Deterministically Counting k-Paths and Trees Parameterized by Treewidth in Single-Exponential Time

Jonne Visser
Utrecht University, The Netherlands

Hans L. Bodlaender
Utrecht University, The Netherlands

Abstract

In this paper, we give new and faster deterministic algorithms to count the number of k-paths and trees in host graphs of bounded treewidth. Our algorithms use time that is single-exponential in the treewidth, and employ the determinant method from [4]. Modifications of the algorithms count in single-exponential time the number of k-paths between specified end-points, the number of k-cycles, and the number of trees with k vertices that are a subgraph of the host graph.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Counting Subgraphs, #k-path, Dynamic Programming, Tree Decomposition, Determinant Method

Digital Object Identifier 10.4230/LIPIcs.IPEC.2025.20

1 Introduction

In this paper, we consider the problem of counting subgraphs isomorphic to a graph H in a given graph G. This well-studied problem has several applications, e.g., in bio-informatics when modeling biological networks [1, 12, 13] or when analyzing social networks [8, 17]. The frequency of specific subgraphs may give valuable insights into the functionality of a network and can be used to test the quality of network models.

Counting problems are generally much harder than their *decision* counterparts, asking whether a subgraph H occurs in G at all. Indeed, decision problems with a trivial solution may have a counting variant that is as hard as the counting variant of some NP-complete problems, and many counting problems are $\#P$-Complete [14]. For this reason, most research focuses on the *parameterized* complexity of subgraph counting problems.

To categorize the complexity of intractable parameterized counting problems, a complexity class hierarchy $\#W[t], t \geqslant 1$ analogous to the W-hierarchy of Downey and Fellows was described independently by Flum and Grohe [9] and McCartin [11]. Notably, Flum and Grohe showed that, while deciding whether there is a path of length k in a given graph is FPT, counting k-paths parameterized by k is $\#W[1]$-complete. This makes it unlikely that there is a polynomial time algorithm that counts k-paths even when k is small.

The main contributions of this paper are new and faster algorithms to count respectively paths and trees in graphs of bounded treewidth. First, an $O(18^{tw}k^2n)$ time Dynamic Program for counting the number of k-paths in a host graph G is presented. Although the counting of k-paths parameterized by path length k has been researched a lot [2, 3, 5, 10], exact algorithms parameterized by treewidth have seen little attention. The currently fastest exact algorithm for counting k-paths parameterized by treewidth is a well-known elementary Dynamic Program using a tree decomposition of G to solve the problem in $tw^{O(tw)}poly(k, n)$

time. The presented Dynamic Program improves on this by providing a *single-exponential time* algorithm for the problem, meaning that a solution is found in $O(c^{O(tw)} poly(k, n))$ time for a constant c.

The second contribution of this paper is an $\ell^{O(tw)} \left(\frac{k}{\ell}\right)^{O(\ell)} n$ time algorithm for $\#k, \ell$-Tree, the problem of counting the number of copies of a tree T with k vertices and ℓ leaves in a host graph G. This result generalizes the k-path counting algorithm in the first part of the paper. Since k-paths are trees, the counting of labeled trees is at least as hard as counting k-paths, showing $\#W[1]$-hardness when parameterized by k. Verstraete and Mubayi proved a lower bound for the number of labeled copies of tree T in host graph G of $(1-\epsilon)nd(d-1)...(d-t+1)$, where n is the number of vertices in G, d is the average degree of vertices in G, t is the number of edges in T and $\epsilon = \frac{(4t)^5}{d^2}$, provided that $d \geqslant t$ [16]. The current fastest exact algorithm to solve $\#k, \ell$-Tree parameterized by treewidth is an elementary Dynamic Program that takes $k^{O(tw)} 2^{O(k)} n$ time. The presented algorithm, therefore, improves on this running time when ℓ is significantly smaller than k and even provides a $2^{O(tw)} poly(k, n)$ time algorithm when ℓ is constant.

The techniques presented in this paper build upon the determinant method introduced by Bodlaender et al. in 2015 [4]. This method counts classes of subgraphs that are isomorphic to trees and include a specific vertex, such as Hamiltonian Paths and Steiner Trees. To develop single-exponential time algorithms for $\#k$-Path and $\#k, \ell$-Tree, we relax the constraint of including a specific vertex. This modification enables the counting of subgraphs without restrictions on which vertices are included. Additionally, we demonstrate that the determinant method can be extended to count subgraphs formed by “gluing together” simpler graphs.

Furthermore, the constructed Dynamic Programs can be relatively straightforwardly modified to count the number of fixed end-point k-paths and k-cycles, and to solve $\#k$-Tree (asking for the number of occurrences of *all* trees with k edges in host graph G) in $O(18^{tw} k^4 n)$ time. This last problem is proven to be $\#W[1]$-hard when parameterized by k by Brand and Roth [6].

2 Preliminaries

Nice Tree decompositions

In this paper, we utilize nice tree decompositions, following the definition provided by Cygan et al. [7].

Counting trees using determinants

This section explores a determinant-based method for counting subgraphs, developed by Bodlaender et al. [4].

Given an undirected graph $G = (V, E)$ and a nice path or tree decomposition $\mathbb{T}$ of G with width pw or tw respectively, define the *incidence matrix* of G as follows:

▶ **Definition 1** (incidence matrix). *The incidence matrix $A = (a_{v,e})$ of a graph G is an $n \times m$ matrix whose rows are indexed by the vertices in V and whose columns are indexed by the edges in E. Its entries are:*

$$a_{v,e} = \begin{cases} 1, & \textit{if } e = (v,w) \textit{ and } v > w \\ -1, & \textit{if } e = (v,w) \textit{ and } v < w \\ 0, & v \notin e, \end{cases} \tag{1}$$

where we use the inherent ordering of vertices in $\mathbb{T}$. Let $V(S)$ be the set of end-points of edges in S; that is, $V(S) = \{v \mid (v, u) \in S\}$. Now, consider the *reduced incidence matrix* of an edge set $S \subseteq E$:

▶ **Definition 2** (reduced incidence matrix). *Let $S \subseteq E$, and $p \in V(S)$. The $(|V(S)| - 1) \times |S|$ matrix $A_p[S]$ is the submatrix of A whose rows are those of A with indices in $V(S) \setminus \{p\}$, and whose columns are those of A with indices in S.*

Leveraging the definition of reduced incidence matrices, we can explore a central lemma in the determinant-based counting method; this lemma is used as a step in the proof for the Matrix Tree Theorem (see, for instance, the combination of Lemmas 2.2 and 2.5 in [15]).

