $H$ in $G$. If furthermore $F \simeq G[V_F]$ then $F$ is an induced copy. We denote by $\text{sub}(H, G)$ and $\text{ind}(H, G)$ the number of copies and induced copies of $H$ in $G$. When no confusion arises we can omit $G$ in the notation. In most of the paper we give to the edges of $H$ an acyclic orientation $\sigma$. We denote the resulting dag by $P$. All notation described above applies to $P$ in the obvious way. We denote by $s(H)$ the dag treewidth of $H$ (defined later).

The degeneracy of $G$ is the smallest integer $d$ such that there exists an acyclic orientation of $G$ with maximum outdegree bounded by $d$ (this is equivalent to the definition given in the introduction). It is folklore that such an orientation can be found in time $O(|E|)$.
by repeatedly removing from \( G \) a lowest-degree node \(^2\). From now on we assume \( G \) has this orientation\(^2\). We assume \( G \) is encoded as sorted adjacency lists. To lighten the notation we assume we can check if an arc \( uv \) is in \( G \) in time \( O(1) \), although this would be \( \tilde{O}(d) \). We denote by \( \delta \) and \( \Delta \) respectively the average and maximum degree of \( G \) (seen as undirected). We assume \( k = O(1) \); nonetheless most of our bounds hold in their current form for \( k = O(\ln n) \) or \( k = O(\sqrt{\ln n}) \) as well. The \( O() \) notation hides \( \text{poly}(k) \) factors and the \( \tilde{O}() \) notation hides \( \text{polylog}(\cdot) \) factors.

### 1.3 Related work

The tractability of subgraph counting depends on what one is computing (\( \text{hom}(H,G) \), \( \text{sub}(H,G) \), or \( \text{ind}(H,G) \)), and under which parameterization. Computing \( \text{hom}(H,G) \) can be done in time \( f(k) \cdot O(n^{t+1}) \) \(^\ddagger\) where \( t = t(H) \) is the treewidth of \( H \) (see Appendix A.1), and almost-matching lower bounds based on ETH (the Exponential Time Hypothesis) are known \(^1\). Computing \( \text{sub}(H,G) \) can be done in time \( f(k) \cdot n^{\omega(H)} + O(1) \), where \( c(H) \) is the vertex-cover number of \( H \) \(^\ddagger\) \( \omega \) \(^\ddagger\) \(^\ddagger\), and again almost-tight lower bounds for cycles or matchings based on ETH exist \(^1\). Computing \( \text{ind}(H,G) \) can be done in time \( O(n^k) \) by trivial enumeration, and even detecting a \( k \)-clique in an \( n \)-node graph requires time \( n^{\Omega(k)} \) \(^\ddagger\) \(^\ddagger\) \(^\ddagger\) unless ETH fails. We note that in general there is no relationship between our dag treewidth \( s(H) \) and \( t(H) \), \( c(H) \), \( |V_H| \), and in particular we can have \( s(H) = O(1) \) and \( t(H), c(H) \in \Omega(k) \). For completeness, we shall recall that detection can be substantially easier than counting – for example, \( k \)-paths can be found in time \( f(k) \cdot n^{O(1)} \) \(^\ddagger\).

Concerning the precise dependence on \( n \), the fastest known algorithm for computing \( \text{ind}(H,G) \) is the one by Nešetřil and Poljak \(^\ddagger\), with running time \( O(n^{\omega(k/3)} \cdot (k \mod 3) + 1) \). This algorithm does not run faster if \( G \) is sparse, since it works on an auxiliary graph which can be dense even if \( \delta = O(1) \). One of our goals is precisely to lower the exponent of \( n \) when \( G \) is sparse (as has been done when \( H \) is sparse \(^\ddagger\)). Note that we do so for all patterns.

Finally, coming to exploiting the degeneracy \( d \) of \( G \), bounds are known only for special classes of patterns. Chiba and Nishizeki \(^\ddagger\) show how to count \( k \)-cliques in time \( O(d^{d-1}n) \), which can be improved to \( O(d^{\omega(k-1/3)} n) \) \(^\ddagger\) via fast matrix multiplication \(^\ddagger\). Eppstein shows how to list complete bipartite subgraphs in time \( O(d^{3/2} n) \) \(^\ddagger\) and maximal cliques in \( O(d^{3/2} n) \) \(^\ddagger\). All these algorithms exploit the degeneracy orientation of \( G \). Finally, in bounded-degree graphs and planar graphs, where \( d = O(1) \), one can count any pattern on \( k \) nodes in time \( f(k) \cdot O(n) \) \(^\ddagger\). However, no non-trivial general upper bounds or lower bounds parameterized by \( n, k, d \) are known. We also note that there exists a tree decomposition for directed graphs \(^\ddagger\), which however bears no resemblance to ours.

### Paper organization

Section \(^2\) introduces our dag tree decomposition and sketches a proof of the upper bounds of Theorem \(^1\); these are our two main technical tools. Section \(^3\) gives dag treewidth bounds for quasi-cliques and quasi-complete multipartite graphs, as well as for general patterns, proving Theorems \(^2\) \(^\sharp\) \(^\sharp\) \(^\sharp\). Section \(^4\) proves the lower bounds of Theorem \(^4\).

Finally, Section \(^5\) proves Theorem \(^5\) All technical parts omitted for space limitations can be found in the appendix.

\(^2\) It is easy to see that any (even cyclic) orientation of \( G \) contains a node with outdegree at least \( d/2 \).

Therefore choosing an acyclic orientation causes no substantial loss.

\(^3\) Variants exist with running time \( O(n^{\omega(k/3)} \cdot (k-1/3) \cdot (k/3)) \) where \( \omega(p,q,r) \) is the cost of multiplying an \( n^p \times n^q \) matrix by an \( n^q \times n^r \) matrix (see \(^\ddagger\)).
2 DAG tree decompositions

In this section we introduce a tree decomposition for dags, designed to exploit the degeneracy orientation of $G$. Let $P = (V_P, E_P)$ be a directed acyclic pattern on $k$ nodes. We denote by $S_P \subseteq V_P$, or simply $S$ if not ambiguous, the set of nodes of $P$ having no incoming arc. We call them the sources of $P$. We denote by $V_P(u)$ the transitive closure of $u$, i.e. the set of nodes reachable in $P$ from $u$, and we let $P(u) = P[V_P(u)]$. More generally, for a subset of sources $B \subseteq S$ we let $V_P(B) = \cup_{u \in B} V_P(u)$ and $P(B) = P[V_P(B)]$. We call $B$ a bag of sources.

Let us start with a simple example. Suppose we want to count the copies of $P$ in $G$. Since we have oriented $G$ acyclically, we can just orient $H$ acyclically in every possible way, count the copies of every such orientation in $G$, and sum the counts. If $H$ was a $k$-clique, then all of its acyclic orientations have exactly one source. We can then just list the $n$ possible images of this source in $G$, and search the $\leq d$ neighbors for the images of the remaining $k - 1$ nodes. This is the classic Chiba-Nishizeki clique listing algorithm that runs in time $O(d^{k-1}n)$. More generally, if an acyclically oriented version of $H$ has $s$ sources, we can extend the argument above and list its copies in $G$ in time $O(d^{k-s}n^s)$ – we try all $\binom{n}{s}$ images of the source set, and search for the remaining $k - s$ nodes by following the arcs of $G$. Unfortunately, $s$ can be as large as $k - 1$ and thus we get a running time of $O(dn^{k-1})$.

This is where our dag tree decomposition comes into play. First of all, we limit ourselves to computing $\text{hom}(H,G)$, and derive $\text{sub}(H,G)$ and $\text{ind}(H,G)$ via standard inclusion-exclusion arguments. Now, instead of naively enumerating the homomorphisms of the entire pattern, we break it into smaller pieces that we can enumerate faster. This enables the same kind of bottom-up counting approach given by the standard tree decomposition of a graph. The decomposition we need is however different: the nodes of the tree decomposition are source bags, and if a node is reachable from sources of two different bags, then it is reachable also from sources in every bag on the path between the two. The total cost is dominated by $n$ raised to the size of the largest bag. Therefore, if we can find a dag tree decomposition where all bags are small, we are done. It is time to introduce formally our decomposition.

\textbf{Definition 6.} Let $P = (V_P, A_P)$ be a dag. A dag tree decomposition (dtd) of $P$ is a rooted tree $T = (B, E)$ such that:

1. each node $B \in B$ is a bag\(^4\) of sources $B \subseteq S_P$
2. $\bigcup_{B \in B} B = S_P$
3. if $B$ lies on the unique path between $B_1$ and $B_2$ in $T$, then $V_P(B_1) \cap V_P(B_2) \subseteq V_P(B)$

The width of $T$ is $s(T) = \max_{B \in B} |B|$. The dag treewidth $s(P)$ of $P$ is the minimum of $s(T)$ over all dag tree decompositions $T$ of $P$.

