Faster Subgraph Counting in Sparse Graphs

Marco Bressan
Department of Computer Science, Sapienza University of Rome, Italy
bressan@di.uniroma1.it

Abstract

A fundamental graph problem asks to compute the number of induced copies of a k-node pattern graph H in an n-node graph G. The fastest algorithm to date is still the 35-years-old algorithm by Nešetřil and Poljak [28], with running time $f(k) \cdot O(n^{\omega \lfloor \frac{k}{3} \rfloor + 2})$ where $\omega \leq 2.373$ is the matrix multiplication exponent. In this work we show that, if one takes into account the degeneracy d of G, then the picture becomes substantially richer and leads to faster algorithms when G is sufficiently sparse. More precisely, after introducing a novel notion of graph width, the *DAG-treewidth*, we prove what follows. If H has DAG-treewidth $\tau(H)$ and G has degeneracy d, then the induced copies of H in G can be counted in time $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$; and, under the Exponential Time Hypothesis, no algorithm can solve the problem in time $f(d,k) \cdot n^{o(\tau(H)/\ln \tau(H))}$ for all H. This result characterises the complexity of counting subgraphs in a d-degenerate graph. Developing bounds on $\tau(H)$, then, we obtain natural generalisations of classic results and faster algorithms for sparse graphs. For example, when $d = O(\text{poly} \log(n))$ we can count the induced copies of any H in time $f(k) \cdot \tilde{O}(n^{\lfloor \frac{k}{4} \rfloor + 2})$, beating the Nešetřil-Poljak algorithm by essentially a cubic factor in n.

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

Keywords and phrases subgraph counting, tree decomposition, degeneracy

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.6

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

Funding Marco Bressan is supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award "ALL4AI", and by the "Dipartimenti di Eccellenza 2018-2022" grant awarded to the Department of Computer Science of the Sapienza University of Rome.

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 copies of H in G. This problem is at the heart of many algorithmic applications but, unfortunately, is largely intractable. The fastest algorithm known has running time $O(n^{\omega \lfloor \frac{k}{3} \rfloor + 2})$ where ω is the matrix multiplication exponent [28]; and the margin to improve the dependence on k is limited, since under the Exponential Time Hypothesis [21] $n^{\Omega(k)}$ operations are required even just to detect a clique [8, 9]. The picture changes, however, if we make additional assumptions on G. A natural assumption is that G be *sparse*, as is often the case in practice. Under certain notions of sparsity, indeed, it is known that subgraph counting becomes tractable: for example, any H can be counted in time $f(k) \cdot O(n)$ if G has bounded maximum degree $\Delta(G) = O(1)$ [29]. Alternatively, any H can be counted in time $f(k) \cdot O(n)$ if G has bounded treewidth $\mathrm{t}(G) = O(1)$, as a consequence of Courcelle's theorem [27]. Similar bounds can be proved when G is planar [14]. These assumptions are much stronger than just having $O(1)$ average degree, and often do not hold in practice. For example, in social networks $\mathrm{t}(G)$ is typically large [26].

In this work we adopt a different measure of sparsity: the *degeneracy* of G, denoted by $d = d(G)$. The degeneracy can be defined as the minimum, over all acyclic orientations of G, of the maximum outdegree of a node; it is a notion strictly stronger than the average

degree (which it bounds from above), but strictly weaker than the maximum degree or the treewidth. Unlike $\Delta(G)$ or $\mathrm{t}(G)$, in social networks d is typically small [16]. Moreover, low-outdegree orientations of G, like the one that defines d, seem to help subgraph counting in practice [35, 22, 30]. Therefore, d seems a good candidate for a parameterization. Yet, no good bounds in terms of d exist, save for specific patterns such as cliques or complete bipartite graphs. This work aims at filling this gap. We develop techniques for counting subgraphs that exploit the low-outdegree orientation of G. This leads to a rich picture, and to faster algorithms to count subgraphs in sparse graphs.

1.1 Results

We present algorithms 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 the degeneracy d of G. Our contributions are of two kinds: bounds and techniques. For simplicity we assume $k = O(1)$ (see Subsection 1.2 for more details).

1.1.1 Bounds

Let $\hom(H,G)$, $\mathrm{sub}(H,G)$, and $\mathrm{ind}(H,G)$ denote respectively the number of homomorphisms, copies, and induced copies of H in G. Our first result is a general-purpose bound, that is, one holding for all H (including disconnected ones):

▶ **Theorem 1.** *For any k-node pattern H one can compute* $\hom(H,G)$*,* $\mathrm{sub}(H,G)$*, and* $\mathrm{ind}(H,G)$ *in time $f(d,k) \cdot \tilde{O}(n^{\lfloor \frac{k}{4} \rfloor + 2})$.*

This bound improves to $0.25k + O(1)$ the exponent of n, which is $0.791k + O(1)$ in the state-of-the-art algorithm of [28]. As an immediate consequence, we have:

▶ **Theorem 2.** *Suppose G has degeneracy $d = O(\mathrm{polylog}(n))$. Then for any k-node pattern H one can compute* $\hom(H,G)$*,* $\mathrm{sub}(H,G)$*, and* $\mathrm{ind}(H,G)$ *in time $f(k) \cdot \tilde{O}(n^{0.25k+2})$.*

Hence, when $d = O(\mathrm{polylog}(n))$, for all sufficiently large k our algorithm beats [28] by a cubic factor. In fact, our algorithm is faster than [28] already for $d < n^{0.721}$ assuming $\omega \approx 2.373$ [25], and in any case for $d < n^{\frac{5}{9}} \approx n^{0.556}$ since $\omega \geq 2$. A second consequence of Theorem 1 comes from the well-known fact that $|E(G)| \geq \binom{d}{2}$. Indeed, this implies:

▶ **Theorem 3.** *Suppose G has average degree $O(\mathrm{polylog}(n))$. Then for any k-node pattern H one can compute* $\hom(H,G)$*,* $\mathrm{sub}(H,G)$*, and* $\mathrm{ind}(H,G)$ *in time $f(k) \cdot \tilde{O}(n^{0.625k+1})$.*

Again, this holds for *all* patterns H, even disconnected ones. To the best of our knowledge, this is the first general-purpose algorithm faster than [28] for graphs with small *average degree* – the weakest possible notion of sparsity.

We give bounds for special classes of patterns, too. First, we consider quasi-cliques, a typical pattern in social graph mining [33, 32, 31]. We prove:

▶ **Theorem 4.** *If H is the clique minus ϵ edges, then one can compute* $\hom(H,G)$*,* $\mathrm{sub}(H,G)$*, and* $\mathrm{ind}(H,G)$ *in time $f(d,k) \cdot \tilde{O}(n^{\lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil})$.*

This generalizes the classic $O(nd^{k-1})$ bound for counting cliques [10], at the price of a polylogarithmic factor. Next, we look at complete quasi-multipartite graphs. We prove:

▶ **Theorem 5.** *If H is a complete multipartite graph, then one can compute* $\hom(H,G)$ *and* $\mathrm{sub}(H,G)$ *in time $f(d,k) \cdot \tilde{O}(n)$. If H is a complete multipartite graph plus ϵ edges, then one can compute* $\mathrm{sub}(H,G)$ *in time $f(d,k) \cdot \tilde{O}(n^{\lfloor \frac{\epsilon}{4} \rfloor + 2})$.*

This generalizes a classic $f(d,k) \cdot O(n)$ bound to count non-induced complete *bi*-partite graphs [13], again at the price of an additional factor.