▶ **Lemma 3.** *Let $S \subseteq E$ such that $|S| = |V(S)| - 1$. For any vertex p in $V(S)$, the following holds: if $(V(S), S)$ is a tree, then $|\det(A_p[S])| = 1$. Otherwise, $|\det(A_p[S])| = 0$.*

▶ **Corollary 4.** *Let $S \subseteq E$ such that $|S| = |V(S)| - 1$. If $(V(S), S)$ is a tree, then $\sum_{p \in V(S)} |\det(A_p[S])| = |V(S)|$. Otherwise, $\sum_{p \in V(S)} |\det(A_p[S])| = 0$.*

Corollary 4 allows us to count the number of trees within a collection of edge sets $\mathcal{S} \subseteq \mathcal{P}(E)$ by calculating the value of

$$\sum_{S \in \mathcal{S}} \left(\frac{1}{|V(S)|} \cdot \sum_{p \in V(S)} |\det(A_p[S])| \right), \tag{2}$$

provided that that $|S| = |V(S)| - 1$ for all $S \in \mathcal{S}$.

3 Counting k-paths

Let $G = (V, E)$ be an undirected graph, and let $k \in \mathbf{N}$. We address the following problem:

#k-Path
Input: A graph $G = (V, E)$ and a parameter $k \in \mathbf{N}$
Output: $P_k(G)$; the number of (not necessarily induced) subgraphs of G that are isomorphic to a simple (non-overlapping) path with exactly k vertices.

This section details a Dynamic Programming algorithm for solving the #k-Path problem. Our method first identifies a specific collection of edge sets, $\mathcal{S}$, such that the set of all trees in $\mathcal{S}$ is identical to the set of all simple paths within the graph G. We then utilize $\mathcal{S}$ within Corollary 4 to establish a partial sum constraint that all Dynamic Programming entries must satisfy.

3.1 Partial sum

In line with Corollary 4, we define the collection of edge sets $\mathcal{S}$ as follows:

$$\mathcal{S} = \{S \subseteq E \mid \forall v \in V(S), \deg_S(v) \leqslant 2; |S| = |V(S)| - 1\}.$$

An edge set $S \subseteq E$ constitutes a simple path if and only if the graph $(V(S), S)$ is a tree where every vertex has a degree of at most two. Consequently, the set of trees in $\mathcal{S}$ is identical to the set of simple paths within the graph G. The condition that $|S| = |V(S)| - 1$ for all

$S \in \mathcal{S}$ is inherently satisfied by all trees and thus does not alter the number of trees, but does ensure the compatibility of $\mathcal{S}$ with Corollary 4. The number of simple paths with length k in graph G is now given by

$$\begin{aligned} P_k(G) &= \sum_{S\in\mathcal{S}} [(V(S), S) \text{ is a tree}] \\ &= \frac{1}{k} \sum_{S\in\mathcal{S}, |S|=k-1} \sum_{p\in V(S)} |\det(A_p[S])|. \end{aligned} \tag{3}$$

The condition that $|S| = |V(S)| - 1$, while perhaps intuitive, complicates our Dynamic Programming design by requiring us to track both the number of edges and vertices within any edge set. A more computationally efficient strategy involves counting only the total number of vertices and the number of degree-one vertices. This is particularly advantageous because a simple path invariably has exactly two degree-one vertices. Consequently, we will employ the following proposition:

▶ **Proposition 5.** *Suppose $S \subseteq E$ is a nonempty set such that $deg_S(v) \leq 2$ for all $v \in V(S)$. Then the number of degree one vertices in $V(S)$ is equal to two if and only if $|S| = |V(S)| - 1$.*

With the equivalence from Proposition 5, we can rewrite our definition of $\mathcal{S}$ (without changing $\mathcal{S}$ itself):

$$\mathcal{S} = \{S \subset E \mid \forall v \in V(S), \deg_S(v) \leqslant 2; |\{v \in V(S) \mid \deg_S(v) = 1\}| = 2\}. \tag{4}$$

We can expand the determinant in Equation 3 to obtain

$$P_k(G) = \frac{1}{k} \sum_{S\in\mathcal{S}, |S|=k-1} \sum_{p\in V(S)} \left(\sum_{f:S \overset{1-1}{\to} V(S)\setminus\{p\}} \operatorname{sgn}(f) \prod_{e\in S} a_{f(e),e} \right)^2 \tag{5}$$

$$= \frac{1}{k} \sum_{S\in\mathcal{S}, |S|=k-1} \sum_{p\in V(S)} \sum_{\substack{f_1:S \overset{1-1}{\to} V(S)\setminus\{p\} \\ f_2:S \overset{1-1}{\to} V(S)\setminus\{p\}}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e\in S} a_{f_1(e),e} a_{f_2(e),e}. \tag{6}$$

The bijection $f : S \overset{1-1}{\to} V(S) \setminus p$ maps each edge $e = (u, v)$ in S to a unique vertex $f(e)$ in $V(S) \setminus p$. Observe that if $f(e)$ is distinct from both u and v, the corresponding term in the sum becomes zero due to the zero entry $a_{f(e),e}$ in the incidence matrix. Utilizing the property that $|\det(A_p[S])| \in 0, 1$, we have $|\det(A_p[S])| = (\det(A_p[S]))^2$.