For any $B \in B$ we denote by $T(B)$ the subtree of $T$ rooted at $B$, and by $\Gamma[B]$ the union of all bags in $T(B)$, also called down-closure of $B$ in $T$. Note that $P(B)$ is the subgraph of $P$ reachable from $B$, while $P(\Gamma[B])$ is the subgraph of $P$ reachable from $\Gamma[B] \supseteq B$.

One last definition, and we can move to our main result on computing $\text{hom}(H,G)$.

\textbf{Definition 7.} Let $P_1 = (V_{P_1}, A_{P_1}), P_2 = (V_{P_2}, A_{P_2})$ be two subgraphs of $P$, and let $\phi_1 : P_1 \to G$ and $\phi_2 : P_2 \to G$ be two homomorphisms. We say $\phi_1$ and $\phi_2$ respect each other if $\phi_1(u) = \phi_2(u)$ for all $u \in V_{P_1} \cap V_{P_2}$. We denote by $\text{hom}(P_1, G, \phi_2)$ or simply $\text{hom}(P_1, \phi_2)$ the number of homomorphisms from $P_1$ to $G$ that respect $\phi_2$.

\(^4\) In classic tree decompositions, often the bags are associated to the nodes of $T$ rather than being themselves the nodes. We opted for a slightly informal definition to keep the discussion lighter.
We can then prove (see Appendix A.2):

**Theorem 8.** Let $T = (B, E)$ be a dag tree decomposition for $P$, and choose any $B \in B$. There is an algorithm HomCount($P, T, B$) that computes $\text{hom}(P(\Gamma[B]), \phi_B)$ for all $\phi_B : P(B) \to G$ in time $f(d, k) \cdot \tilde{O}(n^{\omega(T)})$.

In a nutshell, HomCount($P, T, B$) proceeds as follows. First, for all leaves $B' \in T$ it explicitly enumerates all homomorphisms $\phi : P(B') \to G$. Note that $P(B') = P(\Gamma[B'])$. For every internal node $B' \in T$, it goes as follows. First, again it enumerates all homomorphisms $\phi : P(B') \to G$. For every such $\phi$, then, it combines the counts $P(\Gamma[B']', \phi)$ computed before for every child $B'_i$ of $B'$. This gives the value of $\text{hom}(P(\Gamma[B']), \phi)$. Once at the $B$, the algorithm returns $\text{hom}(P(\Gamma[B]), \phi)$ for every $\phi : P(B) \to G$. This is essentially the dynamic programming algorithm over tree decompositions (see e.g. [19]), but showing its correctness requires a proof from scratch.

To compute $\text{hom}(P, G)$ we just compute a dag tree decomposition $T$ and then invoke HomCount($P, T, B$) where $B$ is the root of $T$. By Theorem 8 this returns $\text{hom}(P(\Gamma[B]), \phi_B)$ for all $\phi_B : P(B) \to G$. But if $B$ is the root then $P(\Gamma[B]) = P(S_P) = P$. Hence $\text{hom}(P, G) = \sum_{\phi_B : P(B) \to G} \text{hom}(P(\Gamma[B]), \phi)$, that is, we just need to sum all counts. Note that a dag tree decomposition for $P$ of width $s(P)$ can obviously be found in time $f(k)$ (in Section 3 we show how to compute a low-width $T$ in time $O(1.7549k + \text{poly}(k))$). Therefore one can compute $\text{hom}(P, G)$ in time $f(k) \cdot \tilde{O}(n^{s(P)})$.

**From homomorphisms to (induced) copies.** Any upper bound on the cost of computing $\text{hom}(P, G)$ translates to upper bounds on the cost of computing $\text{hom}(H, G)$, $\text{sub}(H, G)$, and $\text{ind}(H, G)$ via standard inclusion-exclusion arguments. Let $H$ be any simple $k$-node graph. Let $\Sigma(H)$ be the set of all dags $P$ that can be obtained by orienting $H$ acyclically. Let $\Theta(H)$ be the set of all equivalence relationships on $V_H$, and for $\theta \in \Theta$ let $H/\theta$ be the pattern obtained from $H$ by identifying equivalent nodes according to $\theta$ and removing loops and multiple edges. Let $D(H)$ be the set of all supergraphs of $H$ (including $H$) on the same node set $V_H$.

**Definition 9.**

\begin{align*}
  s_1(H) &= \max\{s(P) : P \in \Sigma\} \\
  s_2(H) &= \max\{s_1(H/\theta) : \theta \in \Theta\} \\
  s(H) &= s_2(H) = \max\{s_2(H') : H' \in D(H)\}
\end{align*}

Then (see Appendix A.3):

**Lemma 10.** One can compute $\text{hom}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{s_1(H)})$, $\text{sub}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{s_2(H)})$, and $\text{ind}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{s(H)})$.

We can thus focus on bounding $s_1(H)$, $s_2(H)$ and $s(H)$, which we do in the next section.

## 3 Bounds for the dag treewidth

In this section we bound $s_1(H), s_2(H), s(H)$, which by Lemma 10 implies time bounds for counting $\text{hom}(H, G)$, $\text{sub}(H, G)$, $\text{ind}(H, G)$. First, we bound $s(H)$ for cliques minus $\epsilon$ edges; this implies a generalization of the classic clique counting bound by Chiba and Nishizeki [9]. Second, we bound $s_2(H)$ for complete multipartite graphs plus $\epsilon$ edges, which implies a generalization of a result by Eppstein [13]. Third, we bound $s(H)$ for every pattern. This
leads to better upper bounds for computing \( \text{hom}(H, G) \), \( \text{sub}(H, G) \), and \( \text{ind}(H, G) \) if \( G \) is sparse enough. Finally, we show that \( s(H) = \Theta(\alpha(H)) \), where \( \alpha(H) \) is the independence number of \( H \), i.e. the size of its largest independent set.

### 3.1 Quasi-cliques

**Lemma 11.** If \( H \) has \( \binom{|S|}{2} - \epsilon \) edges then \( s(H) \leq \left[ \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \right] \).

**Proof.** The source set \( |S| \) of \( P \) is an independent set, hence \( |E_H| \leq \binom{|S|}{2} - \binom{|S|}{2} \). Therefore \( \epsilon \geq \binom{|S|}{2} \), which implies \( |S| \leq 1 + \sqrt{\frac{\epsilon}{2}} \). A d.t.d. for \( P \) is the tree on two bags \( B_1, B_2 \) that satisfy \( B_1 \cup B_2 = S \), \( |B_1| = |S|/2 \), and \( |B_2| = |S|/2 \). Hence \( s(P) \leq |S|/2 \leq \left[ \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \right] \).

Now consider any \( H' \) obtained from \( H \) by adding edges or identifying nodes. Obviously \( |E_{H'}| \geq \binom{|S|}{2} - \epsilon \) where \( k' = |V| \), and the argument above implies \( s(P') \leq \left[ \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \right] \) for any orientation \( P' \) of \( H' \). By Definition 9 then, \( s(H) \leq \left[ \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \right] \).

By Lemma 10 and since \( s_1(H) \leq s_2(H) \leq s(H) \) it follows that we can compute \( \text{hom}(H, G) \), \( \text{sub}(H, G) \), and \( \text{ind}(H, G) \) in time \( f(d, k) \cdot \tilde{O}(n^{\frac{3}{4} + \sqrt{\frac{\epsilon}{2}}}) \), proving Theorem 2.

### 3.2 Quasi-multipartite graphs

**Lemma 12.** If \( H \) is a complete multipartite graph, then \( s_2(H) = 1 \). If \( H \) is a complete multipartite graph plus \( \epsilon \) edges, then \( s_2(H) \leq \left[ \frac{1}{2} \right] + 2 \).

**Proof.** Suppose \( H \) is complete multipartite. Let \( H = (V_H, E_H) \) with \( V_H = V_H^1 \cup \ldots \cup V_H^n \) where each \( H[V_H^j] \) is a maximal independent set. Note that, in any orientation \( P \) of \( H \), all sources are contained in exactly one \( V_H^j \). Moreover, \( V_H(u) = V_H(u') \) for any two sources \( u, u' \).

A d.t.d. \( T \) of \( H \) is the trivial one with one source per bag.

Note then we add \( \epsilon \) edges to \( H \). Again, in any orientation \( P \) of \( H \), all sources are contained in exactly one \( V_H^j \), but we might have \( V_H(u) \neq V_H(u') \) for different sources \( u, u' \).

This arguments apply also to any pattern \( H' \) obtained by identifying nodes of \( H \): if there is a source node \( u \) in \( H' \) that in \( H \) is in \( V_H^j \), then every node of \( H' \) that in \( H \) is in \( V_H \setminus V_H^j \) is reachable from \( u \). In addition, if a node in \( V_H \setminus V_H^j \) has been identified with a node in \( V_H \setminus V_H^j \) then all nodes are reachable from all sources and there is a trivial d.t.d. of width 1. Otherwise, in \( H' \) the nodes of \( V_H^j \) have been identified with a subset of \( V_H^j \) itself and we just need a d.t.d. of width at most \( \left[ \frac{1}{2} \right] + 2 \) as above.