1.1.2 Techniques

The bounds above are instantiations of a more general result, stated in terms of a novel notion of width, that we call *dag treewidth* $\tau(H)$ of H. Formally, we prove:

▶ **Theorem 6.** *For any k-node pattern H one can compute* $\hom(H,G)$, $\mathrm{sub}(H,G)$, *and* $\mathrm{ind}(H,G)$ *in time* $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$.

This bound, and the width measure $\tau(H)$, arise as follows. As a first step, we orient G acyclically so that it has maximum outdegree d (see below). The problem then becomes counting the copies of all acyclic orientations P of H in G. By inclusion-exclusion arguments, we reduce the problem to counting homomorphisms between a dag P and the dag G. At this point we introduce our technical tool, the *dag tree decomposition* of P. This decomposition allows one to count homomorphisms naturally via dynamic programming, exactly like the standard tree decomposition of a graph; and the running time of the dynamic program is $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$, where the *dag treewidth* $\tau(H)$ is, simplifying a little, the width of the decomposition. The crucial fact is that for $\tau(H)$ we can provide bounds better than just k or $\omega\lfloor\frac{k}{3}\rfloor + 2$ (for example, we prove $\tau(H) \leq \lfloor\frac{k}{4}\rfloor + 2$).

We complement Theorem 6 with a conditional lower bound, showing how $\tau(H)$ characterises the complexity of counting subgraphs in d-degenerate graphs:

▶ **Theorem 7.** *Under the Exponential Time Hypothesis [21], no algorithm can compute* $\mathrm{sub}(H,G)$ *or* $\mathrm{ind}(H,G)$ *in time* $f(d,k) \cdot n^{o(\tau(H)/\ln \tau(H))}$ *for all H.*

▶ Remark 8. Our algorithms work 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, arbitrary graphs. For any subset $V' \subseteq V$ we denote by $G[V']$ the subgraph of G induced by V'; the same notation applies to any graph. 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 write $\phi : H \to G$ to highlight the edges that ϕ preserves. When H and G are oriented, ϕ must preserve the direction of the arcs. If ϕ is injective then we have an injective homomorphism. We denote by $\hom(H,G)$ and $\mathrm{inj}(H,G)$ the number of homomorphisms and injective homomorphisms from H to G. We denote by ψ a map that is not necessarily a homomorphism. The symbol $\simeq$ denotes isomorphism. A *copy* of H in G is a subgraph $F \subseteq G$ such that $F \simeq H$. If moreover $F \simeq G[V_F]$ then F is an induced copy. We denote by $\mathrm{sub}(H,G)$ and $\mathrm{ind}(H,G)$ the number of copies and induced copies of H in G; we may omit G if clear from the context. We denote by P a generic dag obtained by orienting H acyclically. All the notation described above applies to directed graphs in the natural way.

We denote by Δ the maximum degree of G. The *degeneracy* of G is the smallest d such that there is an acyclic orientation of G with maximum outdegree bounded by d. Such an orientation can be found in time $O(|E|)$ by repeatedly removing from G a minimum-degree node [27]. From now on, we assume G has this orientation. We also assume G is encoded

via sorted adjacency lists (every node keeps a sorted list of its out-neighbors). Checking for an arc in G thus takes time $O(\log(d))$, but to lighten the notation we assume it is $O(1)$. We always assume $k = O(1)$. Nonetheless, most of our bounds hold in their current form for $k = O(\ln n)$ or $k = O(\sqrt{\ln n})$. All asymptotic notations hide $\text{poly}(k)$ factors, and the $\tilde{O}(\cdot)$ notation hides $\text{polylog}(ndk)$ factors.

Finally, we recall the definitions of tree decomposition and treewidth of a graph. For any two nodes X, Y in a tree T, we denote by $T(X, Y)$ the unique path between X and Y in T.

▶ **Definition 9** (see [12], Ch. 12.3)**.** *Given a graph $G = (V, E)$, a tree decomposition of G is a tree $D = (V_D, E_D)$ such that each node $X \in V_D$ is a subset $X \subseteq V$, and that*[1]*:*
1. $\cup_{X \in V_D} X = V$
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 $X \in D(X', X'')$ then $X' \cap X'' \subseteq X$*
The width of a tree decomposition T is $\mathrm{t}(T) = \max_{X \in V_T} |X| - 1$. The treewidth $\mathrm{t}(G)$ of a graph G is the minimum of $\mathrm{t}(T)$ over all tree decompositions T of G.

1.3 Related work

The fastest algorithms known for computing $\text{ind}(H, G)$ are based on matrix multiplication and have running time $O(n^{\omega\lfloor \frac{k}{3} \rfloor + (k \bmod 3)})$ [28] or $O(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$ [17], 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 and $\omega = \omega(n, n, n)$. With the current matrix multiplication algorithms, these bounds are essentially $O(n^{0.791k+2})$. These algorithms ignore the sparsity of G, and do not run faster if $d = O(1)$. In contrast, our goal is to reduce the running time when G is sparse. We mention that alternative techniques exist for probabilistic *approximate* counting. Notably, the color coding technique of Alon et al. [2] can be used to sample pattern copies uniformly at random from G in time $O(c^k|G|)$ for some constant $c > 0$ [5, 6, 7].

It is known that several notions of sparsity lead to bounds linear in n. If G has bounded maximum degree $\Delta = O(1)$, then we can compute $\text{ind}(H, G)$ in time $f(k) \cdot O(n)$ via multivariate graph polynomials [29]. If instead G has bounded treewidth $\mathrm{t}(G) = O(1)$, and we are given a tree decomposition of G of width $O(1)$, then by an extension of Courcelle's theorem we can compute $\text{ind}(H, G)$ in time $f(k) \cdot O(n)$ [27]. Similarly, if G is planar, then it can be partitioned into pieces of small treewidth, leading again to an $f(k) \cdot O(n)$ bound [14]. All these conditions are strictly stronger than (and they imply) bounded degeneracy, $d = O(1)$. The techniques used for these bounds are radically different from ours.

For d-degenerate graphs, bounds are known only for special classes of patterns. Chiba and Nishizeki [10] show how to count k-cliques in time $O(d^{k-1}n)$, which can be improved to $O(d^{\omega\lceil (k-1)/3 \rceil}n)$ via matrix multiplication [1]. Eppstein shows how to list all complete bipartite subgraphs in time $O(d^3 2^{2d} n)$ [13] and all maximal cliques in $O(d3^{d/3}n)$ [15]. All these algorithms exploit the degeneracy orientation of G. We exploit such orientation as well, but in a more systematic way and without listing explicitly all occurrences of the pattern; we exploit the structure of H, too, which results in richer bounds.