To compute Equation 6, we will utilize Dynamic Programming on a given tree decomposition $\mathbb{T}$ of G. For this purpose, we define the following parameters:

- For every bag $y \in \mathbb{T}$, $s_{\deg} \in \{0, 1, 2\}^{|B_y|}$ denotes the degree of every vertex in B_y in $G[S]$
- For every bag $y \in \mathbb{T}$, $s_1 \in \{0, 1\}^{|B_y|}$ denotes for every vertex in B_y whether it has been used as a value in the bijection f_1
- For every bag $y \in \mathbb{T}$, $s_2 \in \{0, 1\}^{|B_y|}$ denotes for every vertex in B_y whether it has been used as a value in the bijection f_2
- Let t denote the number of forgotten vertices (in $V_y \setminus B_y$) that have degree one in S
- Let l denote the number of edges in S

Furthermore, we define the following useful sets:

- Let $\mathcal{S}_y(s_{\deg}, t, l)$ be the collection of all edge sets $S \subseteq E_y$ with exactly l edges, with t vertices of degree one in $V(S) \setminus B_y$, and such that the degree of a vertex v in B_y is equal to $s_{\deg}(v)$ and the degree of all vertices is at most two.

- Consider a set of edges $S \subseteq E_y$, and a set of vertices $P \subseteq V(S) \setminus B_y$. For $i \in 1, 2$, we define $\mathcal{F}_y(S, P, s_i)$ as the set of all bijective functions f that map each edge in S to a unique vertex in $V(S) \setminus (s_i^{-1}(0) \cup P)$.
  Intuitively, $\mathcal{F}_y(S, P, s_i)$ represents the possible bijections between the edges in S and a selection of vertices from $V(S)$. Since Proposition 5 states that $|V(S)| > |S|$, not all vertices in $V(S)$ can be part of this bijection. The sets $s_i^{-1}(0)$ and P both identify the vertices that are excluded from this mapping. A key distinction is that $s_i^{-1}(0)$ contains vertices within the bag B_y, whereas P contains vertices that are strictly outside of B_y. For a valid k-path we will have that $|P| = 1$ as in Equation 6. However, during the construction of this k-path, we might encounter intermediate states with multiple connected components. In such cases, $|P|$ might need to be greater than one to ensure a valid bijection exists, as more vertices need to be excluded from $V(S)$. Ultimately, we are guaranteed that $|P| = 1$. This is because if $|P| = 0$, the resulting edge sets form cycles, leading to a determinant of zero. Conversely, if $|P| > 1$, the resulting edge sets become disconnected, implying more than two vertices with degree one.

For every entry of the Dynamic Program in bag y we will require that the following partial sum holds:

$$A_y(s_{\deg}, s_1, s_2, t, l) =$$

$$\overset{(1)}{\sum_{S \in \mathcal{S}_y(s_{\deg}, t, l)}} \; \overset{(2)}{\sum_{P \subseteq V(S) \setminus B_y}} \; \overset{(3)}{\sum_{\substack{f_1 \in \mathcal{F}_y(S, P, s_1) \\ f_2 \in \mathcal{F}_y(S, P, s_2)}}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \overset{(4)}{\prod_{e \in S}} a_{f_1(e), e} a_{f_2(e), e}, \tag{7}$$

where we have labeled the Σ's and Π for later reference. Filling in variables, we see that in a root bag z where $E_z = E$ and $B_z = \emptyset$, we have that $\frac{1}{k} A_z(\emptyset, \emptyset, \emptyset, 2, k-1) = P_k(G)$, the number of k-paths in G.

3.2 Dynamic program

This section details the computation of tables for the functions A_y (defined in Equation 7) for each node type in nice tree decompositions.

Leaf bag y

$$A_y(\emptyset, \emptyset, \emptyset, t, l) = \begin{cases} 1, & \text{if } t = l = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

In a leaf bag, we have that $E_y = B_y = \emptyset$. Filling this into Equation 7 we see that Σ (1) has no summands unless $t = l = 0$. In this case, Σ (1) has one summand with $S = \emptyset$, Σ (2) has one summand with $P = \emptyset$, Σ (3) has one summand with $f_1 = f_2 = \emptyset \mapsto \emptyset$ (so that $\operatorname{sgn}(f_1) = \operatorname{sgn}(f_2) = 1$), and Π (4) has no elements.

Introduce vertex u bag y with child z

$$A_y(s_{\deg}, s_1, s_2, t, l) = \begin{cases} A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, t, l) & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) \\ & \quad = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Since vertex u is introduced in this bag, no edge in E_y is incident to u. Therefore $\deg_S(u) = 0$ for all $S \in E_y$, and Σ (1) in Equation 7 has no summands if $s_{\deg}(u) \neq 0$. Similarly, all summands in Σ (3) vanish whenever $s_1(u) \neq 0$ or if $s_2(u) \neq 0$.

If $s_{\deg}(u) = s_1(u) = s_2(u) = 0$, then $A_y(s_{\deg}, s_1, s_2, t, l)$ reduces to $A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, t, l)$.

Forget vertex u bag y with child z

When $t \in \{1, 2\}$, we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = &\, A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ &+ A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t-1, l) \\ &+ A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t-1, l) \\ &+ A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ &+ A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l). \end{aligned}$$

If $t = 0$, we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = &\, A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ &+ A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l) \\ &+ A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l). \end{aligned}$$

In a forget vertex u bag y, the vertex u is excluded from B_y. We determine the number of satisfying edge sets by summing over the possible degrees of u.

When $\deg_S(u) = 0$, the edge sets that satisfy Equation 7 are the same as the edge sets in $\mathcal{S}_z(s_{\deg}[u \leftarrow 0], t, l)$.

When $\deg_S(u) = 1$, the edge sets satisfying Equation 7 in bag y are the same as the edge sets in child z that have one less degree one vertex in $S \setminus B_z$. If $u \in P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t-1, l)$ if $t \geq 1$, and zero otherwise. When $u \notin P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 1], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t-1, l)$ when $t \geq 1$, and zero otherwise.

Finally, when $\deg_S(u) = 2$ we see that A_y reduces to $A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 0], s_2[u \leftarrow 0], t, l)$ when $u \in P$. When $u \notin P$, A_y reduces to $A_z(s_{\deg}[u \leftarrow 2], s_1[u \leftarrow 1], s_2[u \leftarrow 1], t, l)$.