By Lemma 10 it follows that if \( H \) is a complete multipartite graph then we can compute \( \text{hom}(H, G) \) and \( \text{sub}(H, G) \) in time \( f(d, k) \cdot \tilde{O}(n) \). If instead \( H \) is a complete multipartite graph plus \( \epsilon \) edges, then we can compute \( \text{hom}(H, G) \) and \( \text{sub}(H, G) \) in time \( f(d, k) \cdot \tilde{O}(n^{\frac{3}{4} + \epsilon}) \).

This proves Theorem 3.

### 3.3 General patterns

This subsection is entirely devoted to prove:

**Theorem 13.** For any dag \( P = (V_P, A_P) \), in time \( O(1.7549^k + poly(k)) \) we can compute a dag tree decomposition \( T = (B, E) \) with \( |B| \leq 4k \) and \( s(T) \leq \min\left(\left\lfloor \frac{k}{4} \right\rfloor, \left\lceil \frac{k}{2} \right\rceil \right) + 2 \), where \( k = |V_P| \) and \( e = |A_P| \).
The proof makes heavy use of the skeleton graph of $P$. Let us say a node $v \in V_P$ is a joint if it is reachable from distinct sources, i.e. if $v \in V_P(u) \cap V_P(u')$ for some $u, u' \in S$ with $u \neq u'$. We write $J_P$ or simply $J$ for the set of all joints of $P$, and $J(u)$ for the set of joints reachable from $u$. Then:

**Definition 14.** The skeleton of a dag $P = (V_P, A_P)$ is the directed bipartite graph $\Lambda(P) = (S \cup J, E_\Lambda)$ where $E_\Lambda \subseteq S \times J$ and $(u, v) \in E_\Lambda$ if and only if $v \in V_P(u)$.

Figure 1 gives an example. Note that $\Lambda(P)$ ignores nodes that are neither sources nor joints.

![Figure 1](image)

**Figure 1** Left: a dag $P$. Right: its skeleton $\Lambda(P)$ (sources $S$ above, joints $J$ below).

The proof of Theorem 13 proceeds as follows. Let $\Lambda = \Lambda(P)$ for short. First, we build $\Lambda$, which clearly takes $O(\text{poly}(k))$. We then greedily pick a subset $B \subseteq S$ such that $|J(B)| \geq 3|B|$, removing $B \cup J(B)$ from $\Lambda$. If this empties $\Lambda$, then we simply set $T = (\{B\}, \emptyset)$ and we have finished. Otherwise, $\Lambda$ is shattered into $\ell \geq 1$ connected components $\Lambda_1, \ldots, \Lambda_\ell$. We therefore take each $\Lambda_i$ in turn, and use it to build a d.t.d. $T_i = (B_i, E_i)$; we show all these $T_i$ can be combined together with $B$ into a d.t.d. for $P$. Most of the work is in building $T_i$, and in particular in bounding its width, which requires several manipulations of $\Lambda_i$ followed by an invocation of treewidth bounds on a certain “core” graph which is in some sense the hard part of $\Lambda_i$.

Let us now delve into the proof. For any joint $v \in J$ we let $S(v) = \{u \in S : v \in J(u)\}$, and similarly for a pair of joints $(v, v')$ we let $S(v, v') = \{u \in S : J(u) = \{v, v'\}\}$. For any node $x$, by $d_x$ we mean its degree in the skeleton graph $\Lambda$.

1. **Shattering the skeleton.** Set $B^{(0)} = \emptyset$ and let $\Lambda^{(0)} = (S^{(0)} \cup J^{(0)}, E_\Lambda^{(0)})$ be a copy of $\Lambda$. Set $i = 0$ and proceed iteratively as follows. If there is a source $u \in S^{(i)}$ with $d_u \geq 3$, let $B^{(i+1)} = B^{(i)} \cup \{u\}$, and let $\Lambda^{(i+1)}$ be obtained from $\Lambda^{(i)}$ by removing $\{u\}$ and $J^{(i)}(u)$; otherwise stop. Suppose the procedure stops at $i = i$. Since at each step we add one source node to $B^{(i)}$ and remove at least 4 nodes from $\Lambda^{(i)}$, then $|B^{(i)}| \leq |\Lambda^{(i)}|/4 \leq (k - |\Lambda^{(0)}|)/4$.

Now consider the nodes $\{u\} \cup J^{(i)}(u)$ removed at step $i$. Note that $\Lambda^{(i)}$ is just the skeleton of $P^{(i)} = P \setminus P(B^{(i)})$, and that $J^{(i)}(u) \subseteq P^{(i)}(u)$. This implies $P^{(i)}(u)$ contains at least 3 arcs. Moreover, there must be at least one arc from $P^{(i)}(u)$ to $P^{(i)}(u)$, otherwise $P^{(i)}(u)$ would not contain joints of $P^{(i)}$. We have therefore at least 4 arcs pointing to nodes of $P^{(i)}(u)$.

**Lemma 15.** For $i = 1, \ldots, \ell$ let $T_i = (B_i, E_i)$ be a d.t.d. of $\Lambda_i$. Consider the tree $T$ obtained as follows. The root of $T$ is the bag $B^{(i)}$, and the subtrees below $B^{(i)}$ are $T_1, \ldots, T_\ell$, where each bag $B \in T_i$ has been replaced by $B \cup B^{(i)}$. Then $T = (B, E)$ is a d.t.d. of $P$ with $s(T) \leq |B^{(i)}| + \max_{i=1, \ldots, \ell} s(T_i)$ and $|B| = \sum_{i=1}^\ell |B_i|$.
Proof. The claims on \( s(T) \) and \(|B|\) are trivial. Let us then check via Definition 6 that \( T \) is a d.t.d. of \( P \). Point (1) is immediate. For point (2), note that \( \cup_{B \in B_i} = S_i \) because \( T_i \) is by hypothesis a d.t.d. of \( \Lambda_i \); by construction, then, \( \cup_{B \in B} = B^{i(1)} \cup_{i=1}^|S| S_i = S_P \). Now point (3). Pick any two bags \( B' \cup B^{i(1)} \) and \( B'' \cup B^{i(1)} \) of \( T \), where \( B' \in T_i \) and \( B'' \in T_j \) for some \( i,j \in \{1, \ldots, \ell \} \), and any bag \( B \cup B^{i(1)} \in T(B' \cup B^{i(1)}, B'' \cup B^{i(1)}) \). Suppose first \( i = j \); thus by construction \( B \in T(B', B'') \). Since \( T_i \) is a d.t.d., then \( J_i(B') \cap J_i(B'') \subseteq J_i(B) \), and in \( T_i \) this implies \( V_P(B' \cup B^{i(1)}) \cap V_P(B'' \cup B^{i(1)}) \subseteq V_P(B \cup B^{i(1)}) \). Suppose instead \( i \neq j \). Thus \( J_i(S_i) \cap J_j(S_j) = \emptyset \) and this means that \( J_i(S_i) \cap J_j(S_j) \subseteq J(B^{i(1)}) \). But \( V_P(B_i) \cap V_P(B_j) \subseteq J(S_i) \cap J(S_j) \) and \( J(B^{i(1)}) \subseteq V_P(B^{i(1)}) \), thus \( V_P(B_i) \cap V_P(B_j) \subseteq V_P(B^{i(1)}) \). It follows that for every bag \( B \cup B^{i(1)} \) of \( T \) it holds \( V_P(B_i \cup B^{i(1)}) \cap V_P(B_j \cup B^{i(1)}) \subseteq V_P(B \cup B^{i(1)}) \).  

2. Peeling \( \Lambda_i \). We then focus on decomposing \( \Lambda_i = \langle S_i \cup J_i, E_i \rangle \). Note that \( d_u \leq 2 \) for all \( u \in S_i \) since we removed all \( u : d_u \geq 3 \) in the shattering phase.

Now, if \( |S_i| = 1 \), then \( T_i = \langle \{S_i\}, \emptyset \rangle \) is a d.t.d. of \( \Lambda_i \) of width 1. Assume then \( |S_i| > 1 \).