Concerning the structure of H, $\hom(H, G)$ can be computed in time $f(k) \cdot O(n^{\mathrm{t}(H)+1})$ [18], and $\text{sub}(H, G)$ can be computed in time $f(k) \cdot n^{c(H)+O(1)}$ where $c(H)$ is the vertex-cover number of H [24, 34, 3]. Our bounds are instead parameterized by a novel notion of width, $\tau(H)$, that is within constant factors of the independence number but gives tighter bounds.

[1] Formally, we should define a tree together with a mapping between its nodes and the subsets of V. To lighten the notation, we opt for a less formal definition where the nodes are themselves subsets of V.

Similarly, although several notions of tree decomposition for directed graphs exist [19], our dag tree decomposition is novel and different from all of them.

No lower bound in terms of d and of the structure of H, such as those we give, was known before. The existing lower bounds, based on the Exponential Time Hypothesis (ETH) [21], adopt the parameterizations mentioned above in terms of $\mathrm{t}(H)$ or $c(H)$; see [11] and [8, 9].

Manuscript organisation. Section 2 is a gentle and intuitive introduction to our approach. Section 3 introduces our dag tree decomposition, the dynamic programming for counting homomorphisms, and the corresponding running time bounds. Section 4 bounds the dag treewidth for several classes of patterns, leading to our faster algorithms. Finally, Section 5 proves our lower bounds. All proofs omitted due to space limitations can be found in [4].

2 Exploiting degeneracy orientations

We build the intuition behind our approach, starting from the classic algorithm for counting cliques of [10]. The algorithm begins by orienting G acyclically so that $\max_{v \in G} d_{\mathrm{out}}(v) \leq d$, which requires linear time. With G oriented acyclically, we take each $v \in G$ in turn and enumerate every subset of $(k-1)$ out-neighbors of v. In this way we can explicitly find all k-cliques of G in time $f(k) \cdot O(nd^{k-1}) = f(d,k) \cdot O(n)$. What makes the algorithm tick is the fact that an acyclically oriented clique has exactly one *source*, that is, a node with no incoming arcs. We would like to extend this approach to an arbitrary pattern H. Since every copy of H in G appears with *some* acyclic orientation, we can just take every possible acyclic orientation P of H, count the copies of P in G, and sum all the counts. Thus, we can reduce the problem to the following one: given a k-node dag P, and an n-node dag G with maximum outdegree d, count the copies of P in G.

Let us try a first approach. If P has $s = s(P)$ sources, we enumerate all the $\binom{n}{s} = O(n^s)$ ordered s-uples of V to which those sources can be mapped. For each such s-uple, we list the possible mappings of the remaining $k-s$ nodes, which can be done in time $O(d^{k-s})$ by listing the mappings of a fixed spanning forest of P. Finally, we check if the k nodes induce P in G. The total running time is $f(k) \cdot O(n^s d^{k-s})$. Unfortunately, if P is an independent set then $s = k$ and the running time is $O(n^k)$. The situation does not improve even if P is connected, as we can have $s = k-1$ (for the inward-oriented star).

Here our approach comes into play. We use the pattern P in Figure 1 as a toy example. Instead of listing all occurrences of P in G, we decompose P into two *pieces*, $P(1)$ and $P(2,3)$, where $P(u)$ denotes the subgraph of P reachable from u (that is, the transitive closure of u in P), and $P(u_1, \ldots, u_r) = \cup_{i=1}^{r} P(u_i)$. The idea is to compute the count of P by combining the counts of the two pieces, $P(1)$ and $P(2,3)$.

To simplify the task, we focus on counting the *homomorphisms* between P and G; we can then recover the number of induced copies by inclusion-exclusion arguments. In fact, we solve a slightly more complex problem: for each pair of nodes $x, y \in G$, we count the homomorphisms from P to G that map nodes 2 and 4 (see Figure 1) to x and y respectively. To recover $\hom(P,G)$ we then just sum over all pairs x, y. Formally, for a given pair x, y let $\phi : \{2,4\} \mapsto V$ be the map given by $\phi(2) = x$ and $\phi(4) = y$, and let $\hom(P,G,\phi)$ be the number of homomorphisms from P to G whose restriction to $\{2,4\}$ is ϕ. In the same way define $\hom(P(1),G,\phi)$ and $\hom(P(2,3),G,\phi)$. It is easy to see that:

$$\hom(P,G,\phi) = \hom(P(1),G,\phi) \cdot \hom(P(2,3),G,\phi) \tag{1}$$

To compute $\hom(P,G,\phi)$ we then just need $\hom(P(1),G,\phi)$ and $\hom(P(2,3),G,\phi)$. But we can compute $\hom(P(1),G,\phi)$ simultaneously for all ϕ in time $f(d,k) \cdot \tilde{O}(n)$, using our

listing technique to build a dictionary mapping each ϕ to its count. (The $\tilde{O}(\cdot)$ factor comes from the cost of accessing the dictionary, which has size $\text{poly}(n)$). Similarly, we can compute $\hom(P(2,3), G, \phi)$ in time $f(d,k) \cdot \tilde{O}(n^2)$. The total running time is $f(d,k) \cdot \tilde{O}(n^2)$, whereas our first approach would take time $f(d,k) \cdot O(n^3)$.

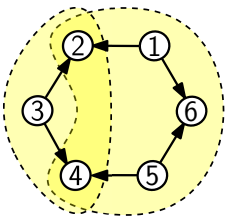

Figure 1 Toy example: an acyclic orientation P of $H = C_6$, decomposed into two pieces.

Abstracting from our toy example, we want to decompose P into a set of pieces $P_1, \ldots, P_\kappa$ with the following properties: (i) each piece P_i has a small number of sources $s(P_i)$, and (ii) we can obtain $\hom(P, G, \phi)$ by combining the homomorphism counts of the P_i. This is achieved precisely by the *dag tree decomposition*, which we introduce in Section 3. Like the tree decomposition of an undirected graph, the dag tree decomposition leads to a dynamic program to compute $\hom(P, G)$. The running time is $\tilde{O}(n^{\max_i s(P_i)})$, hence to make the algorithm useful we must show that a decomposition with "small" $\max_i s(P_i)$ always exists, which we do in Section 4.

3 DAG tree decompositions

Let $P = (V_P, A_P)$ be a directed acyclic graph. We denote by S_P, or simply S, the set of nodes of P having no incoming arc. These are the *sources* of P. We denote by $V_P(u)$ the transitive closure of u in P, i.e. the set of nodes of P reachable from u, and we let $P(u) = P[V_P(u)]$ be the corresponding subgraph of P. More generally, for a subset of sources $B \subseteq S$ we let $V_P(B) = \cup_{u \in B} V_P(u)$ and $P(B) = P[V_P(B)]$. Thus, $P(B)$ is the subgraph of P induced by all nodes reachable from any source in B. We call B a *bag* of sources. We can now formally introduce our decomposition.