Introduce edge $d = (u, w)$ bag y with child z

For $s_{\deg}(u) > 0$ and $s_{\deg}(w) > 0$ we set

$$\begin{aligned} A_y(s_{\deg}, s_1, s_2, t, l) = &\, A_z(s_{\deg}, s_1, s_2, t, l) + (-1)^{\mathrm{inv}(s_1^{-1}(1), u') + \mathrm{inv}(s_2^{-1}(1), w')} \\ \times &\sum_{\substack{u' \in s_1^{-1}(1) \cap d \\ w' \in s_2^{-1}(1) \cap d}} A_z(s'_{\deg}, s_1[u' \leftarrow 0], s_2[w' \leftarrow 0], t, l-1) \cdot a_{u',d} \cdot a_{w',d}, \end{aligned}$$

where $s'_{\deg}$ is equal to $s_{\deg}$, except that $s'_{\deg}(u) = s_{\deg}(u) - 1$, and $s'_{\deg}(w) = s_{\deg}(w) - 1$. If $s_{\deg}(u) = 0$ or $s_{\deg}(w) = 0$, we can not have that $d \in S$. In this case, we set

$$A_y(s_{\deg}, s_1, s_2, t, l) = A_z(s_{\deg}, s_1, s_2, t, l).$$

In an introduce edge d bag we have two options: either d is included in S, or d is not included in S. If d is not included in S, then A_y reduces to $A_z(s_{\deg}, s_1, s_2, t, l)$.

If $d = (u, w)$ is included in S, we have to account for all possible bijections f_1 and f_2. First note that for all bijections f where $f(d) \notin d$, we have that $a_{f(d),d} = 0$ and the term vanishes. Therefore we fix $f_i(d) = u$ or $f_i(d) = w$.

The inclusion of d into the bijections might change their sign. Since for all $v \in B_y$ and $x \in V_y \setminus B_y$ we have that $v > x$, no vertex in $V_y \setminus B_y$ can contribute to a change in sign. Also, since d is introduced in bag y, we have that $d > e$ for all $e \in E_y \setminus \{d\}$. The change in sign of f_i is therefore equal to $(-1)^{\mathrm{inv}(s_i^{-1}(1), f_i(d))}$.

Join bag y with children z_1 and z_2

$$A_y(s_{\deg}, s_1, s_2, t, l) = \sum_{\substack{s_{\deg,z_1}+s_{\deg,z_2}=s_{\deg}\\ s_{1,z_1}+s_{1,z_2}=s_1\\ s_{2,z_1}+s_{2,z_2}=s_2\\ t_{z_1}+t_{z_2}=t\\ l_{z_1}+l_{z_2}=l}} A_{z_1}(s_{\deg,z_1}, s_{1,z_1}, s_{2,z_1}, t_{z_1}, l_{z_1}) A_{z_2}(s_{\deg,z_2}, s_{1,z_2}, s_{2,z_2}, t_{z_2}, l_{z_2}) \cdot (-1)^{\mathrm{inv}(s_{1,z_1}^{-1}(1), s_{1,z_2}^{-1}(1)) + \mathrm{inv}(s_{2,z_1}^{-1}(1), s_{2,z_2}^{-1}(1))}, \tag{10}$$

where we use coordinate-wise addition.

3.3 Running time analysis

Given a host graph G together with a nice path or tree decomposition, we can traverse the decomposition in post-ordering, calculating entries according to the Dynamic Program described above. This procedure gives rise to the following theorem:

▶ **Theorem 6.** *There exists a deterministic algorithm that, given a graph G, a parameter k and a nice tree decomposition of G with treewidth tw, can solve #k-Path in $18^{tw} tw^{O(1)} k^2 n$ time. Given a nice path decomposition with pathwidth pw, there exists an algorithm that can solve #k-Path in $7^{pw} pw^{O(1)} kn$ time.*

4 Counting trees

In this section, we will discuss a generalization of the k-path problem where we count subgraphs isomorphic to a tree T.

Let $G = (V, E)$ be an undirected graph. Then the #k, ℓ-Tree problem on G is defined as:

#k, ℓ-TREE
Input: A graph $G = (V, E)$ and a tree T that has ℓ leaf nodes and k vertices
Output: $T_k(G)$; the number of (not necessarily induced) subgraphs of G that are isomorphic to T, not accounting for internal symmetries in T.

To count occurrences of a target tree T using Dynamic Programming, we will decompose it into path components. This approach allows us to build upon the path-counting methods detailed in Section 3. However, vertices with a degree of three or more, acting as convergence points for multiple paths, will pose a specific challenge requiring dedicated attention.

4.1 Tree labeling

To count occurrences of a specific tree T, we first need to establish a unique labeling for it. This labeling is derived by decomposing T into path segments, a process that considers three cases based on the degree of each vertex v in T:

- $\deg_T(v) = 1$; v is a **leaf**.
- $\deg_T(v) = 2$; v is a **path node**.
- $\deg_T(v) > 2$; v is an **internal junction**.

We define the path segments as follows:

- Starting in every leaf node with an unlabeled edge, we follow the unique path along the (possibly empty) set of connected path nodes, until we encounter an internal junction j_i or another leaf. The paths traced in this way are called the **leaf paths** of T. Note that if a leaf path terminates in two leaf nodes, T is isomorphic to a simple path, and its occurrences can be counted using the method in Section 3. For the remainder of this paper, we assume all leaf paths end in one leaf and one internal junction.
- For each internal junction, trace the paths starting in its unlabeled edges along the (possibly empty) set of connected path nodes until we encounter another internal junction (note that we can not encounter a leaf node, as per the previous step). These path segments are called **internal paths**.

For an illustration of this procedure, see Figure 1. By design, all edges along paths originating from leaves or internal junctions are labeled. Consequently, since any path not starting at such a point must form a cycle, all edges in the (acyclic) tree T are labeled.