First, suppose any one of these three conditions holds:

1. \( d_u = 1 \) for some \( u \in S_i \)
2. \( J_i(u) = J_i(u') \) for some \( u, u' \in S_i \) with \( u \neq u' \)
3. \( d_v = 1 \) for some \( v \in J_i \)

Then we create \( T_i \) recursively as follows. If (1) holds, let \( u \) be the source with \( d_u = 1 \) and \( u' \neq u \) be any source with \( J_i(u) \cap J_i(u') \neq \emptyset \). If (2) holds, let \( u \) and \( u' \) be defined as above. If (3) holds, let \( u \) be the unique source such that \( J_i(u) = \{v\} \) and \( u' \neq u \) be any source with \( J_i(u) \cap J_i(u') \neq \emptyset \). Note that in any case \( u' \) must exist since \( |S_i| > 1 \) and \( \Lambda \) is connected. Let then \( \Lambda'_i = \Lambda_i \setminus \{u\} \), and assume we have a d.t.d. \( T'_i \) of \( \Lambda'_i \). Since \( u' \neq u \) then \( u' \in S_i \setminus \{u\} \), and thus for some \( B' \in T'_i \) it holds \( u' \in B' \). Create the bag \( B_u = \{u\} \) and set is as a child of \( B' \). We obtain a tree \( T_i \) where \( B_u \) is a leaf; and note that, by construction, for any \( u'' \in S_i \setminus \{u, u'\} \) it holds \( J_i(u) \cap J_i(u') \subseteq J_i(u'') \). This implies that \( T_i \) is a d.t.d. for \( \Lambda_i \). Then remove \( u \) from \( \Lambda_i \), as well as any \( v : d_v = 0 \). Repeat the process until either \( |S_i| = 1 \), in which case the base case above applies, or \( |S_i| > 1 \), in which case we move to the next paragraph.

![Figure 2](image-url) Above: example of a skeleton component \( \Lambda_i \). Below: the skeleton obtained from \( \Lambda_i \) after peeling (left), and its encoding as a core graph \( C \) (right).

3. Decomposing the core. Suppose the peeling phase stopped with \( |S_i| > 1 \). This implies conditions (1-3) violated, and in particular \( d_u = 2 \) for all \( u \in S_i \) and \( d_v \geq 2 \) for all \( v \in J_i \). We can then conveniently encode (what remains of) \( \Lambda_i \) as a simple graph where each node is a joint and each edge represents a distinct source. Formally, let \( C = (V_C, E_C) \) where \( V_C = J \) and \( E_C = \{e_u : u \in S \} \). We call this the core of \( \Lambda_i \) (see Figure 2); we drop the subscript \( i \) for readability since no ambiguity can arise. We shall now show that \( \Lambda_i =
Counting subgraphs via DAG tree decompositions

(S_\ell \cup J_\ell, E_\ell), from which C was obtained, has a d.t.d. T_\ell = (B_\ell, E_\ell) with s(T_\ell) \leq \frac{|E_\ell| + |V_\ell|}{4} + 2 and |B_\ell| \leq 4(|E_\ell| + V_\ell|). Since each e \in E_\ell encodes a distinct source, and each v \in V_\ell encodes a distinct joint, then |E_\ell| + |V_\ell| \leq k_\ell where k_\ell = |S_\ell \cup J_\ell|.

First, suppose that |V_\ell| \leq 4. Then C has an edge cover E_\ell of size 2. We then build T_\ell by setting E_\ell as root bag, and B_u = {u} for every u \in E_\ell \setminus E_\ell as child of E_\ell. Clearly s(T_\ell) = 2 \leq \frac{|E_\ell|}{4} + 2 and |B| \leq |E_\ell| \leq 4(|E_\ell| + V_\ell|).

Suppose instead |V_\ell| \geq 5. We then show one can build T_\ell via a tree decomposition of C, which together with a treewidth bound from [23], yields the following lemma (proof in Appendix A.4):

**Lemma 16.** In time $O(1.7549^{k_\ell} + \text{poly}(k_\ell))$ we can compute a d.t.d. $T_C = (B_C, E_C)$ of C such that $s(T_C) \leq \frac{|E_C|}{4} + 3$ and $|B_C| \leq 4(|E_C| + |V_C|)$.

Now, $\frac{|E_C|}{4} + 3 \leq \frac{|E_C| + |V_C|}{4} + 2$ holds if $1 \leq \frac{|E_C|}{4} + \frac{|V_C|}{2}$. One can check this holds since $|E_C| \geq |V_C|$ and $|V_C| \geq 5$.

Therefore in any case $s(T_C) \leq \frac{|E_C| + |V_C|}{4} + 2 \leq \frac{k_\ell}{4} + 2$. Note also that, in the peeling phase, for every bag added we did remove at least one distinct source from $A_i$, which therefore does not appear in $C$. This implies that $T_i = (B_i, E_i)$ built for $A_i$ satisfies $|B_i| \leq 4k_i$, and obviously $s(T_i) \leq \frac{k_i}{4} + 2$.

4. Assembling the tree. Let now $T_i : i = 1, \ldots, \ell$ be the d.t.d.’s for $A_1, \ldots, A_\ell$, and let $T = (B, E)$ be the d.t.d. for $P$ obtained from $T_1, \ldots, T_\ell$ as described in Lemma 15. By Lemma 15, $s(T) \leq |B^{(i)}| + \max_{i=1,\ldots, \ell} s(T_i)$, thus:

$$s(T) \leq |B^{(i)}| + \max_{i=1,\ldots, \ell} k_i + 2 \quad (4)$$

It is immediate to see that $k_i$ is a lower bound on the number of nodes in $P^{(i)}(U_i)$, and on the number of its arcs, too. However, from the shattering phase we know $P \setminus P^{(i)}$ has at least $4|B^{(i)}|$ nodes and at least $4|B^{(i)}|$ arcs. Similarly, from the core decomposition we know $P^{(i)}$ has at least $\sum_{i=1}^\ell k_i$ nodes and at least $\sum_{i=1}^\ell k_i$ arcs. Therefore $s(T) \leq \frac{k_i}{4} + 2$ and $s(T) \leq \frac{k_i}{4} + 2$. Moreover, since $\sum_{i=1}^\ell k_i \leq k$, again by Lemma 15 we get $|B| = \sum_{i=1}^\ell 4k_i \leq 4k$, as claimed by Theorem 13. Finally, by Lemma 16 the time to build each $T_i$ is $O(1.7549^{k_i})$, since the peeling phase takes poly$(k_i)$ = $O(1)$; the total time to build $T$ is then clearly $O(1.7549^k)$. The proof of Theorem 13 is complete.

3.4 Independence number and dag treewidth

We conclude this section by proving that the dag treewidth $s(H) = s_3(H)$ of an undirected graph $H$ is within constant facors of its independence number $\alpha(H)$. This means Theorem 1 holds (up to constants at the exponent) if one can replaces $s(H)$ by $\alpha(H)$.

**Lemma 17.** $s_3(H) = O(H)$.

**Proof.** Let $H$ be any pattern graph on $k$ nodes. To see that $s(H) = s_3(H) = O(H)$, just note that the source set of any oriented pattern is an independent set, and that $\alpha(H') \leq H'$ for any $H'$ obtained by adding edges or identifying nodes of $H$.

To prove $s_3(H) = \Omega(H)$ we exhibit a pattern $H'$ obtained from $H$ such that any of its acyclic orientations $P$ achieves $s(P) = \Omega(H)$. Let $I \subseteq V_H$ be an independent set of $H$ with $|I| = \Omega(H)$ and $|I| \mod 5 = 0$. Partition $I$ in $I_1, I_2$ where $|I_1| = \frac{2}{7}|I|$ and $|I_2| = \frac{2}{7}|I|$. On top of $I_1$ we virtually build a 3-regular expander $E = (I_1, E_e)$ of
linear treewidth $t(\mathcal{E}) = \Omega(|I_1|)$ \footnote{It is well-known that such expanders exist (see e.g., Proposition 1 and Theorem 5 of \cite{20}).}. Now, for each edge $uv \in E$ we choose a distinct node $e_{uv} \in I_2$ and add to $H[I]$ the edges $e_{uv}u$ and $e_{uv}v$. In other terms, $H[I]$ is now the 1-subdivision of the aforementioned expander. Let $H'$ be the pattern obtained. Note that $t(\mathcal{E}) = \Omega(|I_1|) = \Omega(|I|) = \Omega(\alpha(H))$.