▶ **Definition 10** (Dag tree decomposition). *Let $P = (V_P, A_P)$ be a dag. A dag tree decomposition (d.t.d.) of P is a rooted tree $T = (\mathcal{B}, \mathcal{E})$ with the following properties:*

1. *each node $B \in \mathcal{B}$ is a bag of sources $B \subseteq S_P$*
2. $\bigcup_{B \in \mathcal{B}} B = S_P$
3. *for all $B, B_1, B_2 \in T$, if $B \in T(B_1, B_2)$ then $V_P(B_1) \cap V_P(B_2) \subseteq V_P(B)$*

One can see immediately the resemblance to the standard tree decomposition of a graph (Definition 9). However, our dag tree decomposition differs crucially in two aspects. First, the bags of the tree are subsets of S rather than subsets of V_P. This is because the time needed to list the homomorphisms between $P(B_i)$ and G is driven by $|B_i|$, which is the number of sources in $P(B_i)$. Second, the path-intersection property concerns not the bags themselves, but the pieces reachable from the bags themselves. The reason is that, to combine the counts of two pieces together, their intersection must form a separator in G (similarly to what happens with the standard tree decomposition).

The dag tree decomposition induces immediately the following notions of *width*, that we use throughout the rest of the article.

▶ **Definition 11.** *The* width *of* T *is* $\tau(T) = \max_{B \in \mathcal{B}} |B|$. *The* dag treewidth $\tau(P)$ *of* P *is the minimum of* $\tau(T)$ *over all dag tree decompositions* T *of* P.

Figure 2 shows a pattern P, together with a d.t.d. T of width 1. Note that $1 \leq \tau(P) \leq k$ for any P. Note also that $\tau(P)$ has no relationship with the treewidth $\mathrm{t}(H)$ of H: they can both be $\Theta(k)$, or we can have $\tau(P) = 1$ but $\mathrm{t}(H) = k$ (when H is a clique). In fact, $\tau(P)$ is within constant factors of the independence number $\alpha(H)$ of H (see Section 4.3), and thus decreases as H becomes denser. The intuition is that adding arcs increases the number of nodes reachable from the sources, hence we need smaller bags to cover all of P. When H is a clique, P is reachable from just one source and $\tau(P) = 1$.

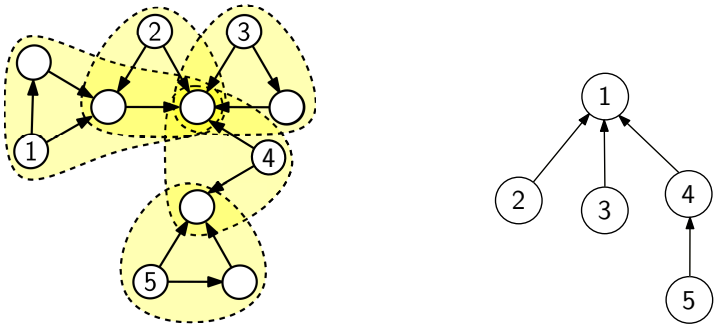

Figure 2 A dag P with one possible decomposition into five pieces (left), and one possible dag tree decomposition T for P (right). Since $\tau(T) = 1$, we can compute $\hom(P, G)$ in time $f(d, k) \cdot O(n)$.

3.1 Counting homomorphisms via dag tree decompositions

For any $B \in \mathcal{B}$ let $T(B)$ be the subtree of T rooted at B. We let $\Gamma[B]$ be the down-closure of B in T, that is, the union of all bags in $T(B)$. Consider $P(\Gamma[B])$, the subgraph of P induced by the nodes reachable from any $u \in \Gamma[B]$ (note the difference with $P(B)$, which contains only the nodes reachable from any $u \in B$). We compute $\hom(P(\Gamma[B]), G)$ in a bottom-up fashion over all B, starting with the leaves of T and moving towards the root. This is, in essence, the dynamic program given by the standard tree decomposition (see [18])[2].

As anticipated, we actually compute a refined count: $\hom(P(\Gamma[B]), \phi)$, the number of homomorphisms that extend a fixed mapping ϕ. Formally:

▶ **Definition 12.** *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* ϕ_1 *and* ϕ_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* $\hom(P_1, G, \phi_2)$ *or simply* $\hom(P_1, \phi_2)$ *the number of homomorphisms from* P_1 *to* G *that respect* ϕ_2.

We can now present our main algorithmic result.

▶ **Lemma 13.** *Let* P *be any* k*-node dag, and* $T = (\mathcal{B}, \mathcal{E})$ *be a d.t.d. for* P. *Fix any* $B \in \mathcal{B}$ *as the root of* T. *There exists a dynamic programming algorithm HomCount*(P, T, B) *that computes* $\hom(P(\Gamma[B]), \phi_B)$ *for all* $\phi_B : P(B) \to G$ *in time* $f(d, k) \cdot \tilde{O}(n^{\tau(T)})$.

[2] The correctness of our dynamic program does not follow automatically from the dynamic program over standard tree decompositions. A proof of correctness is given in the full version of the manuscript.

Note that $\hom(P,G)$ is simply the sum of all the counts $\hom(P(\Gamma[B]),\phi_B)$ returned by the algorithm. Therefore, we can compute $\hom(P,G)$ in time $f(d,k)\cdot\tilde{O}(n^{\tau(T)})$. Now, a d.t.d. that minimises $\tau(T)$ can obviously be found in time $f(k)=O(f(d,k))$. Therefore:

▶ **Theorem 14.** *We can compute* $\hom(P,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau(P)})$.

Equipped with Theorem 14, we can turn to the original problem of counting the copies of H.

3.2 Inclusion-exclusion arguments

We turn to computing $\hom(H,G)$, $\mathrm{sub}(H,G)$ and $\mathrm{ind}(H,G)$. We do so via standard inclusion-exclusion arguments, using our algorithm for computing $\hom(P,G)$ as a primitive. To this end, we shall define appropriate notions of width for undirected pattern graphs. 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/θ be the pattern obtained from H by identifying equivalent nodes according to θ 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 15.** *The* dag treewidth *of* H *is* $\tau(H)=\tau_3(H)$*, where:*

$$\tau_1(H)=\max\{\tau(P):P\in\Sigma\} \tag{2}$$
$$\tau_2(H)=\max\{\tau_1(H/\theta):\theta\in\Theta\} \tag{3}$$
$$\tau_3(H)=\max\{\tau_2(H'):H'\in D(H)\} \tag{4}$$

We can then state:

▶ **Theorem 16.** *One can compute:*
- $\hom(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau_1(H)})$,
- $\mathrm{sub}(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau_2(H)})$,
- $\mathrm{ind}(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau(H)})$.

The algorithmic part of our work is complete. We shall now focus on bounding $\tau_1(H)$, $\tau_2(H)$, and $\tau(H)$, so to instantiate Theorem 16 and prove the upper bounds of Section 1.1.