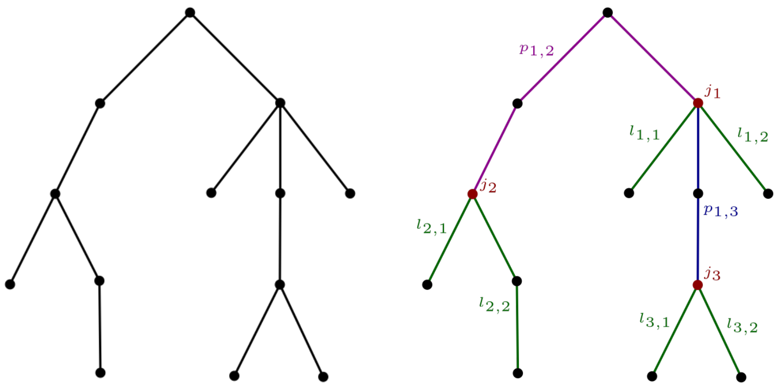

Figure 1 A tree T (left), together with its labeling (right). In the labeled tree, the internal junctions with degree at least 3 are labeled j_1, j_2, j_3, the internal paths are labeled $p_{q,r}$ for a path between j_q and j_r, and the leaf paths are labeled $l_{q,i}$ for the i'th leaf path originating from internal junction j_q.

Let $N = \{j_q\}_q$ be the set of internal junctions in T, let $I = \{p_{q,r} = (j_q, v_1, v_2, ..., j_r), ...\}$ the set of internal paths in T denoted by their vertices, and $L = \{l_{q,i} = (j_q, v_1, v_2, ..., v_k), ...\}$ the set of leaf paths in T denoted by their vertices. We will call (N, I, L) the labeling of T. Using this labeling, we can prove the following theorem, which will help us count the distinct graphs isomorphic to T:

▶ **Theorem 7.** *Let T be a tree with labeling (N, I, L), let $G = (V, E)$ be a graph, and let $S \subseteq E$ be a set of edges in G. Then the subgraph $(V(S), S)$ is isomorphic to T if and only if:*

- *$(V(S), S)$ is a tree*
- *There exists a labeling function $f : V(S) \to N \cup I \cup L$, such that:*

  1) ***Internal junctions are the image of exactly one vertex in $V(S)$***
  - *i.e. for all $v \in N$, we have that $|f^{-1}(v)| = 1$*

  2) ***Path length is preserved***
  - *i.e. for all internal paths $p_{q,r} \in I$, we have that $|f^{-1}(p_{q,r}) \cup f^{-1}(j_q) \cup f^{-1}(j_r)| = len_T(p_{q,r})$*
  - *and, for all leaf paths $l_{q,i} \in L$, we have that $|f^{-1}(l_{q,i}) \cup f^{-1}(j_q)| = len_T(l_{q,i})$*

  3) ***Vertex degrees in paths are preserved***
  - *i.e. for all internal paths $p_{q,r} \in I$, and for all vertices $v \in f^{-1}(p_{q,r})$, we have that $deg_S(v) = 2$.*
  - *and, for all leaf paths $l_q \in L$, there is exactly one vertex $t \in f^{-1}(l_q)$ such that $deg_S(t) = 1$. For all vertices $v \in f^{-1}(l_q) \setminus t$, we have that $deg_S(v) = 2$.*

  4) ***Neighborhoods are preserved***
  - *i.e. for all internal paths $p_{q,r} \in I$, and for all vertices $v \in f^{-1}(p_{q,r})$, let $N(v)$ denote the neighborhood of v in S. We have that $f(N(v)) \subseteq \{p_{q,r}, j_q, j_r\}$.*
  - *and, for all leaf paths $l_{q,i} \in L$, and for all vertices $v \in f^{-1}(l_{q,i})$, we have that $f(N(v)) \subseteq \{l_{q,i}, j_q\}$.*

4.2 Partial sum

Consider a graph $G = (V, E)$ and a subset of its edges $S \subseteq E$. Let T be a tree with k vertices, l leaves, and labeling $\mathcal{L}_T = (N, I, L)$. Define $\mathcal{F}_{\mathcal{L}_T}(S)$ as the set of functions $f : V(S) \to N \cup I \cup L$ satisfying the conditions of Theorem 7. If $T(G)$ denotes the number of distinct labeled subgraphs of G isomorphic to T, then Theorem 7 implies that

$$\begin{aligned} T(G) &= \sum_{\substack{S\subseteq E \\ \exists f\in\mathcal{F}_{\mathcal{L}_T}(S)}} [(V(S), S) \text{ is a tree}] \\ &= \frac{1}{k} \sum_{\substack{S\subseteq E \\ |S|=|V(S)|-1 \\ \exists f\in\mathcal{F}_{\mathcal{L}_T}(S)}} \sum_{p\in V(S)} |\det(A_p[S])|, \end{aligned} \tag{11}$$

where the second step follows from Corollary 4. Consider any edge set S. If there exists a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$, then by its defining properties (1) and (2), we have:

- $|f^{-1}(N)| = |N|$
- $|f^{-1}(p_{q,r} \cup j_q \cup j_r)| = \text{len}(p_{q,r})$ for all internal paths $p_{q,r}$
- $|f^{-1}(l_{q,i} \cup j_q)| = \text{len}(l_{q,i})$ for all leaf paths $l_{q,i}$

Since every vertex in $V(T)$ is either an internal junction or belongs to an internal/leaf path, it follows that $|V(S)| = |f^{-1}(N \cup I \cup L)| = |V(T)|$. Consequently, the condition $|S| = |V(S) - 1|$ in the first summation simplifies to $|S| = k - 1$. This allows us to expand the determinant in Equation 11 analogously to Section 3, resulting in

$$T(G) = \frac{1}{k} \sum_{\substack{S\subseteq E \\ |S|=k-1 \\ \exists f\in\mathcal{F}_{\mathcal{L}_T}(S)}} \sum_{p\in V(S)} \sum_{\substack{f_1:S\overset{1-1}{\to}V(S)\setminus\{p\} \\ f_2:S\overset{1-1}{\to}V(S)\setminus\{p\}}} \text{sgn}(f_1)\text{sgn}(f_2) \prod_{e\in S} a_{f_1(e),e}a_{f_2(e),e}. \tag{12}$$

We now construct the partial sum that defines each entry in our Dynamic Program. The definitions of $s_{\deg}, s_1, s_2$, and $\mathcal{F}_y(S, P, s_i)$ are retained from Section 3. Furthermore, we will define new parameters:

- For every bag $y \in \mathbb{T}$, let $s_a \in (N \cup I \cup L \cup \{0\})^{|B_y|}$ denote the label assigned to every vertex in bag y. Relating to Theorem 7, $s_a(v)$ can be interpreted as $f(v)$ when restricted to the vertices v in B_y.
- Let $\alpha \in \{0,1\}^{|N|}$ denote for every internal junction j_q in N whether some vertex $v \in V(S) \cup B_y$ should be labeled with j_q in the construction of a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$. The vector α is indexed by the internal junctions in N, and we will denote by $\alpha(j_q)$ the value corresponding to internal junction j_q. Conversely, we will denote by $\alpha^{-1}(1)$ all internal junctions in N for which α is equal to one.
- Let $\lambda \in \mathbb{N}^{|I \cup L|}$ denote for every internal path in I and leaf path in L how many vertices in $V(S)$ should be labeled with this path in the construction of a function $f \in \mathcal{F}_{\mathcal{L}_T}(S)$. The vector λ is indexed by the internal and leaf paths in $I \cup L$, and we will denote by $\lambda(p)$ the value corresponding to the path p. Conversely, we will denote by $\lambda^{-1}(\mathbb{N}_{\geqslant 1})$ all paths in $I \cup L$ for which λ is at least one.
- Let $t_{leaf} \in \{0,1\}^{|L|}$ denote for every leaf path $l_{q,i}$ in L the number of forgotten vertices v with degree one for which $f(v)$ should map to $l_{q,i}$ in the construction of some $f \in \mathcal{F}_{\mathcal{L}_T}(S)$.
- Let $\mathcal{S}_y(s_{\deg}, s_a, \alpha, \lambda, t_{leaf})$ be the collection of all edge sets $S \subseteq E_y$ such that there exists a labeling function $f_{y,S} : V(S) \cup B_y \to \alpha^{-1}(1) \cup \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cup \{0\}$ that obeys the following requirements:

  0) ***The functions $f_{y,S}$ and s_a are identical when restricted to bag vertices***
  - i.e. for all bag vertices $v \in B_y$, we have that $f_{y,S}(v) = s_a(v)$

  1) ***All internal junctions in $\alpha^{-1}(1)$ are the image of exactly one vertex in $V(S) \cup B_y$ under $f_{y,S}$***
  - i.e. for all internal junctions $v \in \alpha^{-1}(1)$, we have that $|f^{-1}_{y,S}(v)| = 1$

  2) ***All paths p in $I \cup L$ are the image of exactly $\lambda^{-1}(p)$ vertices in $V(S) \cup B_y$ under $f_{y,S}$***
  - i.e. for all paths $p \in \lambda^{-1}(\mathbb{N}_{\geqslant 1})$, we have that $|f^{-1}_{y,S}(p)| = \lambda(p)$

  3) ***Vertex degrees are preserved in forgotten vertices and are equal to s_{deg} in bag vertices***
  - i.e. for all internal junctions $j_q \in \alpha^{-1}(1)$, if the vertex $v \in f^{-1}_{y,S}(j_q)$ (there is exactly one such vertex as per (1)) is a forgotten vertex, then $\deg_S(v) = \deg_T(j_q)$. If v is a bag vertex, then $\deg_S(v) = s_{\deg}(v)$.
  - for all internal paths $p_{q,r} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap I$, we have that all forgotten vertices $v \in f^{-1}_{y,S}(p_{q,r}) \setminus B_y$ have degree two in S. All bag vertices $v \in f^{-1}_{y,S}(p_{q,r}) \cap B_y$ have degree $s_{\deg}(v)$ in S.
  - and, for all leaf paths $l_{q,i} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap L$, we have that exactly $t_{leaf}(l_{q,i})$ forgotten vertices in $f^{-1}_{y,S}(l_{q,i}) \setminus B_y$ have degree one in S, and all other vertices in $f^{-1}_{y,S}(l_{q,i}) \setminus B_y$ have degree two in S. All bag vertices $v \in f^{-1}_{y,S}(l_{q,i}) \cap B_y$ have degree $s_{\deg}(v)$ in S.

  4) ***Neighborhoods are preserved***
  - i.e. for all internal paths $p_{q,r} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap I$, and for all vertices $v \in f^{-1}_{y,S}(p_{q,r})$, let $N(v)$ denote the neighborhood of v in S. We have that $f_{y,S}(N(v)) \subseteq \{p_{q,r}, j_q, j_r\}$.
  - and, for all leaf paths $l_{q,i} \in \lambda^{-1}(\mathbb{N}_{\geqslant 1}) \cap L$, and for all vertices $v \in f^{-1}_{y,S}(l_{q,i})$, we have that $f_{y,S}(N(v)) \subseteq \{l_{q,i}, j_q\}$.

With the parameters defined above, we will determine the Dynamic Program entries A_y for each bag y such that

$$A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) =$$
$$\overset{(1)}{\sum_{S\in\mathcal{S}_y(s_{\deg},s_a,\alpha,\lambda,t_{leaf})}} \ \overset{(2)}{\sum_{P\subseteq V(S)\setminus B_y}} \ \overset{(3)}{\sum_{\substack{f_1\in\mathcal{F}_y(S,P,s_1)\\ f_2\in\mathcal{F}_y(S,P,s_2)}}} \mathrm{sgn}(f_1)\mathrm{sgn}(f_2) \overset{(4)}{\prod_{e\in S}} a_{f_1(e),e}a_{f_2(e),e}. \tag{13}$$

Consider a vector $\lambda_T \in \mathbb{N}^{|I\cup L|}$ indexed by all internal paths (I) and leaf paths (L) of $\mathcal{L}_T$. For each internal path $p \in I$, the corresponding value in λ_T is $\mathrm{len}(p) - 2$, and for each leaf path $p \in L$, the value is $\mathrm{len}(p) - 1$. Then, in a bag y with $E_y = E$ and $B_y = \emptyset$, the expression $\frac{1}{k}A_y(\emptyset, \emptyset, \emptyset, \emptyset, 1^{|N|}, \lambda_T, 1^{|L|})$ precisely equals Equation 12, thus yielding the number of subgraphs in G isomorphic to T.