Let now $P = (V_P, E_P)$ be any acyclically oriented graph obtained from $H'$ by setting $I_2$ as source set. We show that $s(P) \geq \frac{1}{2} (t(\mathcal{E}) + 1)$. Let $T$ be a d.t.d. of $P$. Build a tree decomposition $D$ for $\mathcal{E}$ by simply replacing each bag $B \in T$ with the bag $J(B)$. We shall prove $D$ is indeed a tree decomposition of $\mathcal{E}$ (see Definition \ref{def:tdg}). First, from point (2) of Definition \ref{def:td} we have $\cup_{B \in T} B = S_P$. It follows that $\cup_{J(B) \in D} J(B) = I_1$. This proves property (1). Second, by construction for every $e \in \mathcal{E}$ we have a node $u = u_e \in E_P$. Then, again from point (2) of Definition \ref{def:td} there exists $B \in T$ such that $u \in P$; and by construction of $D$ it holds $e = \{v, w\} \subseteq J(B)$. This proves property (2). Third, pick any $J(B_1), J(B_2), J(B_3) \in D$ such that $J(B_1)$ is on the unique path from $J(B_2)$ to $J(B_3)$ in $D$, and consider any node $v \in J(B_2) \cap J(B_3)$. There there exists $u \in B_2$ such that $v \in J(u)$, and $u' \in B_3$ such that $z \in J(u')$. Thus $v \in J(u) \cap J(u') \subseteq J(B_2) \cap J(B_3)$; but since $B_1$ is on the unique path from $B_2$ to $B_3$ in $T$, point (3) of Definition \ref{def:tdg} implies $v \subseteq J(B_1)$. This proves property (3). Hence $D$ is a tree decomposition of $\mathcal{E}$. Finally, by construction $|I_2| \leq 2|J(I_2)|$. Then by definitions \ref{def:td} and \ref{def:tdg} we have $t(\mathcal{E}) \leq 2s(P) - 1$, that is, $s(P) \geq \frac{1}{2} (t(\mathcal{E}) + 1)$. But $t(\mathcal{E}) = \Omega(\alpha(H))$, thus $s(P) = \Omega(\alpha(H))$.

\section{Lower bounds}

We prove the lower bound of Theorem \ref{thm:lb}

\begin{theorem}
For any function $a : [k] \rightarrow [1, k]$ there exists an infinite family of patterns $\mathcal{H}$ such that (1) $s(H) = \Theta(a(k))$ for each $H \in \mathcal{H}$, and (2) if there exists an algorithm that computes $\text{ind}(H, G)$ or $\text{sub}(H, G)$ in time $f(d, k) \cdot n^o(a(k) \ln o(k))$ for all $H \in \mathcal{H}$ where $d$ is the degeneracy of $G$, then ETH fails.
\end{theorem}

\textbf{Proof.} We reduce the problem of counting cycles in an arbitrary graph to the problem of counting a gadget pattern on $k$ nodes and dag treewidth $O(s(k))$, where $s(k) = a(k)$, in a d-degenerate graph.

The pattern is the following. Consider a simple cycle on $k_0 \geq 3$ nodes. Choose an integer $d = d(k) \geq 2$ with $d(k) \in \Omega(\frac{k_0}{\ln(k_0)})$. For each edge $e = uv$ of the cycle create a clique $C_e$ on $d - 1$ nodes; delete $e$ and connect both $u$ and $v$ to every node of $C_e$. The resulting pattern $H$ has $dk_0 = k$ nodes. Let us prove $s(H) \leq k_0$. This implies $s(H) = O(s(k))$ since $k_0 = \frac{k}{d} \in O(s(k))$. Consider again the generic edge $e = uv$. Since $C_e \cup u$ is itself a clique, it has independent set size 1; and thus in any orientation $H_\sigma$ of $H$, $C_e \cup u$ contains at most one source. Applying the argument to all $e$ shows $S(H_\sigma) \leq k_0$, and since $s(H_\sigma) \leq |S(H_\sigma)|$, we have $s(H_\sigma) \leq k_0$. Note any $H'_\sigma$ obtained from $H_\sigma$ by adding edges or identifying nodes has at most $k_0$ roots, too. Hence $s(H) \leq k_0$.

Now consider the task of counting the cycles of length $k_0 \geq 3$ in a simple graph $G_0$ on $n_0$ nodes and $m_0$ edges. We replace each edge of $G_0$ as described above. The resulting graph $G$ has $n = m_0(d - 1) + n_0 = O(dn_0^2)$ nodes, has degeneracy $d$, and can be built in $\text{poly}(n_0)$ time. Note that every $k_0$-cycle of $G_0$ is univocally associated to a (an induced) copy of $H$ in $G$. Suppose then there exists an algorithm that computes $\text{ind}(H, G)$ or $\text{sub}(H, G)$ in time $f(d, k) \cdot n^o(s(H) \ln s(H))$. Since $s(H) \leq k_0$, $k = f(d, k_0)$, $n = O(dn_0^2)$, and $d = f(k_0)$, the
running is time $f(d, k_0) \cdot n^{o(k_0/\ln k_0)}$. This implies one can count the number of $k_0$-cycles in $G$ in time $f(k_0) \cdot n^{o(k_0/\ln k_0)}$. It remains to invoke the following result:

**Theorem 19** ([11], Theorem I.2). The following problems are $\#W[1]$-hard and, assuming ETH, cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ for any computable function $f$: counting (directed) paths or cycles of length $k$, and counting edge-colorful or uncolored $k$-matchings in bipartite graphs.

The proof is complete.

Note that, since $s(H) = \Theta(\alpha(H))$ by Lemma 17, the lower bound holds if one replaces $s(H)$ by $\alpha(H)$ in the statement.

### 5 Bounds parameterized by $\Delta$

We give a bound for computing $\text{hom}(H, G)$ parameterized by the maximum degree $\Delta$ of $G$. Theorem 5 follows by the inclusion-exclusion arguments of Lemma 10. This improves the dependence on $\Delta$ of a recent $\tilde{O}((7\Delta)^2 k n)$ bound for computing $\text{ind}(H, G)$ by Patel et al. [29]. While their algorithm is based on graph polinomials, our relies on simple combinatorial arguments.

**Theorem 20.** For any $k$-node pattern $H$ one can compute $\text{hom}(H, G)$ in time $O(B_k \Delta^{k-1} n)$, where $B_k < \left(\frac{0.792k}{\ln(k+1)}\right)^k$ is the $k$-th Bell number.

**Proof.** We first build the set of patterns obtainable by identifying subsets of $V_H$ (loops and multiple edges are ignored). To this end we enumerate the partitions $\theta$ of $V_H$, which takes time $O(B_k)$ where $B_k < \left(\frac{0.792k}{\ln(k+1)}\right)^k$ is the $k$-th Bell number [17, 4]. For each $\theta$, in time $O(1)$ we obtain $H' = H/\theta$ and insert it into a sorted list if not present. The running time is $O(B_k)$. Next, for each connected quotient pattern $H'$ we choose a rooted spanning tree $T = T(H')$ and list all homomorphisms $\phi : T \to G$, which takes time $O(\Delta^{j-1} n)$ if $T$ has $j$ nodes. For each such $\phi$ we check whether $\phi : H' \to G$ is a homomorphism as well (in time $O(1)$, by checking the image of its arcs) and increase the counter of $H'$. The running time is thus $O(B_k \Delta^{k-1} n)$. At this point we know $\text{hom}(H', G)$ for all connected quotient patterns $H'$. For each $H'$ not connected, we use the identity $\text{hom}(H', G) = \prod_{i=1}^{c} \text{hom}(H'_i, G)$, where $H'_1, \ldots, H'_c$ are the connected components of $H'$; the overall time is again $O(B_k)$. The overall running time is thus $O(B_k \Delta^{k-1} n)$.

### 6 Conclusions

We have shown how one can parameterize the complexity of subgraph counting by the sparsity of the host graph and, in particular, how to harness degeneracy orientations to build faster exact counting algorithms. As a byproduct, we have improved or generalized classic bounds. It would be interesting to know if our main technical tool, the tree decomposition for directed acyclic graphs, can be helpful in other settings.
A.1 Tree decomposition and treewidth of a graph

We recall the definition of tree decomposition of an undirected graph $G = (V, E)$. Many slightly different but equivalent definitions exist; we adopt one similar to [12], Ch. 12.3:

**Definition 21.** Given a graph $G = (V, E)$, a tree decomposition of $G$ is a tree $D = (V_D, E_D)$ such that each node $X \subseteq V_D$ is a subset $X \subseteq V$, and that:
1. $\bigcup_{X \subseteq V_D} X = V$
2. for every edge $e = \{u, v\} \in G$ there exists $X \in D$ such that $u, v \in X$
3. $\forall X, X', X'' \in V_T$, if $X$ lies on the unique path $D(X', X'')$ then $X' \cap X'' \subseteq X$

The width of a tree decomposition $T$ is $t(T) = \max_{X \subseteq V_T} |X| - 1$. The treewidth $t(G)$ of a graph $G$ is the minim $t(T)$ over all tree decompositions $T$ of $G$.

A.2 Proof of Theorem 8

We will go through a series of lemmata and intermediate results. We first need an analogue of the path separator property for tree decompositions.