4 Bounds on the dag treewidth

In this section we develop upper bounds on $\tau_1(H),\tau_2(H),\tau(H)$ as a function of H. First, we bound $\tau(H)$ for cliques minus ϵ edges, obtaining a generalization of the classic clique counting bound of [10]. Then, we bound $\tau_2(H)$ for complete multipartite graphs plus ϵ edges, obtaining a generalization of a result by Eppstein [13]. Next, we show that $\Omega(\alpha(H))\leq\tau(H)\leq\alpha(H)$. Finally, we show that $\tau(H)\leq\lfloor\frac{k}{4}\rfloor+2$ for *every* pattern H, including disconnected ones. This requires a nontrivial proof.

Before proceeding, we need some definitions. We say a node $v\in P$ is a *joint* if it is reachable from two or more 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. We write $J(u)$ for the set of joints reachable from u, and for any $X\subseteq V_P$ we let $J(X)=\cup_{u\in X}J(u)$.

4.1 Quasi-cliques

▶ **Lemma 17.** *If* H *has* $\binom{k}{2}-\epsilon$ *edges then* $\tau(H)\leq\lceil\frac{1}{2}+\sqrt{\frac{\epsilon}{2}}\rceil$.

Proof. The source set $|S|$ of P is an independent set, hence $|E_H| \leq \binom{k}{2} - \binom{|S|}{2}$. Therefore $\epsilon \geq \binom{|S|}{2}$, which implies $|S| \leq 1 + \sqrt{2\epsilon}$. A d.t.d. for P is the tree on two bags B_1, B_2 that satisfy $B_1 \cup B_2 = S$, $|B_1| = \lfloor |S|/2 \rfloor$, and $|B_2| = \lceil |S|/2 \rceil$. Hence $\tau(P) \leq \lceil |S|/2 \rceil \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$. Now consider any H' obtained from H by adding edges or identifying nodes. Obviously $|E_{H'}| \geq \binom{k'}{2} - \epsilon$ where $k' = |V_{H'}|$, and the argument above implies $\tau(P') \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$ for any orientation P' of H'. By Definition 15, then, $\tau(H) \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$. ◀

By Theorem 16 and since $\tau_1(H) \leq \tau_2(H) \leq \tau(H)$ it follows that we can compute $\hom(H, G)$, $\mathrm{sub}(H, G)$ and $\mathrm{ind}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{\lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil})$, proving Theorem 4.

4.2 Quasi-multipartite graphs (non-induced)

▶ **Lemma 18.** *If H is a complete multipartite graph, then $\tau_2(H) = 1$. If H is a complete multipartite graph plus ϵ edges, then $\tau_2(H) \leq \lfloor \frac{\epsilon}{4} \rfloor + 2$.*

Proof. Suppose $H = (V_H, E_H)$ is a complete multipartite graph. Let $V_H = V_H^1 \cup \ldots \cup V_H^\kappa$ 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, $V_P(u) = V_P(u')$ for any two sources u, u'. A d.t.d. T of width $\tau(T) = 1$ is the trivial one with one source per bag.

Suppose then we add ϵ edges to H. Again, in any orientation P of H, all sources are contained in exactly one V_H^j, but we might have $V_P(u) \neq V_P(u')$ for two distinct sources u, u'. Note however that all nodes in $V_H \setminus V_H^j$ are reachable from all sources and are thus irrelevant to a d.t.d.. More precisely, any d.t.d. for $P[V_H^j]$ is a d.t.d. for P. But $P[V_H^j]$ has at most ϵ edges and by Theorem 20 (see below) it has a d.t.d. of width at most $\lfloor \frac{\epsilon}{4} \rfloor + 2$.

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^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 $\lfloor \frac{\epsilon}{4} \rfloor + 2$ as above. ◀

By Theorem 16, if H is complete multipartite then we can compute $\hom(H, G)$ and $\mathrm{sub}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n)$. If instead H is complete multipartite plus ϵ edges, then we can compute $\hom(H, G)$ and $\mathrm{sub}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{\lfloor \frac{\epsilon}{4} \rfloor + 2})$. This proves Theorem 5.

4.3 Independence number and dag treewidth

▶ **Lemma 19.** *Any k-node graph H satisfies $\Omega(\alpha(H)) \leq \tau(H) \leq \alpha(H)$.*

Proof. Let H be any pattern graph on k nodes. For the upper bound, note that in any acyclic orientation P of H the sources form an independent set, and that $\alpha(H') \leq \alpha(H)$ for any H' obtained by adding edges or identifying nodes of H.

For the lower bound, we exhibit a pattern H' obtained by adding edges to H such that $\tau(P) = \Omega(\alpha(H))$ for all its acyclic orientations P. Let $I \subseteq V_H$ be an independent set of H with $|I| = \Omega(\alpha(H))$ and $|I| \bmod 5 \equiv 0$. Partition I in I_1, I_2 where $|I_1| = \frac{2}{5}|I|$ and $|I_2| = \frac{3}{5}|I|$. On top of I_1 we virtually build a 3-regular expander $\mathcal{E} = (I_1, E_\mathcal{E})$ of linear treewidth $t(\mathcal{E}) = \Omega(|I_1|)$. It is well known that such expanders exist (see e.g. Proposition 1 and Theorem 5 of [20]). For each edge $uv \in E_\mathcal{E}$ 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 words, $H[I]$ is the 1-subdivision of $\mathcal{E}$. Let H' be the resulting pattern. Note that $t(\mathcal{E}) = \Omega(|I_1|) = \Omega(|I|) = \Omega(\alpha(H))$.

Let now $P = (V_P, A_P)$ be any acyclic orientation of H' having I_2 as source set, and let T be any d.t.d. of P. We show that $\tau(T) \geq \frac{1}{2}(\mathrm{t}(\mathcal{E}) + 1)$. To this end, we build a tree D by replacing each bag $B \in T$ with the bag $J(B)$. We can show that D is a tree decomposition of $\mathcal{E}$ (see Definition 9). First, by point (2) of Definition 10 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 $e \in E_{\mathcal{E}}$ we have a node $u = u_e \in V_P$. Then, again by point (2) of Definition 10, there is $B \in T$ such that $u \in B$; and by construction of D it holds $e = \{v, w\} \subseteq J(B)$. This proves property (2). Third, fix any $J(B_1), J(B_2), J(B_3) \in D$ such that $J(B_1)$ is on the path from $J(B_2)$ to $J(B_3)$ in D, and consider any $v \in J(B_2) \cap J(B_3)$. There is $u \in B_2$ 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 path from B_2 to B_3 in T, point (3) of Definition 9 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 Definition 9 and Definition 11 we have $\mathrm{t}(\mathcal{E}) \leq 2\tau(P) - 1$, that is, $\tau(P) \geq \frac{1}{2}(\mathrm{t}(\mathcal{E}) + 1)$. But $t(\mathcal{E}) = \Omega(\alpha(H))$, thus $\tau(P) = \Omega(\alpha(H))$. ◀