4.3 Dynamic program

In this section, we will describe a Dynamic Program that iteratively calculates all partial sums of Equation 13. The entry calculations are accompanied by a brief explanation.

Leaf Bag y

$$A_y(\emptyset, \emptyset, \emptyset, \emptyset, \alpha, \lambda, t_{leaf}) = \begin{cases} 1, & \text{if } \alpha = 0^{|N|}, \lambda = 0^{|I\cup L|}, t_{leaf} = 0^{|L|}, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Introduce vertex u bag y with child z

$$A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) =$$
$$\begin{cases} A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, \alpha, \lambda, t_{leaf}) & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ & s_a(u) = \emptyset, \\ A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ \qquad \alpha[j_q \leftarrow 0], \lambda, t_{leaf}) & s_a(u) = j_q \in N, \alpha(j_q) = 1, \\ A_z(s_{\deg}|_{B_z}, s_1|_{B_z}, s_2|_{B_z}, s_a|_{B_z}, \alpha, & \text{if } s_{\deg}(u) = s_1(u) = s_2(u) = 0, \\ \qquad \lambda[p \leftarrow \lambda(p) - 1], t_{leaf}) & s_a(u) = p \in I \cup L, \lambda(p) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Because vertex u is introduced in this bag, no edge in E_y is incident to u, and therefore the partial sum reduces to zero if $s_{\deg}(u) \neq 0, s_1(u) \neq 0$ or $s_2(u) \neq 0$. Depending on the label $s_a(u)$ there are slightly different cases.

Forget vertex u bag y with child z

$$
\begin{aligned}
&A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \\
&\quad A_z(s_{\deg}[u \leftarrow 0], s_1[u \leftarrow 0], s_2[u \leftarrow 0], s_a[u \leftarrow 0], \alpha, \lambda, t_{leaf}) \\
&\quad + \sum_{p \in t_{leaf}^{-1}(1)} \begin{pmatrix} A_z(s_{\deg}[u{\leftarrow}1], s_1[u{\leftarrow}0], s_2[u{\leftarrow}0], s_a[u{\leftarrow}p], \alpha, \lambda, t_{leaf}[p{\leftarrow}0]) \\ +A_z(s_{\deg}[u{\leftarrow}1], s_1[u{\leftarrow}1], s_2[u{\leftarrow}1], s_a[u{\leftarrow}p], \alpha, \lambda, t_{leaf}[p{\leftarrow}0]) \end{pmatrix} \\
&\quad + \sum_{p \in \lambda^{-1}(\mathbb{N}_{\geqslant 1})} \begin{pmatrix} A_z(s_{\deg}[u{\leftarrow}2], s_1[u{\leftarrow}0], s_2[u{\leftarrow}0], s_a[u{\leftarrow}p], \alpha, \lambda, t_{leaf}) \\ +A_z(s_{\deg}[u{\leftarrow}2], s_1[u{\leftarrow}1], s_2[u{\leftarrow}1], s_a[u{\leftarrow}p], \alpha, \lambda, t_{leaf}) \end{pmatrix} \\
&\quad + \sum_{j_q \in \alpha^{-1}(1)} \begin{pmatrix} A_z(s_{\deg}[u{\leftarrow}\deg_T(j_q)], s_1[u{\leftarrow}0], s_2[u{\leftarrow}0], s_a[u{\leftarrow}j_q], \alpha, \lambda, t_{leaf}) \\ +A_z(s_{\deg}[u{\leftarrow}\deg_T(j_q)], s_1[u{\leftarrow}1], s_2[u{\leftarrow}1], s_a[u{\leftarrow}j_q], \alpha, \lambda, t_{leaf}) \end{pmatrix}.
\end{aligned} \tag{16}
$$

To obtain the total number of partial solutions in a forget bag we sum over all possible degrees of u in B_z (denoted as $s_{\deg,z}(u)$). The case that $s_{\deg,z}(u) = 0$ accounts for the edge sets S such that $u \notin S$. The case that $s_{\deg,z}(u) = 1$ accounts for the edge sets where u is a leaf, the case that $s_{\deg,z}(u) = 2$, accounts for the edge sets where u is a path node, and the case where $s_{\deg,z}(u) > 2$, accounts for the edge sets where u is an internal junction.

Introduce edge $d = (u, w)$ bag y with child z

Due to the preservation of neighborhoods requirement (4) for $f_{y,S}$, the edge d can only be included in S if there is some internal path $p_{q,r} \in I$ such that $s_a(u), s_a(w) \in \{p_{q,r}, j_q, j_r\}$, or if there is some leaf path $l_{q,i}$ such that $s_a(u), s_a(w) \in \{l_{q,i}, j_q\}$. Let $1_{I \cup L}(u, w)$ be the indicator function that is equal to one if there exists such a path, and zero otherwise. If $s_{\deg}(u) = 0$, $s_{\deg}(w) = 0$, or if $1_{I \cup L}(u, w) = 0$, we set

$$
A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}), \tag{17}
$$

Otherwise, we set

$$
\begin{aligned}
A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) &= A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) \\
&\quad + \sum_{\substack{u' \in s_1^{-1}(1) \cap d \\ w' \in s_2^{-1}(1) \cap d}} \begin{pmatrix} A_z(s'_{\deg}, s_1[u'{\leftarrow}0], s_2[w'{\leftarrow}0], s_a, \alpha, \lambda, t_{leaf}) \\ \cdot a_{u',d} \cdot a_{w',d} \cdot (-1)^{\mathrm{inv}(s_1^{-1}(1), u') + \mathrm{inv}(s_2^{-1}(1), w')} \end{pmatrix}.
\end{aligned} \tag{18}
$$

In an introduce edge d bag we have two options: either d is included in S, or it is not. In the case that $d \notin S$, then $A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf})$ reduces to $A_z(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf})$. If $d \in S$ we have to correctly account for the change in sign of bijections f_1 and f_2. This is done in the same way as in Section 3.2.