**Lemma 22.** Let $T$ be a d.t.d. and let $B_1, \ldots, B_l$ be the children of $B$ in $T$. Then for all $i = 1, \ldots, l$:
  a. $V_T(\Gamma[B_i]) \cap V_T(\Gamma[B_j]) \subseteq V_T(B)$ for all $j \neq i$
  b. for any arc $(u, u') \in P(\Gamma[B])$, if $u \in V_T(\Gamma[B_i]) \setminus V_T(B)$ then $u' \in V_T(\Gamma[B_i])$
  c. for any arc $(u', u) \in P(\Gamma[B])$, if $u \in V_T(\Gamma[B_i]) \setminus V_T(B)$ then $u' \in V_T(\Gamma[B_i])$

**Proof.** For brevity let $P_B = P(B)$, $V_B = V_P(B)$, and for $i = 1, \ldots, l$ let $P_i = P(\Gamma[B_i])$ and $V_i = V_P(\Gamma[B_i])$. Let us prove (a). Suppose for some $j \neq i$ there exists $u \in (V_i \setminus V_B) \cap (V_j \setminus V_B)$. This implies $u \in V_P(Y) \cap V_P(Y')$ for some $Y \in T(B_i)$ and $Y' \in T(B_j)$. But then point (3) of Definition 6 implies $u \in V_P(B) = V_B$, a contradiction.

Let us prove (b). Suppose $(u, u') \in P(\Gamma[B])$ with $u \in V_i \setminus V_B$ and $u' \notin V_i$. Then $u'$ is reachable from $u$, hence $u' \in V_i$, a contradiction. Suppose instead $(u', u) \in P(\Gamma[B])$ with $u \in V_i \setminus V_B$ and $u' \notin V_i$. Then $u$ is reachable from $u'$. But if $u' \in V_B$ then $u \in V_B$, a contradiction; it remains to note that if instead $u' \in V_j$ for some $j \neq i$, by point (3) of Definition 6 we still have $u \in V_B$.

In a nutshell, Lemma 22 says that $V_P(B)$ is a separator for the sub-patterns $P(\Gamma[B_i])$ in $P$. Thanks to this, we can compute hom$(P(\Gamma[B]))$ by combining hom$(P(\Gamma[B_i]))$, ..., hom$(P(\Gamma[B_i]))$ appropriately.

Now pick any $\phi_B : P(B) \to G$, and for each $i = 1, \ldots, l$ pick a $\phi_i : P(\Gamma[B_i]) \to G$ that respects $\phi_B$. Define $\phi = \phi_B \phi_1 \ldots \phi_l : P(\Gamma[B]) \to G$ as follows: $\phi(u) = \phi_B(u)$ if $u \in V_P(B)$, and $\phi(u) = \phi_i(u)$ if $u \in V_P(\Gamma[B_i]) \setminus V_P(B)$. By Lemma 22 we can show that any homomorphism $\phi : P(\Gamma[B]) \to G$ can be written in this form.

**Lemma 23.** Fix any $\phi_B : P(B) \to G$. Let $\Phi(\phi_B) = \{\phi : P(\Gamma[B]) \to G | \phi \text{ respects } \phi_B\}$, and for $i = 1, \ldots, l$ let $\Phi_i(\phi_B) = \{\phi : P(\Gamma[B_i]) \to G | \phi \text{ respects } \phi_B\}$. Then there is a bijection between $\Phi(\phi_B)$ and $\Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B)$, and therefore:

$$\text{hom}(P(\Gamma[B]), G, \phi_B) = \prod_{i=1}^l \text{hom}(P(\Gamma[B_i]), G, \phi_B)$$ (5)
Counting subgraphs via DAG tree decompositions

Proof. Suppose \( \phi : P(\Gamma[B]) \to G \) respects \( \phi_B \). Let \( \phi' \) be the restriction of \( \phi \) to \( P(V_P(\Gamma[B]) \setminus V_P(B)) \). Note that \( \phi = \phi_B \phi' \). Now, by Lemma \( 22 \) \( V_P(\Gamma[B]) \setminus V_P(B) \) is the union of node-disjoint subsets \( V_P(\Gamma[B]) \setminus V_P(B) : i = 1, \ldots, l \). Therefore \( \phi' \) can be written as \( \phi'_1 \cdots \phi'_l \) where for \( i = 1, \ldots, l \) the map \( \phi'_i \) is a homomorphism \( \phi'_i : P(V_P(\Gamma[B]) \setminus V_P(B)) \to G \). For each \( \phi'_i \) let then \( \phi_i = \phi'_i \phi_B \), and we have \( \phi = \phi_B \phi_1 \cdots \phi_l \). (Note that the \( \phi_i \) are uniquely determined by \( \phi \) and the d.t.d. \( T \).) It is immediate to see the converse holds as well: every combination \( \phi_1, \ldots, \phi_l \) of homomorphisms respecting \( \phi_B \) gives a unique homomorphism \( \phi = \phi_B \phi_1 \cdots \phi_l : P(\Gamma[B]) \to G \) that respects \( \phi_B \). Thus there is a bijection between \( \Phi(\phi_B) \) and \( \Phi_1(\phi_B) \times \ldots \times \Phi_l(\phi_B) \). ▶

We can now describe our dynamic programming algorithm, HomCount, to compute \( \hom(P(\Gamma[B]), G) \). First, we need an easy but crucial bound on the cost of enumerating all homomorphisms of a subpattern \( P(S) \) onto \( G \).

Lemma 24. Given any \( B \subseteq S \), the set of homomorphisms \( \Phi = \{ \phi : P(B) \to G \} \) can be enumerated in time \( O(d^{k-1} |B| n^k(B)) \).

Proof. Let \( s = |S| \) and \( S = \{ u_1, \ldots, u_s \} \). Pick a rooted spanning forest \( \{ T_1, \ldots, T_l \} \) of \( P(S) \), so that each \( T_i = (V_i, A_i) \) is a subtree of \( P(u_i) \) rooted at \( u_i \), all nodes of \( T_i \) are reachable from \( u_i \) in \( T_i \), and \( U_i = V_P(B) \). Let \( \Phi_i \) be the set of homomorphisms from \( T_i \) to \( G \), and let \( \Psi = \Phi_1 \times \ldots \times \Phi_s \). We see each \( \psi = (\phi_1, \ldots, \phi_s) \in \Psi \) as the map \( \psi : V_P(S) \to G \) defined by combining \( \phi_1, \ldots, \phi_s \) in the straightforward way. Note that \( \Phi \subseteq \Psi \), and given any \( \psi \in \Psi \) we can determine if \( \psi \in \Phi \) in time \( O(k^2) = O(1) \) by checking its arcs in \( G \).

We are then left with enumerating \( \Psi = \Phi_1 \times \ldots \times \Phi_s \), which boils down to enumerating \( \Phi_i \). To this end we pick in turn each \( v \in G \). Then, we enumerate all \( \phi_i \in \Phi_i \) that map \( s_i \) to \( v \). Note that such \( \psi_i \) are at most \( d^{|V_i|-1} \), since for each arc \( (x, y) \in T_i \), once we have fixed the image of \( x \) we have at most \( d \) choices for \( y \) in \( G \), and the image of the root \( r_i \) is fixed at \( v \). Therefore \( |\Psi| \leq \prod_{i=1}^s d^{|V_i|-1} n \leq d^{k-1} n^s \). Now recursively split \( T_i \) into two disjoint subtrees \( T_i' \) and \( T_i'' \), where \( T_i' \) is rooted at \( r_i \), and \( T_i'' \) is rooted at a child \( r_i'' \) of \( r_i \). Then enumerate recursively all combinations of \( \phi' : T_i' \to G \) such that \( \phi'(r_i) = v \) and, for all outgoing arcs \( (v, w) \in G \), of \( \phi'' : T_i'' \to G \) such that \( \phi''(r_i'') = w \). The overall time to enumerate \( \Phi_i \) is then \( O(d^{|V_i|-1}|V_i|) \), and the total time to enumerate \( \Psi \) is \( O(d^{k-1} n^s) \). The overall time to enumerate \( \Phi \) is therefore \( O(d^{k-1} n^s) \). ▶

The programming goes bottom-up starting from the leaves of a tree decomposition \( T \) of \( P \), but we write it recursively for readability. The key step exploits Lemma 23 to compute \( |\Phi(\phi_B)| = \prod_{i=1}^s |\Phi_i(\phi_B)| \). The counters \( c(\ldots) \) are dictionaries with default value 0.

We can finally prove:

Lemma 25. HomCount\( P(T, B) \) returns a dictionary \( c(\cdot) \) satisfying \( c(\phi_B) = \hom(P(\Gamma[B]), \phi_B) \) for every \( \phi_B : P(B) \to G \). The running time of HomCount\( P(T, B) \) is \( O(|B| d^{k-1} n^k(T) \ln n) \).