4.4 All patterns

This subsection is devoted entirely to prove:

▶ **Theorem 20.** *For any dag $P = (V_P, A_P)$, in time $O(1.7549^k)$ we can compute a dag tree decomposition $T = (\mathcal{B}, \mathcal{E})$ with $\tau(T) \leq \min(\lfloor \frac{e}{4} \rfloor, \lfloor \frac{k}{4} \rfloor) + 2$, where $k = |V_P|$ and $e = |A_P|$.*

Combined with Definition 15, this gives:

▶ **Corollary 21.** *Any k-node graph H satisfies $\tau(H) \leq \lfloor \frac{k}{4} \rfloor + 2$.*

The proof of Theorem 20 proceeds as follows. First, we greedily find a subset $B^* \subseteq S$ such that $|V_P(B^*)| \geq 4|B^*|$. If this subset coincides with S, we are done. Otherwise we delete B^* and $V_P(B^*)$ from P, partitioning P itself in $\ell \geq 1$ connected components. We can easily show that the d.t.d.'s of the individual components can be combined into a d.t.d. for P, if we add B^* to every bag. The crux, then, is bounding the dag treewidth of the individual components. We show that, if the i-th component has k_i nodes, then it admits a d.t.d. of width $\frac{k_i}{4} + 2$. This requires to first "peel" the component, getting rid of the tree-like parts, and then decomposing its core using tree decompositions.

The proof makes heavy use of the *skeleton* of P, defined as follows.

▶ **Definition 22.** *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 $v \in J(u)$.*

Figure 3 gives an example. Note that $\Lambda(P)$ does not contain nodes that are neither sources nor joints; the reason is that those nodes are irrelevant to the construction of a d.t.d.. Note also that building $\Lambda(P)$ takes time $O(\mathrm{poly}(k))$.

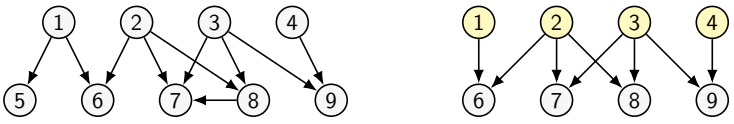

Figure 3 Left: a dag P. Right: its skeleton $\Lambda(P)$ (sources S above, joints J below).

Let us now delve into the proof. For any node x, we denote by d_x the current degree of x in the skeleton.

1. Shattering the skeleton. Set $B^{(0)} = \emptyset$ and let $\Lambda^{(0)} = (S^{(0)} \cup J^{(0)}, E^{(0)}_\Lambda)$ be a copy of Λ. Set $j = 0$ and proceed iteratively as follows. If there is a source $u \in S^{(j)}$ with $d_u \geq 3$, let $B^{(j+1)} = B^{(j)} \cup \{u\}$, and let $\Lambda^{(j+1)} = (S^{(j+1)} \cup J^{(j+1)}, E^{(j+1)}_\Lambda)$ be obtained from $\Lambda^{(j)}$ by removing u and $J^{(j)}(u)$; otherwise stop. Suppose the procedure stops at $j = j^*$, producing the subset $B^* = B^{(j^*)}$ and the sub-skeleton $\Lambda^* = \Lambda^{(j^*)} = (S^* \cup J^*, E^*_\Lambda)$. We prove:

▶ **Lemma 23.** *$|B^*| \leq \min\left(\frac{k-|\Lambda^*|}{4}, \frac{e-|E^*_\Lambda|}{4}\right)$, where $k = |V_P|$ and $e = |A_P|$.*

Proof. Since each step removes at least 4 nodes from $\Lambda^{(j)}$, then $4|B^*| \leq |\Lambda \setminus \Lambda^*| \leq (k - |\Lambda^*|)$, and $|B^*| \leq \frac{k-|\Lambda^*|}{4}$. Now consider the nodes $\{u\} \cup J^{(j)}(u)$ removed at step j. Note that $\Lambda^{(j)}$ is just the skeleton of $P^{(j)} = P \setminus P(B^{(j)})$, and that $J^{(j)}(u) \subseteq P^{(j)}(u)$. This implies $P^{(j)}(u)$ contains at least 3 arcs. Moreover, there must be at least one arc from $P^{(j)} \setminus P^{(j)}(u)$ to $P^{(j)}(u)$, otherwise $P^{(j)}(u)$ would not contain joints of $P^{(j)}$. We have therefore at least 4 arcs pointing to nodes of $P^{(j)}(u)$. These arcs are counted only once, since $P^{(j)}(u)$ is then removed from $P^{(j)}$. Hence $e \geq 4|B^*| + |E^*_\Lambda|$, and $|B^*| \leq \frac{e-|E^*_\Lambda|}{4}$. ◀

Now, if $B^* = S$, then obviously $T = (\{B^*\}, \emptyset)$ is a d.t.d. of P. Moreover in this case Λ^* is empty, so Lemma 23 gives $\tau(T) = |B^*| \leq \min(\frac{k}{4}, \frac{e}{4})$, proving Theorem 20.

Suppose instead $B^* \subset S$. Let $P^* = P \setminus P(B^*)$, and let $\Lambda_i = (S_i \cup J_i, E_i) : i = 1, \ldots, \ell$ be the connected components of Λ^*. One can check that Λ_i is precisely the skeleton of $P^*(S_i)$, the i-th connected component of P^*. As the next lemma says, we can obtain a d.t.d. for P by simply combining the d.t.d.'s of the $P^*(S_i)$ in a tree and adding B^* to all the bags. For simplicity, we say "a d.t.d of Λ_i" for "a d.t.d. of $P^*(S_i)$".

▶ **Lemma 24.** *For $i = 1, \ldots, \ell$ let $T_i = (\mathcal{B}_i, \mathcal{E}_i)$ be a d.t.d. of Λ_i. Consider the tree T obtained as follows. The root of T is the bag B^*, and the subtrees below B^* are $T_1, \ldots, T_\ell$, where each bag $B \in T_i$ has been replaced by $B \cup B^*$. Then $T = (\mathcal{B}, \mathcal{E})$ is a d.t.d. of P with $\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \tau(T_i)$ and $|\mathcal{B}| = 1 + \sum_{i=1}^{\ell} |\mathcal{B}_i|$.*

