Algorithmica (2023) 85:2318–2347
https://doi.org/10.1007/s00453-023-01096-1

A Color-Avoiding Approach to Subgraph Counting in Bounded Expansion Classes

Felix Reidl[1] · **Blair D. Sullivan**[2]

Received: 22 November 2021 / Accepted: 4 January 2023 / Published online: 3 February 2023

Abstract

We present an algorithm to count the number of occurrences of a pattern graph H on h vertices as an induced subgraph in a host graph G. If G belongs to a bounded expansion class, the algorithm runs in linear time, if G belongs to a nowhere dense class it runs in almost-linear time. Our design choices are motivated by the need for an approach that can be engineered into a practical implementation for sparse host graphs. Specifically, we introduce a decomposition of the pattern H called a *counting dag* $\vec{C}(H)$ which encodes an order-aware, inclusion-exclusion counting method for H. Given such a counting dag and a suitable linear ordering $\mathbb{G}$ of G as input, our algorithm can count the number of times H appears as an induced subgraph in G in time $O(\|\vec{C}\| \cdot h \operatorname{wcol}_h(\mathbb{G})^{h-1}|G|)$, where $\operatorname{wcol}_h(\mathbb{G})$ denotes the maximum size of the weakly h-reachable sets in $\mathbb{G}$. This implies, combined with previous results, an algorithm with running time $O((3h^2 \operatorname{wcol}_h(G))^{h^2}|G|)$ which only takes H and G as input. We note that with a small modification, our algorithm can instead use strongly h-reachable sets with running time $O(\|\vec{C}\| \cdot h \operatorname{col}_h(\mathbb{G})^{h-1}|G|)$, resulting in an overall complexity of $O(h(3 \operatorname{col}_h(G))^{h^2}|G|)$ when only given H and G. Because orderings with small weakly/strongly reachable sets can be computed relatively efficiently in practice (Nadara et al.: in J Exp Algorithmics 103:14:1–14:16, 2018), our algorithm provides a promising alternative to algorithms using the traditional p-treedepth coloring framework (O'Brien and Sullivan in: Experimental evaluation of counting subgraph isomorphisms in classes of bounded expansion, CoRR, arXiv:1712.06690, 2017). We describe preliminary experimental results from an initial open source implementation which highlight its potential.

✉ Felix Reidl
f.reidl@dcs.bbk.ac.uk

Blair D. Sullivan
sullivan@cs.utah.edu

1 Birkbeck, University of London, London, UK

2 School of Computing, University of Utah, Salt Lake City, USA

Keyword Sparse graphs · Subgraph counting · Bounded expansion · Weak coloring number · Strong coloring number

1 Introduction

We consider the problem of counting the number of times a *pattern* graph H appears in a *host* graph G as an induced subgraph. Without any restrictions on G, this problem is already difficult for very simple H: Flum and Grohe [10] showed that it is #W[1]-complete when H is a clique and Chen, Thurley, and Weyer showed that it is #W[1]-complete when it is a path [2] (a related result by Chen and Flum shows that counting *maximal* paths is #W[1]-hard). That is, there is little hope for algorithms with running time $f(|H|) \cdot \mathsf{poly}|G|$ for these problems unless e.g. counting satisfying assignments of a 3-CNF formula is possible in time $2^{o(n)}$ (further details on parameterized counting classes can be found in Flum and Grohe's book [11]).

The situation is less glum when we restrict ourselves to sparse host graphs. For example, Eppstein, Löffler, and Strash showed that enumerating all cliques in a d-degenerate host graph G is possible in time $O(d \cdot 3^{d/3}|G|)$ [8]. More generally, we can count any pattern graph H on h vertices in time $O(f(h) \cdot |G|)$ provided that G is taken from a graph class of *bounded expansion* (where f depends on the class) and time $O(f(h) \cdot |G|^{1+o(1)})$ if it is taken from a *nowhere dense* graph class. These two classes generalize well-known notions like classes excluding a (topological) minor or classes of bounded degree and are therefore attractive targets for algorithmic research.