Proof. First, the correctness. If \( B \) is a leaf of \( T \) then \( P(B) = P(\Gamma[B]) \), and indeed we set \( c(\phi_B) = 1 \) for each \( \phi_B : P(B) \to G \); the thesis follows. Suppose instead \( B \) is an internal node of \( T \) and let \( B_1, \ldots, B_l \) be its children. Assume by inductive hypothesis that the claim holds for HomCount\( P(T, B_i) \) for all \( i = 1, \ldots, l \). Then the dictionary \( c(B_i, \cdot) \) computed at line 7 satisfies \( c(B_i, \phi_i) = \hom(P(\Gamma[B_i]), \phi_i) \) for every \( \phi_i : P(B_i) \to G \). Let \( \Phi_i \) be the set of homomorphisms from \( P(\Gamma[B_i]) \) to \( G \); note that \( \hom(P(\Gamma[B_i]), \phi_i) = |\Phi_i(\phi_i)| \). Then the loop at lines 8-10 sets:

\[
c'(B_i, \phi_i, B) = \sum_{\phi_B : P(B) \to G} |\Phi_i(\phi_i)| = |\Phi_i(\phi_B)|
\]

(6)
Algorithm 1 HomCount\((P, T, B)\)

1: if \(B\) is a leaf then 
2: for every homomorphism \(\phi_B : P(B) \to G\) do 
3: \(c(\phi_B) = 1\) 
4: else 
5: let \(B_1, \ldots, B_l\) be the children of \(B\) in \(T\) 
6: for \(i = 1, \ldots, l\) do 
7: \(c(B_i, \cdot) \leftarrow \text{HomCount}(P, T, B_i)\) 
8: for every key \(\phi\) in \(c(B_i, \cdot)\) do 
9: let \(\phi_{i,B}\) be the restriction of \(\phi\) to \(V_P(B) \cap V_P(\Gamma[B_i])\) 
10: \(c'(B_i, \phi_{i,B}) += c(B_i, \phi_i)\) 
11: for every homomorphism \(\phi_B : P(B) \to G\) do 
12: for \(i = 1, \ldots, l\) let \(\phi_{i,B}\) be the restriction of \(\phi_B\) to \(V_P(B) \cap V_P(\Gamma[B_i])\) 
13: \(c(\phi_B) = \prod_{i=1}^l c'(B_i, \phi_{i,B})\) 
14: return \(c(\cdot)\)

where the second equality follows from a trivial counting argument. Finally, consider the loop at lines 11–13. Note that \(\Phi_i(\phi_{i,B}) = \Phi_i(\phi_B)\) since for any \(\phi \in \Phi_i\) the restriction to \(V_P(B) \cap V_P(\Gamma[B_i])\) equals its restriction to \(V_P(B)\). Thus for each \(\phi_B : P(B) \to G\) we are setting \(c(B, \phi_B) = |\Phi_1(\phi_B)| \cdots |\Phi_l(\phi_B)| = |\Phi_1(\phi_B) \times \cdots \times \Phi_l(\phi_B)|\). But by Lemma 23 \(|\Phi_1(\phi_B) \times \cdots \times \Phi_l(\phi_B)| = \text{hom}(P(\Gamma[B]), \phi_B)\), and the inductive step is proven.

Now to the running time. We can represent a homomorphism \(\phi\) as a tuple of nodes of \(G\); and for each \(i = 1, \ldots, l\) in time \(O(\text{poly}(k))\) we can precompute the indices of the tuple of the restriction of \(\phi\) to \(V_P(B) \cap V_P(\Gamma[B_i])\). Now, if \(B\) is a leaf in \(T\), then the running time is dominated by the cycle at line 2. The cycle performs \(O(d^{k-|B|}ln^{|B|})\) iterations (see Lemma 24), and each iteration takes \(O(ln)\) to update \(c(B, \phi)\). This gives a running time of \(O(d^{k-|B|}ln^{|B|}ln)\). If \(B\) is an internal node of \(T\), then the time taken by Lines 8–10 is dominated by the recursive calls at line 7. The cycle at line 11 follows the analysis above, but instead of \(O(ln)\) we spend \(O(lnln)\) to access the \(l\) dictionary entries and perform the multiplication. This gives a running time of \(O(l(d^{k-|B|}ln^{|B|}ln)ln)\). The total running time excluding recursive calls is then \(O(l(d^{k-|B|}ln^{|B|}lnln)ln)\) as well. The thesis follows by recursing on the subtrees of \(T(B)\) and noting that the sum of the number of children of all bags is at most \(|B|\).

A.3 Proof of Lemma 10

From directed to undirected.

Given an undirected pattern \(H = (V_H, E_H)\), let \(\sigma : E_H \to \{0, 1\}\) be an arbitrary orientation of its edges (if \(\sigma((u, v)) = 1\) then the edge is oriented as \((u, v)\)). We denote by \(\Sigma\) the set of all distinct acyclic orientations of the edges of \(H\), and for each \(\sigma \in \Sigma\) we let \(H_{\sigma}\) be the oriented pattern obtained by applying \(\sigma\) to \(H\). First, we need to prove:

\[\text{hom}(H, G) = \sum_{\sigma \in \Sigma} \text{hom}(H_{\sigma}, G).\]

**Proof.** Let \(\Phi = \{\phi : H \to G\}\) be the set of homomorphisms from \(H\) to \(G\). In the same way define \(\Phi_P = \{\phi_P : P \to G\}\) for any \(P = H_{\sigma}\). (Note that \(\phi\) ignores the orientation of \(G\), while \(\phi_P\) must match it.) We partition \(\Phi\) as follows. For each \(\sigma \in \Sigma\), let \(\Phi_{\sigma} = \{\phi : H \to G : \forall e \in E_H : \sigma(\phi(e)) = \sigma(e)\}\). In words, \(\Phi_{\sigma}\) are the homomorphisms such that the image of
$H$ in $G$ “induces” the orientation $\sigma$ of $H$. First, note that $\Phi_\sigma \cap \Phi_{\sigma'} = \emptyset$ whenever $\sigma \neq \sigma'$. Indeed, if $\sigma \neq \sigma'$ then for some $e \in E_H$ we have $\sigma(e) \neq \sigma'(e)$. Then $\phi \in \Phi_\sigma \cap \Phi_{\sigma'}$ implies $\sigma(\phi(e)) = \sigma(e)$ and $\sigma(\phi(e)) = \sigma'(e)$, thus $\sigma(e) = \sigma'(e)$, a contradiction. Second, note that $\Phi = \cup_{\sigma \in \Sigma} \Phi_\sigma$. Therefore the $\Phi_\sigma$ form a partition of $\Phi$ and $\sum_{\sigma \in \Sigma} |\Phi_\sigma|$. Finally, note that $\Phi_\sigma$ is in a one-to-one relationship between $\Phi_P = \{ \phi : P \to G \}$ where $P = H_\sigma$. Indeed, any $\phi \in \Phi_\sigma$ identifies simultaneously $P = H_\sigma$ and a homomorphism $\phi : P \to G$; the converse holds, too. Thus $\text{hom}(H_\sigma, G) = |\Phi_\sigma|$ and the proof is complete. ▷

Thus, once we know $\text{hom}(P, G)$ for all the acyclically oriented versions of $H$ we can compute $\text{hom}(H, G)$ at an additional cost $O(k!)$. From homomorphisms to non-induced copies.

Denote now by $\theta \in \Theta$ a generic equivalence relationship on $V_H$. Let $H/\theta$ be the quotient graph obtained from $H$ by identifying the nodes in the same equivalence class and then removing loops and multiple edges. By Equation 15 of [17]:

$$\text{inj}(H, G) = \sum_{\theta \in \Theta} \mu(\theta) \hom(H/\theta, G)$$  (7)

where $\mu(\theta) = \prod_{A \in \Theta} (-1)^{|A|-1}(|A| - 1)!$, where $A$ runs over the equivalence classes in $\theta$. It is known that $|\Theta| = 2^{O(k \ln k)}$ (see e.g. [11]), and clearly for each $\theta$ we can compute $\mu(\theta)$ in $O(\text{poly}(k))$. Thus, once we know $\text{hom}(H', G)$ for all the patterns $H'$ obtainable by identifying nodes of $H$, we can compute $\text{inj}(H, G)$ in time $2^{O(k \ln k)}$. In the same way we can finally compute $\text{sub}(H, G)$, since $\text{sub}(H, G) = \text{inj}(H, G)/\text{aut}(H)$, where $\text{aut}(H)$ is the number of automorphisms of $H$, which can be computed in time $2^{O(k \ln k)}$ [26].

From non-induced to induced.

Finally, let $D(H)$ be the set of all graphs obtainable by adding one or more edges to $H$. Then from Equation 14 of [17]:

$$\text{ind}(H, G) = \sum_{H' \in D(H) \cup \{H\}} (-1)^{|E_{H'} \setminus E_H|} \text{inj}(H', G)$$  (8)

and thus once we know $\text{inj}(H', G)$ for all $H' \in D(H) \cup \{H\}$ we can compute $\text{ind}(H, G)$ in time $O(|D(H)| + 1)$. Note that, if $H$ has $\binom{k}{2} - c$ edges, then $|D(H)| \leq 2^c$; in general we have $|D(H)| \leq 2^{k^2}$.

Disconnected patterns.

Suppose $H$ is formed by $h > 1$ connected components $H_1, \ldots, H_h$. Simply note that $\text{hom}(H, G) = \prod_{i=1}^h \text{hom}(H_i)$. All arguments above thereafter apply unchanged.