Proof. The claims on $\tau(T)$ and $|\mathcal{B}|$ are trivial. Let us check via Definition 10 that T is a d.t.d. of P. Point (1) is immediate. For point (2), note that $\cup_{B \in \mathcal{B}_i} = S_i$ because T_i is by hypothesis a d.t.d. of Λ_i; by construction, then, $\cup_{B \in \mathcal{B}} = B^* \cup_{i=1}^{\ell} S_i = S_P$. Now point (3). Choose any two bags $B' \cup B^*$ and $B'' \cup B^*$ of T, where $B' \in T_i$ and $B'' \in T_j$ for some $i, j \in \{1, \ldots, \ell\}$, and any bag $B \cup B^* \in T(B' \cup B^*, B'' \cup B^*)$. 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 this implies $V_P(B' \cup B^*) \cap V_P(B'' \cup B^*) \subseteq V_P(B \cup B^*)$. Suppose instead $i \neq j$. Thus $J_i(S_i) \cap J_j(S_j) = \emptyset$ and this means that $J(S_i) \cap J(S_j) \subseteq J(B^*)$. But $V_P(B_i) \cap V_P(B_j) \subseteq J(S_i) \cap J(S_j)$ and $J(B^*) \subseteq V_P(B^*)$, thus $V_P(B_i) \cap V_P(B_j) \subseteq V_P(B^*)$. It follows that for every bag $B \cup B^*$ of T we have $V_P(B_i \cup B^*) \cap V_P(B_j \cup B^*) \subseteq V_P(B \cup B^*)$. ◀

2. Peeling Λ_i. We now remove the tree-like parts of $\Lambda_i = (S_i \cup J_i, E_i)$; for instance, sources that point to only a single joint. The intuition is that those parts do not increase the dag treewidth. As a base case, if $|S_i| = 1$ then $T_i = (\{S_i\}, \emptyset)$ is a d.t.d. for Λ_i of width 1. Suppose instead $|S_i| > 1$. Note that every $u \in S_i$ satisfies $d_u = |J_i(u)| \leq 2$, for otherwise u would have been removed in the previous phase. Consider the following conditions:

1. $\exists u \in S_i : d_u = 1$. Then fix any such u, and fix any $u' \in S_i \setminus \{u\}$ with $J_i(u) \cap J_i(u') \neq \emptyset$.
2. $J_i(u) = J_i(u')$ for some $u, u' \in S_i$ with $u \neq u'$. Then, fix u and u' as above.
3. $\exists v \in J_i : d_v = 1$ (this is initially false). Then fix any such v, let u be the unique source such that $v \in J_i(u)$, and let $u' \neq u$ 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 Λ_i is always connected. We then "peel" Λ_i by defining T_i recursively, as follows. Let $\Lambda'_i = \Lambda_i \setminus \{u\}$, and assume we have a d.t.d. T'_i of Λ'_i. Since $u' \neq u$ then $u' \in S_i \setminus \{u\}$, and thus for some $B' \in T'_i$ we have $u' \in B'$. Create the bag $B_u = \{u\}$ and set it as a child of B'. We obtain a tree T_i where B_u is a leaf; and note that, by construction, for any $u'' \in S_i \setminus \{u, u'\}$ we have $J_i(u) \cap J_i(u'') \subseteq J_i(u')$. This implies that T_i is a d.t.d. for Λ_i. Then remove u from Λ_i, as well as any $v : d_v = 0$.

We repeat this peeling process until we meet the base case, or until $|S_i| > 1$ and all three conditions above fail. In the latter case, we move to the next phase.

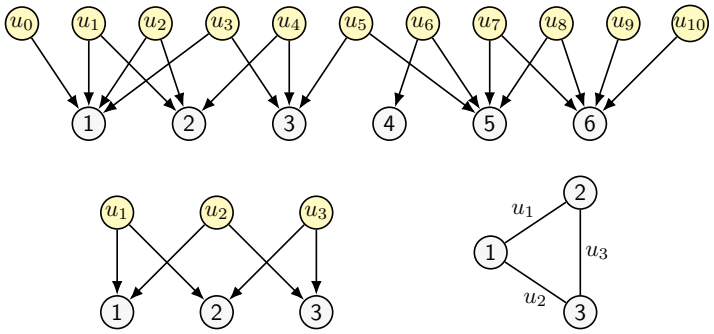

Figure 4 Above: example of a skeleton component Λ_i. Below: the core $\Lambda_i^\bullet$ obtained from Λ_i after peeling (left), and its encoding as C_i (right).

3. Decomposing the core. We denote by $\Lambda_i^\bullet = (S_i^\bullet \cup J_i^\bullet, E_i^\bullet)$ the subgraph of Λ_i left after the peeling. We say $\Lambda_i^\bullet$ is the *core* of Λ_i; intuitively, it is the part determining the dag treewidth of Λ_i. Now, since $\Lambda_i^\bullet$ violates all three conditions of the peeling step, certainly $d_u = 2$ for every source u and $d_v \geq 2$ for every joint v. This means that the joints and sources of $\Lambda_i^\bullet$ can be represented as nodes and edges of a simple graph. Formally, we encode $\Lambda_i^\bullet$ as $C_i = (V_{C_i}, E_{C_i})$ where $V_{C_i} = J_i^\bullet$ and $E_{C_i} = \{e_u : u \in S_i^\bullet\}$, as Figure 4 shows.

Using C_i, we can find a good bound on $\tau(\Lambda_i^\bullet)$ via tree decompositions. The key fact is that any tree decomposition for C_i of width t can be turned in time $poly(k)$ into a d.t.d. for $\Lambda_i^\bullet$ of width $t+1$ (intuitively, the tree decomposition covers the edges of C_i, which are the sources of $\Lambda_i^\bullet$). By a bound of [23], C_i admits a tree decomposition of width at most $\frac{|E_{C_i}|}{5} + 2$, and this can be computed in time $O(1.7549^k)$ [18]. In the end, this yields:

▶ **Lemma 25.** *Let $k_i = |S_i^\bullet \cup J_i^\bullet|$. In time $O(1.7549^{k_i})$ we can compute a d.t.d. $T_i^\bullet = (\mathcal{B}_i^\bullet, \mathcal{E}_i^\bullet)$ of $\Lambda_i^\bullet$ such that $\tau(T_i^\bullet) \leq \lfloor \frac{k_i}{4} \rfloor + 2$.*

With Lemma 25 we have essentially finished. It remains to wrap all our bounds together.

4. Assembling the tree. Let T_i be the d.t.d. for Λ_i, as returned by the recursive peeling followed by the core decomposition. Note that $\tau(T_i) \leq \tau(T_i^\bullet)$, since the peeling phase only add bags of width 1. Therefore, by Lemma 25, $\tau(T_i) \leq \lfloor \frac{k_i}{4} \rfloor + 2$ where $k_i = |S_i^\bullet \cup J_i^\bullet|$.

Let now $T = (\mathcal{B}, \mathcal{E})$ be the d.t.d. for P obtained by composing $T_1, \ldots, T_\ell$ (Lemma 24). By Lemma 24 itself, $\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \tau(T_i)$, thus:

$$\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \lfloor \frac{k_i}{4} \rfloor + 2 \tag{5}$$

Now, from Lemma 23 we know that $P(B^*)$ has at least $4|B^*|$ nodes and $4|B^*|$ arcs. Similarly, since each Λ_i has at least k_i nodes and k_i arcs, then $P \setminus P(B^*)$ has at least $\sum_{i=1,\ldots,\ell} k_i$

nodes and $\sum_{i=1,\dots,\ell} k_i$ arcs. By a simple summation, then, we have $\tau(T) \leq \lfloor \frac{k}{4} \rfloor + 2$ and $\tau(T) \leq \lfloor \frac{e}{4} \rfloor + 2$, hence $\tau(T) \leq \min(\lfloor \frac{k}{4} \rfloor, \lfloor \frac{e}{4} \rfloor) + 2$. Finally, by Lemma 25 the time to build T_i is $O(1.7549^{k_i})$, since the peeling phase clearly takes time $\text{poly}(k_i)$. The total time to build T is therefore $O(1.7549^k)$, which concludes the proof of Theorem 20.