Intuitively, bounded expansion classes have the property that they have bounded average degree and this property is maintained when taking a minor of bounded depth (also called a *shallow* minor), meaning a minor whose branch sets are bounded by a constant (the 'depth'). The average degree of the graphs that can be obtained by such an operation is then only a function of the original graph class and the depth of the minor. Nowhere dense classes are similar, but instead of the average degree we have that the clique number remains constant under this operation. Classes with bounded expansion are degenerate, but there are degenerate classes that do not have bounded expansion (the simplest example are one-subdivision of complete graphs which are 2-degenerate but have neither bounded expansion nor are they nowhere dense). Nowhere dense classes generalise bounded expansion classes and allow a (slightly) superlinear number of edges, as such these classes are incomparable with degenerate classes.

In summary, bounded expansion is a very general notion of sparseness that allows us to design linear-time algorithms for problems that, in all likelihood, do not admit linear-time algorithms on general graphs. Moreover, we have evidence based on random graph models [3, 9, 22] as well as measurements [4, 17] that at least *some* types of networks can be described as "having bounded expansion".[1] We therefore believe that algorithm based on the bounded expansion toolkit can be useful to analyse real-world data, though making these algorithms practical is not without challenges.

[1] Since bounded expansion is a property of graph classes, this statement is of course not mathematically rigorous. See Sect. 5 for a brief discussion on this topic.

Case in point, the two existing types of approaches to count substructures in bounded expansion classes have not shown much promise in practice. One class of algorithms is based on so-called *p-treedepth colorings*: given a class $\mathcal{G}$ of bounded expansion we can color any $G \in \mathcal{G}$ in time $f(p) \cdot |G|$ with $f(p)$ colors so that any subgraph of G with $i < p$ colors has treedepth $\leq i$. By computing an h-treedepth coloring this effectively reduces the problem to counting H in a graph G' of treedepth $t \leq |H|$. Ossona de Mendez and Nešetřil, who also introduced the notion of bounded expansion [18] and nowhere dense classes [19], presented an algorithm for this latter step with a running time of $O(2^{ht} ht \cdot |G'|)$ [20]; with Demaine, Rossmanith, Sánchez Villaamil, and Sikdar we later improved this to $O((6t)^h h^2 \cdot |G'|)$ [3]. Using this subroutine, we can count occurrences of H in G by first computing an h-treedepth coloring with $f'(h)$ colors, then iterate through all $\sum_{i=1}^{h} \binom{f'(h)}{i}$ color combinations and count in the aforementioned time the number of times H appears in the subgraph G' induced by these colors. The final count is then computed via inclusion-exclusion over the counts obtained for the color sets.

While conceptually simple, it turns out that these algorithms are currently impractical: a) computing h-treedepth colorings is currently computationally quite expensive and b) the number of colors $f'(h)$ is so big that already the act of enumerating all relevant color subsets takes too long [21]. It turns out that the underlying technique for these algorithms—so-called transitive-fraternal augmentations [20] (tf-augmentations) with some practical improvements [21, 22]—also lies at the heart of the other available technique. Kazana and Segoufin used tf-augmentations to enumerate first-order queries with constant delay (or to count such queries in linear time) in classes with bounded expansion [14] and Dvořák and Tůma designed a dynamic data structure[2] to count subgraphs with amortized polylogarithmic updates [7]. The latter approach also has the drawback that in order to count induced subgraphs, one must perform a big inclusion-exclusion over all supergraphs of the pattern.[3]

Despite our best efforts to make tf-augmentations *practical*, so far they seem to be only useful in very tame settings like bounded-degree graphs [1]. It is thus natural to ask whether we can solve the subgraph-counting problem *without* relying on p-treedepth colorings or even tf-augmentations. In particular, the computation of so-called *generalized coloring numbers* (a set of graph measures introduced by Kierstead and Yang [15] which provide an alternative characterisation of bounded expansion/nowhere dense classes [24]), appears much more feasible in practice [17], and offers an attractive ordering-based alternative.

Our contribution here is to provide an algorithm to count induced subgraphs which is solely based on the *weak coloring number* (or the *coloring number*). At a high level, we do this by using a suitable linear order of the host graph and counting how often each of the possible pattern graph orders appears in it.[4] The crucial insight here is that under some orderings, the pattern graph can only appear inside certain neighborhood-

[2] To be precise this data structure only uses fraternal augmentations.

[3] Consider, for example, the graphs we need to count in order to compute the number of P_4s in a graph (Fig. 2). This list excludes e.g. K_4.