A.4 Proof of Lemma 16

Let us start by showing how a tree decomposition for $C$ can be turned into a d.t.d. for $C$ in time $O(\text{poly}(k_i))$. Formally:

Lemma 27. Let $D$ be a tree decomposition of $C = (V_C, E_C)$. For every $v \in V_C$ pick an arbitrary incident edge $u_v = \{v, z\} \in E_C$. Now replace each bag $Y \in D$ by $B(Y) = \{u_v : v \in Y\}$, and for every $u \in S \setminus \cup_{Y \in D} B(Y)$, choose a bag $B(Y) : J(u) \subseteq Y$, and set the bag $B_u = \{u\}$ as child of $B(Y)$. Then $T$ is a d.t.d. of $C$, and $s(T) \leq t(D) + 1$. ▷
We now invoke previous results on tree decompositions and treewidth. First, by [19], as usual we denote by $T$ the right-hand expression is just $E_C$.

The rest of the proof deals with point (3). First, if we did set $B_u$ as child of $B_Y$, then by construction $j(B_u) \subseteq j(B_Y)$. Thus we can ignore any such $B_u$ and focus on $B = B(Y)$, $B' = B(Y')$, and $B'' = B(Y'')$ for some $Y, Y', Y'' \in D$. Suppose then $B(Y) \in T(B(Y'), B(Y''))$ and that, by contradiction, there exists $v \in j(B(Y')) \cap j(B(Y''))$ such that $v \notin j(B(Y))$. Note that, by construction, we must have put some $u'$ with $e_{u'} = \{v, z'\}$ in $B'$ and some $u''$ with $e_{u''} = \{v, z''\}$ in $B''$, for some $z', z'' \in V_i$. Moreover, $Y' \cap \{v, z'\} \neq \emptyset$ and $Y'' \cap \{v, z''\} \neq \emptyset$, else we could not have put $u' \in B''$ and $u'' \in B''$. Finally, bear in mind that $v \notin Y$ and $u', u'' \notin B(Y)$, for otherwise $v \in j(B(Y))$, contradicting the hypothesis. Now we consider three cases. We make repeatedly use of properties (2) and (3) of Definition 21.

As usual we denote by $D(Y, Y')$ the unique path from $Y$ to $Y'$ in $D$.

**Case 1.** $v \in Y'$ and $v \in Y''$. But $v \notin Y' \cap Y''$ and thus $v \in Y$, a contradiction.

**Case 2.** $v \in Y'$ and $v \notin Y''$. Then $z' \in Y''$ and $u''$ with $e_{u''} = \{v, z''\}$ is the edge chosen to cover $z''$, else we would not put $u'' \in B_Y$. Moreover there must be $\hat{Y} \in D$ such that $e_{u''} = \{v, z''\} \subseteq \hat{Y}$. For the sake of the proof root $D$ at $Y$, so $Y'$ and $Y''$ are in distinct subtrees. If $\hat{Y} \cap Y'$ and $\hat{Y} \cap Y''$ are in the same subtree then $Y \in D(Y', \hat{Y})$, but $v \notin Y' \cap \hat{Y}$ and thus $v \notin Y$, a contradiction. Otherwise $Y \in D(Y'', \hat{Y})$, and since $z'' \in Y'' \cap \hat{Y}$ then $z'' \subseteq Y$ and then $v'' \in B(Y), a contradiction.

**Case 3.** $v \notin Y'$ and $v \notin Y''$. Then $z' \in Y'$, $z'' \in Y''$, and $u', u''$ are the sources chosen to cover respectively $z'$, $z''$. Moreover there must be $\hat{Y}, \hat{Y}' \in D$ such that $e_{u'} = \{v, z'\} \subseteq \hat{Y}$ and $e_{u''} = \{v, z''\} \subseteq \hat{Y}'$. Root again $D$ at $Y$. If $Y \in D(\hat{Y}, \hat{Y}')$ then since $v \in \hat{Y} \cap \hat{Y}'$ it holds $v \in Y$, a contradiction. Otherwise $\hat{Y}, \hat{Y}'$ are in the same subtree of $D$. If the subtree is the same of $Y''$, then $Y \in D(Y', \hat{Y})$, and $z \in Y' \cap \hat{Y}$ and thus $z' \notin Y$ and thus $u' \in B(Y)$, a contradiction. Otherwise we have $Y \in D(Y'', \hat{Y})$; but $z'' \in Y'' \cap \hat{Y}$, thus $z'' \notin Y$ and $u'' \in B(Y)$, again a contradiction.

We now invoke previous results on tree decompositions and treewidth. First, by [19], Theorem 5.23-5.24, we can compute a minimum-width tree decomposition of an $n$-node graph in time $O(1.7549^n)$; and by [19] Lemma 5.16 we can transform such a decomposition to contain at most $4n$ bags, leaving its width unchanged, in time $O(n)$. Therefore in time $O(1.7549^{|V_C|} + |V_C|)$ we can build a minimum-width tree decomposition $D$ on at most $4|V_C|$ bags for $G = (V_C, E_C)$. It remains to bound the width of $D$. To this end we invoke the following treewidth bound from [23]:

**Theorem 28** (Thm. 2 of [23]). The treewidth of a graph $G = (V, E)$ is at most $\frac{|E|}{5} + 2$.

Hence our $D$ satisfies $t(D) \leq \frac{|E_C|}{5} + 2$. By the above Lemma 27, then, we can produce a d.t.d. $T$ for $C$ such that $s(T) \leq \frac{|E_C|}{5} + 3$. Note that the number of bags in $T$ is bounded by $4|V_C| + |E_C| \leq 4(|V_C| + |E_C|)$, since we add at most $|E_C|$ bags to those of $D$. The total running time is $O(1.7549^{k_t} + \text{poly}(k_t))$ since $k_t \leq |V_C|$. The proof is complete.
References

1. N. Alon, P. Dao, I. Hajarasouliha, F. Hormoziadi, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):241–249, Jul 2008.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.
4. Daniel Berend and Tamir Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Math. Statistics*, 30(2):185–205, 2010.
5. Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting subgraphs via packings faster than meet-in-the-middle time. In *Proc. of ACM-SIAM SODA*, pages 594–603, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.
6. Christian Borgs, Jennifer Chayes, László Lovász, Vera T. Sós, and Katalin Vesztergombi. *Counting Graph Homomorphisms*, pages 315–371. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
7. Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216 – 231, 2005.
8. Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346 – 1367, 2006.
9. Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
10. Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proc. of ACM STOC*, pages 210–223, 2017.
11. Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *Proc. of IEEE Focs*, pages 130–139, Washington, DC, USA, 2014. IEEE Computer Society.
12. Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition, 2017.
13. David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994.
14. David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proc. of ACM-SIAM SODA*, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.
15. David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation*, pages 403–414, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
16. David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, pages 364–375, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
17. M. C. Er. A fast algorithm for generating set partitions. *The Computer Journal*, 31(3):283–284, 1988.
18. Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015.
19. Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 1 edition, 2010.
20. Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory, Series B*, 99(1):218 – 228, 2009.
21. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Proc. of IEEE Focs*, pages 653–662. IEEE, 1998.
22 Shweta Jain and C. Seshadhri. A fast and provable method for estimating clique counts using Turán’s theorem. In Proc. of WWW, pages 441–449, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

23 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Algorithms based on the treewidth of sparse graphs. In International Workshop on Graph-Theoretic Concepts in Computer Science, pages 385–396. Springer, 2005.

24 Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations and matrix multiplication. In Proc. of ACM-SIAM SODA, pages 1468–1476, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics.

25 François Le Gall. Powers of tensors and fast matrix multiplication. In Proc. of ISSAC, pages 296–303, New York, NY, USA, 2014. ACM.

26 Rudolf Mathon. A note on the graph isomorphism counting problem. Information Processing Letters, 8(3):131 – 136, 1979.

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

28 Jaroslav Nešetril and Svatopluk Poljak. On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae, 026(2):415–419, 1985.

29 Viresh Patel and Guus Regts. Computing the number of induced copies of a fixed graph in a bounded degree graph. Algorithmica, Sep 2018.

30 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In Proc. of WWW, pages 1431–1440, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

31 B. Reed. Introducing directed tree width. Electronic Notes in Discrete Mathematics, 3:222 – 229, 1999. 6th Twente Workshop on Graphs and Combinatorial Optimization.

32 Ahmet Erdem Sariyüce and Ali Pinar. Peeling bipartite networks for dense subgraph discovery. In Proc. of ACM WSDM, pages 504–512, New York, NY, USA, 2018. ACM.

33 Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. ACM Trans. Web, 11(3):16:1–16:27, July 2017.

34 Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In Proc. of WWW, pages 1451–1460, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

35 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. SIAM J. Comput., 42(3):831–854, 2013.

36 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In Proc. of ACM KDD, pages 555–564, New York, NY, USA, 2017. ACM.