5 Lower bounds

We prove the lower bound of Theorem 7. Note that, since $\tau(H) = \Theta(\alpha(H))$ by Lemma 19, the bound still holds if one replaces $s(H)$ by $\alpha(H)$ in the statement.

▶ **Theorem 26.** *For any function $a : [k] \to [1, k]$ 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 $f(d, k) \cdot n^{o(a(k)/\ln a(k))}$ for all $H \in \mathcal{H}$ where d is the degeneracy of G, then ETH fails.*

Proof. We reduce counting cycles in an arbitrary graph to counting a gadget pattern on k nodes and dag treewidth $O(\tau(k))$, where $\tau(k) = a(k)$, in a d-degenerate graph.

The gadget is the following. Consider a simple cycle on $k_0 \geq 3$ nodes. Choose an integer $d = d(k) \geq 2$ with $d(k) \in \Omega(\frac{k}{\tau(k)})$. 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 $\tau(H) \leq k_0$. This implies $\tau(H) = O(\tau(k))$ since $k_0 = \frac{k}{d} \in O(\tau(k))$. 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_σ 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 $\tau(H_\sigma) \leq |S(H_\sigma)|$, we have $\tau(H_\sigma) \leq k_0$. Note any H'_σ obtained from H_σ by adding edges or identifying nodes has at most k_0 roots, too. Hence $\tau(H) \leq k_0$.

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(n 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 $f(d, k) \cdot n^{o(\tau(H)/\ln \tau(H))}$. Since $\tau(H) \leq k_0$, $k = f(d, k_0)$, $n = O(dn_0^2)$, 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 $f(k_0) \cdot n^{o(k_0/\ln k_0)}$. The proof is completed by invoking:

▶ **Theorem 27** ([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.* ◀

6 Conclusions

We have shown how one can exploit the sparsity of a graph to count subgraphs faster than with state-of-the-art algorithms. Our main technical ingredient, the dag tree decomposition, not only yields better algorithms, but sheds light on the algorithmic role of degeneracy in subgraph counting, too. It would be interesting to know if our decomposition can be applied to problems other than subgraph counting.

An obvious line of future research is to tighten the bounds. For all patterns, one could improve the upper bound by reducing the exponent by constant or logarithmic factors; larger improvements seem unlikely, due to the lower bounds. For special classes of patterns, instead,

the situation is different: our lower bounds hold for *some* infinite family of patterns, not for *any* infinite family. This leaves open the question of finding special classes of patterns that can be counted even faster, or of tightening the lower bounds. In the second case, the dag treewidth would completely characterise the complexity of subgraph counting when parameterized by the degeneracy of the host graph.

References

1 N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–249, July 2008. doi:10.1093/bioinformatics/btn163.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting Thin Subgraphs via Packings Faster Than Meet-in-the-middle Time. In *Proc. of ACM-SIAM SODA*, pages 594–603, 2014. doi:10.1137/1.9781611973402.45.

4 Marco Bressan. Faster subgraph counting in sparse graphs. *CoRR*, abs/1805.02089, 2018. arXiv:1805.02089.

5 Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Counting Graphlets: space vs. time. In *Proc. of ACM WSDM*, pages 557–566, 2017. doi:10.1145/3018661.3018732.

6 Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif Counting Beyond Five Nodes. *ACM Transactions on Knowledge Discovery from Data*, 20(2):1–25, April 2018. doi:10.1145/3186586.

7 Marco Bressan, Stefano Leucci, and Alessandro Panconesi. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proc. of the VLDB Endowment*, 12(11):1651–1663, July 2019. doi:10.14778/3342263.3342640.

8 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. doi:10.1016/j.ic.2005.05.001.

9 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. doi:10.1016/j.jcss.2006.04.007.

10 Norishige Chiba and Takao Nishizeki. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

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. doi:10.1109/FOCS.2014.22.

12 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

13 David Eppstein. Arboricity and Bipartite Subgraph Listing Algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994. doi:10.1016/0020-0190(94)90121-X.

14 David Eppstein. Subgraph Isomorphism in Planar Graphs and Related Problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27, 1999. doi:10.7155/jgaa.00014.

15 David Eppstein, Maarten Löffler, and Darren Strash. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *Algorithms and Computation*, pages 403–414. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-17517-6_36.

16 David Eppstein and Darren Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs. In *Experimental Algorithms*, pages 364–375. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-20662-7_31.

17 Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and Counting Small Pattern Graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015. doi:10.1137/140978211.

18 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 1 edition, 2010. doi:10.1007/978-3-642-16533-7.

19 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *Journal of Combinatorial Theory, Series B*, 116:250–286, January 2016. doi:10.1016/j.jctb.2015.09.001.

20 Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory, Series B*, 99(1):218–228, 2009. doi:10.1016/j.jctb.2008.06.004.

21 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Proc. of IEEE FOCS*, pages 653–662, 1998. doi:10.1109/sfcs.1998.743516.

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, 2017. doi:10.1145/3038912.3052636.

23 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Algorithms Based on the Treewidth of Sparse Graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 385–396. Springer Berlin Heidelberg, 2005. doi:10.1007/11604686_34.

24 Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and Detecting Small Subgraphs via Equations. *SIAM Journal on Discrete Mathematics*, 27(2):892–909, January 2013. doi:10.1137/110859798.

25 François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proc. of ISSAC*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

26 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *Proc. of ICDT*, pages 12:1–12:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.12.

27 J. Nešetřil and P.O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-27875-4.

28 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

29 Viresh Patel and Guus Regts. Computing the Number of Induced Copies of a Fixed Graph in a Bounded Degree Graph. *Algorithmica*, 81(5):1844–1858, September 2018. doi:10.1007/s00453-018-0511-9.

30 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *Proc. of WWW*, pages 1431–1440, 2017. doi:10.1145/3038912.3052597.

31 Ahmet Erdem Sariyüce and Ali Pinar. Peeling Bipartite Networks for Dense Subgraph Discovery. In *Proc. of ACM WSDM*, pages 504–512, 2018. doi:10.1145/3159652.3159678.

32 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. doi:10.1145/3057742.

33 Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable Motif-aware Graph Clustering. In *Proc. of WWW*, pages 1451–1460, 2017. doi:10.1145/3038912.3052653.

34 Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.

35 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local Higher-Order Graph Clustering. In *Proc. of ACM KDD*, pages 555–564, 2017. doi:10.1145/3097983.3098069.