[4] We view these orderings as a type of graph decomposition and therefore assume they are part of the input. See the Preliminaries for a discussion on how a suitable ordering can be computed efficiently *in theory* and a discussion of *practical* approaches in Sect. 5.

subsets and that all other orderings can be reduced to these easily countable cases via inclusion-exclusion style arguments. Note that in contrast to Dvořák and Tůma's approach, the objects in our inclusion–exclusion are specific ordered graphs and we can therefore avoid counting *all* supergraphs of the pattern.

In order to establish the practicality of our approach, we implemented a prototype of the entire algorithmic pipeline described in this paper using a combination of Rust and Python. The code is available under a BSD 3-clause license at http://www.github.com/theoryinpractice/mandoline.

We begin in Sect. 2 by providing necessary definitions and notation related to ordered graphs, reachability and bounded expansion. We then describe our approach to decomposing the pattern graph and combining counts of partial matches in Sect. 3. We combine these subroutines with a new data structure in Sect. 4 to form the basis of our linear-fpt algorithm. Finally, in Sect. 5, we briefly discuss our experimental results and future work.

2 Preliminaries

We begin with a high-level description of how our algorithm works, in order to motivate the technical concepts and notation defined in this section. To that end, let us first review how complete subgraphs can be counted in linear time in a d-degenerate graph G: We first compute a degeneracy ordering of G, meaning that every vertex v has at most d *left neighbors* $N^-(v)$ i.e. neighbors of v that appear before v in the ordering. Assume some vertex set $X \subseteq V(G)$ induces a complete subgraph K_t in G. Let $v \in X$ be the last vertex of X in the ordering, then note that $X \subseteq N^-(v) \cup \{v\}$. Accordingly, we can count all complete subgraphs in G by exhaustively searching every left-neighborhood $N^-(v) \cup \{v\}$ for every vertex $v \in G$ and tally up the results.

In order to extend this algorithm, we need to generalise it in two directions. First, instead of using degeneracy orderings and left neighborhoods we will use *weak r-coloring* orderings and the related *weakly r-reachable sets*. Similar to the clique example, we find that for *some* pattern graphs H it is enough to search the weakly r-reachable set of each vertex to locate H. For other patterns, however, we can only locate *pieces* of the pattern and we need to combine these pieces in order to count appearances of H.

More specifically, our algorithm enumerates all possible orderings of the pattern H and counts how often the pattern appears *in this order* inside the ordered host graph. For most ordered patterns, the algorithm needs to decompose the pattern into pieces, a process that depends on both the ordering of the pattern as well as certain connectivity properties of the pattern itself. This is formalized by the notion of *tree ordered graphs* and *relaxations* below. The tree order informs how an ordered pattern needs to be decomposed into *piece-sums*, that is, a collection of pieces whose respective counts in the host graph can be combined to compute the number of times the ordered pattern appears.

Finally, the above process needs to be corrected as it will also count certain subgraphs which are *not* isomorphic to the pattern. We call these graphs *defects*—in short these graphs contain all the necessary pieces that we identified in the decomposition

of the pattern, but in such a way that pieces either share vertices or are connected by edges and hence do not form the sought pattern. In order to describe defects precisely, we introduce the notation of *embeddings* of tree-ordered graphs below. Our algorithm counts defects by the same procedure as it counts the patterns, i.e. it decomposes them into pieces and counts those individually, this in turn generates further defects to be counted *etc*. This recursion terminates since at every step the defects become denser and "more linear".

2.1 General Notation

We frequently use the notation $[n] := \{1, \dots, n\}$ for natural numbers n. The symbol $\bullet$ is reserved for variables whose value is unimportant in this contex, we write e.g. $f(\bullet)$ to emphasise that f has a single argument. In the proof of Lemma 6 we make heavy use of Iverson brackets: for any predicate S the expression $[\![S]\!]$ evaluates to 1 if S is true and 0 otherwise.

2.2 Trees

All trees in this paper will be assumed to be rooted. In particular, a *subtree* is always a rooted subtree. For a tree T, we write $\mathrm{root}(T)$ to denote its root and $\mathrm{leaves}(T)$ to denote its leaves. The *root path* $\mathrm{rpath}_T(x)$ for a node $x \in T$ is the unique path from $\mathrm{root}(T)$ to x in T.