Join bag y with children z_1 and z_2

$$
\begin{aligned}
&A_y(s_{\deg}, s_1, s_2, s_a, \alpha, \lambda, t_{leaf}) = \\
&\sum_{\substack{s_{\deg,z_1} + s_{\deg,z_2} = s_{\deg} \\ s_{a,z_1} = s_{a,z_2} = s_a \\ \forall j_q \in N, \alpha_{z_1}(j_q) + \alpha_{z_2}(j_q) = \alpha(j_q) + |s_a^{-1}(j_q)| \\ \forall p \in I \cup L, \lambda_{z_1}(p) + \lambda_{z_2}(p) = \lambda(p) + |s_a^{-1}(p)| \\ t_{leaf,z_1} + t_{leaf,z_2} = t_{leaf} \\ s_{1,z_1} + s_{1,z_2} = s_1 \\ s_{2,z_1} + s_{2,z_2} = s_2}} \begin{matrix} A_{z_1}(s_{\deg,z_1}, s_{1,z_1}, s_{2,z_1}, s_{a,z_1}, \alpha_{z_1}, \lambda_{z_1}, t_{leaf,z_1}) \\ \cdot A_{z_2}(s_{\deg,z_2}, s_{1,z_2}, s_{2,z_2}, s_{a,z_2}, \alpha_{z_2}, \lambda_{z_2}, t_{leaf,z_2}) \\ \cdot (-1)^{\mathrm{inv}(s_{1,z_1}^{-1}(1), s_{1,z_2}^{-1}(1)) + \mathrm{inv}(s_{2,z_1}^{-1}(1), s_{2,z_2}^{-1}(1))} \end{matrix},
\end{aligned} \tag{19}
$$

where we use coordinate-wise addition.

4.4 Running time analysis

Using the Dynamic Program above we can prove the following theorem:

▶ **Theorem 8.** *There exists a deterministic algorithm that, given a graph G together with a nice tree decomposition of G with treewidth tw and given a tree T with ℓ leaf nodes and k vertices with $\ell < \frac{k-1}{2e}$, can solve $\#k, \ell$-Tree in $\ell^{O(tw)} tw^{O(1)} \left(\frac{k}{\ell}\right)^{O(\ell)} n$ time. Given a nice path decomposition of G with pathwidth pw, there exists an algorithm that solves $\#k, \ell$-Tree $\ell^{O(pw)} pw^{O(1)} \left(\frac{k}{\ell}\right)^{O(\ell)} n$ time.*

5 Conclusion

In this paper, we have given deterministic algorithms for the exact counting of k-paths and trees. These algorithms are single-exponential, parameterized by the tree- or pathwidth of the host graph G, and are linear in the number of vertices in G. Both algorithms utilize the determinant-based approach introduced by Bodlaender et al. [4] to construct a Dynamic Program. The algorithms in this paper can also be straightforwardly modified to count the number of cycles of length k, the number of fixed-endpoint k-paths, and the number of k-Trees in single-exponential time when parameterized by the treewidth of the host graph.

Before our work, no algorithms were known for the studied problems with single-exponential dependency on the treewidth. However, the base of the exponents of the algorithm to count trees is rather large. An interesting question is whether these bases can be improved. One possible way in which the algorithm could be improved is if a more efficient way of counting the length of the internal and leaf paths is found. Keeping track of how many edges are in each path quickly blows up the running time, as there are $\left(\frac{k}{\ell}\right)^{O(\ell)}$ possible combinations. A more efficient way of counting would significantly reduce the scaling of the running time with ℓ.

The techniques of this paper can be used to count even more general graph patterns. Suppose for example that some graph H is 'almost' acyclic. That is, there is some small set of edges C that, when removed, H becomes acyclic. Then we can treat the set of end-points $V(C)$ of those edges in C as internal junctions (so we can label vertices as end-points of such edges), and we can verify the existence of edges in G between those end-points without including them in the bijections. The running time of such an algorithm would be similar to that of the tree counting algorithm but with ℓ replaced with $\ell + |V(C)|$.

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008. `doi:10.1093/bioinformatics/btn163`.

2 Noga Alon and Shai Gutner. Balanced hashing, color coding and approximate counting. In *Proceedings 4th International Workshop on Parameterized and Exact Computation, IWPEC 2009*, pages 1–16. Springer, 2009. `doi:10.1007/978-3-642-11269-0_1`.

3 V. Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *Proceedings 13th International Symposium on Algorithms and Computation, ISAAC 2002*, pages 453–464. Springer, 2002. `doi:10.1007/3-540-36136-7_40`.

4 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. `doi:10.1016/j.ic.2014.12.008`.

5 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *ACM SIGACT 50th Annual Symposium on Theory of Computing, STOC 2018*, pages 151–164, 2018. doi:10.1145/3188745.3188902.

6 Cornelius Brand and Marc Roth. Parameterized counting of trees, forests and matroid bases. In *12th International Computer Science Symposium in Russia: Computer Science – Theory and Applications, CSR 2017*, pages 85–98. Springer, 2017. doi:10.1007/978-3-319-58747-9_10.

7 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M.M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

8 Alexandra Duma and Alexandru Topirceanu. A network motif based approach for classifying online social networks. In *Proceedings 9th IEEE International Symposium on Applied Computational Intelligence and Informatics, SACI 2014,*, pages 311–315. IEEE, 2014. doi:10.1109/SACI.2014.6840083.

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

10 Daniel Lokshtanov, Andreas Björklund, Saket Saurabh, and Meirav Zehavi. Approximate counting of k-paths: Simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3), 2021. doi:10.1145/3461477.

11 Catherine McCartin. Parameterized counting problems. *Annals of Pure and Applied Logic*, 138(1):147–182, 2006. doi:10.1016/j.apal.2005.06.010.

12 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

13 N. Pržulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, July 2004. doi:10.1093/bioinformatics/bth436.

14 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

15 Janneke van den Boomen. The Matrix Tree Theorem, 2007. Bachelor thesis, Radbout Universiteit Nijmegen. URL: https://www.math.ru.nl/~bosma/Students/jannekebc3.pdf.

16 Jacques Verstraete and Dhruv Mubayi. Counting Trees in Graphs. *The Electronic Journal of Combinatorics*, 23(3):P3.39, 2016. doi:10.37236/6084.

17 Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*, volume 8. Cambridge University Press, 1994. doi:10.1017/CBO9780511815478.