The *ancestor relationship* $\preccurlyeq^T_{\mathsf{anc}}$ of a tree T is the partial order defined via

$$x \preccurlyeq^T_{\mathsf{anc}} y \iff x \in \mathrm{rpath}_T(y).$$

2.3 Partial and Total Orders

We will use the symbol $\preccurlyeq$ to denote partial orders and the symbol $\prec$ to denote the relation $(x \preccurlyeq y) \wedge (x \neq y)$. Given a partial order $\preccurlyeq$ over S, its *digraph representation* is a dag with vertices S and arcs $\{xy \in S \times S \mid x \prec y\}$.

The *principal digraph* of a partial order $\preccurlyeq$ over S is the dag with vertices S and the arcs

$$\{xy \in S \times S \mid x \prec y \text{ and there is no } z \in S \text{ with } x \prec z \prec y\}$$

Note that if $\vec{D}$ is the principal digraph of $\preccurlyeq$, S; then the transitive closure of $\vec{D}$ is the digraph representation of $\preccurlyeq$, S.

A partial order $\preccurlyeq$ over S is a *tree* if is has a unique minimum and if for every element $x \in S$, the set $\{y \mid y \preccurlyeq x\}$ is well-ordered by $\preccurlyeq$. Alternatively, $\preccurlyeq$ is a tree if its principal digraph is a directed tree, e.g. all arcs are oriented away from the root node.

A *linear extension* of $\preccurlyeq$ is a total order $\leq$ such that $x \preccurlyeq y$ implies $x \leq y$. The linear extensions of $\preccurlyeq$ are precisely the topological orderings of either its digraph representation or its principal digraph.

2.4 Ordered Graphs

A *tree ordered graph* (tog) $\mathbf{G} = (G, \preccurlyeq)$ is a graph whose vertex set $V(\mathbf{G}) := V(G)$ is imbued with a (partial) order relation $\preccurlyeq$ with the following properties:

1. The relation $\preccurlyeq$ is a *tree order*.
2. The relation $E(G)$ is *guarded* by $\preccurlyeq$: for every edge $uv \in E(G)$ it holds that either $u \preccurlyeq v$ or $v \preccurlyeq u$.

We define $T(\mathbf{G})$ to be the tree-representation of $\preccurlyeq$ with node set $V(\mathbf{G})$. We extend the notions and notation of roots, leaves, and root-paths to togs via $\mathrm{root}(\mathbf{G}) := \mathrm{root}(T(\mathbf{G}))$, $\mathrm{leaves}(\mathbf{G}) := \mathrm{leaves}(T(\mathbf{G}))$, and $\mathrm{rpath}_{\mathbf{G}}(\bullet) = \mathrm{rpath}_{T(\mathbf{G})}(\bullet)$. Given a tog $\mathbf{G}$ we write $\preccurlyeq_{\mathbf{G}}$ to denote its tree-order relation and we will use the notation $u \prec_{\mathbf{G}} v$ to mean that $u \preccurlyeq_{\mathbf{G}} v$ and $u \neq v$. An *ordered vertex set* $\bar{x} := x_1, \ldots, x_\ell$ of a tog $\mathbf{G}$ is a sequence of vertices which satisfies $x_1 \prec_{\mathbf{G}} x_2 \prec_{\mathbf{G}} \ldots \prec_{\mathbf{G}} x_\ell$. The *length* of an ordered vertex set $\bar{x}$ is the number of elements in it. We use the symbol $\emptyset$ to denote both the empty set and the empty ordered vertex set and make sure it is clear from the context which is meant.

If $\preccurlyeq_{\mathbf{G}}$ is a total order we call $\mathbf{G}$ a *linear graph*. We will use the symbol $\mathbb{G}$ (instead of $\mathbf{G}$) and $\leq_{\mathbb{G}}$ (instead of $\preccurlyeq_{\mathbf{G}}$) in cases were we want to emphasize that the ordering is linear. For a given graph G we write $\Pi(G)$ for the set of all linear graphs obtained from G by permuting its vertex set.

A tog isomorphism $\mathbf{H} \simeq \mathbf{G}$ is a bijection between the vertex sets of $\mathbf{H}$ and $\mathbf{G}$ that preserves both the edge and the ordering relations. Given a vertex set $X \subseteq V(\mathbf{G})$ with a unique minimum under $\preccurlyeq_{\mathbf{G}}$, the tog induced by X, denoted by $\mathbf{G}[X]$, is the tog $(G[X], \preccurlyeq_{\mathbf{G}}|_X)$. Note that we need X to have a unique minimum in order to ensure that $\preccurlyeq_{\mathbf{G}}|_X$ is still a tree order. Since $\preccurlyeq_{\mathbf{G}}$ guards the edge relation $E(G)$ this is already true if $G[X]$ is connected. In general, a tog $\mathbf{H}$ is an *induced subtog* of a tog $\mathbf{G}$ if there exists a vertex set X such that $\mathbf{H} \simeq \mathbf{G}[X]$ and we write $\mathbf{H} \subseteq \mathbf{G}$.

The *stem* of a tog $\mathbf{G}$ is the ordered set $\bar{x}$ of maximal length such that $\bar{x}$ is linearly ordered under $\preccurlyeq_{\mathbf{G}}$ and $\max \bar{x} \preccurlyeq_{\mathbf{G}} u$ for all vertices $u \in V(\mathbf{G}) - \bar{x}$. If we visualize $\preccurlyeq_{\mathbf{G}}$ as a tree then the stem is the path from the root to the first node with more than one child.

We say that a tog $\mathbf{H}$ *embeds* into a tog $\mathbf{G}$ if there exists an induced subgraph isomorphism ϕ from H to G that further satisfies

$$u \preccurlyeq_{\mathbf{H}} v \implies \phi(u) \preccurlyeq_{\mathbf{G}} \phi(v)$$

and we write $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbf{G}$. If we do not need to asign a variable for the embedding, we will simply write $\mathbf{H} \hookrightarrow \mathbf{G}$ or, in cases where we want specify that the embedding maps stem-vertices $\bar{x}$ of $\mathbf{H}$ onto $\bar{y}$ in $\mathbf{G}$, we write $\mathbf{H} \xhookrightarrow{\bar{x} \mapsto \bar{y}} \mathbf{G}$.

We further write $\mathbf{H} \overset{\phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{G}$ if $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbf{G}$ and ϕ is an isomorphism between H and G, we will call such embeddings *strict*. Again, we simply write $\mathbf{H} \hookrightarrow\!\!\!\!\rightarrow \mathbf{G}$ to denote that such a strict embedding exists or $\mathbf{H} \xhookrightarrow{\bar{x} \mapsto \bar{y}}\!\!\!\!\rightarrow \mathbf{G}$ if we want to specify the mapping of stem-vertices.

For a vertex set $X \subseteq V(\mathbf{G})$, we let $\min_{\mathbf{G}} X$ and $\max_{\mathbf{G}} X$ be the minimum and maximum according to $\preccurlyeq_{\mathbf{G}}$ (if they exist). We extend this notation to subtogs via $\min_{\mathbf{G}} \mathbf{H} := \min_{\mathbf{G}} V(\mathbf{H})$ and $\max_{\mathbf{G}} \mathbf{H} := \max_{\mathbf{G}} V(\mathbf{H})$. We observe that minima are preserved by tog embeddings:

Observation 1 *Let* $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbf{G}$ *and let* $\mathbf{H}' \subseteq \mathbf{H}$. *Then* $\min_{\mathbf{G}} \phi(\mathbf{H}') = \phi(\min_{\mathbf{H}} \mathbf{H}')$.

Proof Since $\mathbf{H}'$ is a subtog of $\mathbf{H}$ the minimum $\min_{\mathbf{H}} \mathbf{H}'$ exists. From $\min_{\mathbf{H}} \mathbf{H}' \preccurlyeq_{\mathbf{H}} u$ for every $u \in \mathbf{H}'$ and the fact that ϕ is an embedding we conclude that $\phi(\min_{\mathbf{H}} \mathbf{H}') \preccurlyeq_{\mathbf{G}} \phi(u)$ for every $u \in \mathbf{H}'$. Therefore $\min_{\mathbf{G}} \phi(\mathbf{H}') = \phi(\min_{\mathbf{H}} \mathbf{H}')$. □

This in particular implies that embeddings preserve ordered vertex sets: if $\bar{x}$ is an ordered vertex set of $\mathbf{H}$ and $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbf{G}$, then $\phi(\bar{x})$ is an ordered vertex set of $\mathbf{G}$. Similarly, if $\bar{x}$ is the stem of $\mathbf{H}$ and $\bar{y}$ is the stem of $\mathbf{G}$ then $\phi(\bar{x})$ is a prefix of $\bar{y}$.

Finally, we note that embeddings are transitive:

Observation 2 *If* $\mathbf{K} \overset{\phi}{\hookrightarrow} \mathbf{H}$ *and* $\mathbf{H} \overset{\psi}{\hookrightarrow} \mathbf{G}$ *then* $\mathbf{K} \overset{\psi \circ \phi}{\hookrightarrow} \mathbf{G}$.

The notion of *elimination trees* (known also under the name *treedepth decomposition* and many others) connects tree ordered graphs to linearly ordered graphs.

Definition 1 (*Elimination tree*) Given a connected linearly ordered graph $\mathbb{H}$, the *elimination tree* $\mathrm{ET}(\mathbb{H})$ is defined recursively as follows: Let $x := \min \mathbb{H}$ and let $\mathbb{K}_1, \ldots, \mathbb{K}_s$ be the connected components of $\mathbb{H} - x$. Then $\mathrm{ET}(\mathbb{H})$ has x as its root with the roots of $\mathrm{ET}(\mathbb{K}_1), \ldots, \mathrm{ET}(\mathbb{K}_s)$ as its children.

Definition 2 (*Tree order relaxation*) Given a connected linearly ordered graph $\mathbb{H}$ and its elimination tree $T := \mathrm{ET}(\mathbb{H})$, we define its *tree order relaxation* as the tog $\mathsf{relax}(\mathbb{H}) = (H, \preccurlyeq^T_{\mathsf{anc}})$.

Observe that $\mathsf{relax}(\mathbb{H}) \hookrightarrow\!\!\!\!\rightarrow \mathbb{H}$ and these embeddings have the stem of $\mathsf{relax}(\mathbb{H})$ as fixed points.

Definition 3 (Elimination-ordered graph (etog)) A tog $\mathbf{H}$ for which there exists a linear graph $\mathbb{H}$ such that $\mathsf{relax}(\mathbb{H}) = \mathbf{H}$ is called an *elimination-ordered graph* (etog).

Lemma 1 *Let* $\mathbf{H} = \mathsf{relax}(\mathbb{H})$ *be a tree order relaxation of a connected linear graph* $\mathbb{H}$. *Then for every pair of vertices* $x, y \in V(\mathbf{H})$ *it holds that that* $x \preccurlyeq_{\mathbf{H}} y$ *if and only if there exists an* x–y*-path* P *with* $\min_{\mathbf{H}} P = x$.

Proof Let $T = \mathrm{ET}(\mathbb{H})$ be the elimination tree whose ancestor relationship defines $\preccurlyeq_{\mathbf{H}}$. First assume that $x \preccurlyeq_{\mathbf{H}} y$. By definition, the nodes of the subtree T_x induce a connected subtog of $\mathbb{H}$ and hence in $\mathbf{H}$, thus there exists a path P from x to y in $\mathbf{H}[V(T_x)]$ and hence $\min_{\mathbf{H}} P = x$. In the other direction, assume an x–y-path P with $\min_{\mathbf{H}} P = x$ exists. Since $y \in P$ it follows that $x \preccurlyeq_{\mathbf{H}} y$, as claimed. □

Corollary 1 *Let* $\mathbf{H} = \mathsf{relax}(\mathbb{H})$ *be a tree order relaxation of a connected linear graph* $\mathbb{H}$. *Then for every pair of vertices* $x, y \in V(\mathbf{H})$ *which are incomparable under* $\preccurlyeq_{\mathbf{H}}$, *every path* P *from* x *to* y *satisfies* $\min_{\mathbf{H}} P \notin \{x, y\}$.

2.5 Reachability and Bounded Expansion

For any integer r, we define the set $\mathcal{P}^r(u)$ as the set of all paths with length between 1 and r which have u as one of their endpoints. Similarly, we the set $\mathcal{P}^r(u, v)$ as the set of all u–v-paths of length $\leq r$. With this notation, we can now define the *weak r-neighbors* of a vertex u in a linear graph $\mathbb{G}$ as the set

$$W^r_{\mathbb{G}}(u) = \{\min P \mid P \in \mathcal{P}^r(u)\} \setminus \{u\},$$

that is, $W^r_{\mathbb{G}}(u)$ contains all vertices that are *weakly r-reachable* from u. Notice that by definition every vertex in $W^r_{\mathbb{G}}(u)$ precedes u in $\mathbb{G}$. In words, this set contains those vertices $v \preccurlyeq_{\mathbb{G}} u$ that can be reached from u by using a path of length at most r which does not use any vertices to the left of v.

We also define the *strong r-neighbors* as the set

$$S^r_{\mathbb{G}}(u) = \{v \preccurlyeq_{\mathbb{G}} u \mid \exists P \in \mathcal{P}^r(u, v) \text{ s.t. } u \preccurlyeq_{\mathbb{G}} (P - v)\},$$

that is, $S^r_{\mathbb{G}}(u)$ contains all vertices that are *strongly r-reachable* from u. This set contains those vertices $v \preccurlyeq_{\mathbb{G}} u$ which can be reached by paths of length at most r which, apart from its endpoints, lie entirely to the right of u. For convenience, we define $W^r_{\mathbb{G}}[u] := W^r_{\mathbb{G}}(u) \cup \{u\}$ and $S^r_{\mathbb{G}}[u] := S^r_{\mathbb{G}}(u) \cup \{u\}$. As usual, we omit the subscript $\mathbb{G}$ if clear from the context.

The notions of weak and strong reachability are at the core of the generalized colorings numbers col_r and wcol_r. For a linear graph[5] $\mathbb{G}$, we define them as

$$\mathrm{wcol}_r(\mathbb{G}) := \max_{v \in \mathbb{G}} |W^r_{\mathbb{G}}[v]|,$$
$$\mathrm{col}_r(\mathbb{G}) := \max_{v \in \mathbb{G}} |S^r_{\mathbb{G}}[v]|,$$

which lets us define the generalized coloring numbers for unordered graphs as

$$\mathrm{wcol}_r(G) := \min_{\mathbb{G} \in \Pi(G)} \mathrm{wcol}_r(\mathbb{G}),$$
$$\mathrm{col}_r(G) := \min_{\mathbb{G} \in \Pi(G)} \mathrm{col}_r(\mathbb{G}).$$

Kierstead and Yang [15] showed that the weak r-coloring number is bounded iff the r-coloring number is:

$$\mathrm{col}_r(G) \leq \mathrm{wcol}_r(G) \leq \mathrm{col}_r(G)^r,$$

and Zhu related the above graph measures to classes of bounded expansion [24]. As a result, we can work with the following characterisation of bounded expansion and nowhere dense classes:

[5] These definitions could also be applied to tree-ordered graphs, but this does not bear any benefit here.

Proposition 1 *The following statements about a graph class $\mathcal{G}$ are equivalent:*

1. *$\mathcal{G}$ has* bounded expansion,
2. *there exists a function f such that $\operatorname{col}_r(G) < f(r)$ for all $G \in \mathcal{G}$ and all $r \in \mathbb{N}_0$,*
3. *there exists a function g such that $\operatorname{wcol}_r(G) < g(r)$ for all $G \in \mathcal{G}$ and all $r \in \mathbb{N}_0$.*

In nowhere dense classes these measures might depend on the size of the graph, albeit only sublinearly [13]:

Proposition 2 *The following statements about a graph class $\mathcal{G}$ are equivalent:*

1. *$\mathcal{G}$ is* nowhere dense,
2. *there exists a sequence of functions $(f_r)_{r\in\mathbb{N}_0}$ with $f_r(n) = O(n^{o(1)})$ such that $\operatorname{col}_r(G) < f_r(|G|)$ for all $G \in \mathcal{G}$ and all $r \in \mathbb{N}_0$,*
3. *there exists a sequence of functions $(g_r)_{r\in\mathbb{N}_0}$ with $g_r(n) = O(n^{o(1)})$ such that $\operatorname{wcol}_r(G) < g_r(|G|)$ for all $G \in \mathcal{G}$ and all $r \in \mathbb{N}_0$.*

We are left with the question of computing orderings which provide small values for W^r or S^r. Finding optimal orderings for weakly reachable sets is NP-complete [12] for $r \geq 3$, we therefore have to resort to approximations. To our knowledge, the best current option for theoretical purposes is via *admissibility*, yet another order-based measure: the r-admissibility $\operatorname{adm}_r^{\mathbb{G}}(v)$ of a vertex v in an ordered graph $\mathbb{G}$ is the maximum number of paths of length at most r which a) only intersect in v and b) end in vertices that come before v in $\leq_{\mathbb{G}}$. The admissibility of an ordered graph $\mathbb{G}$ is then

$$\operatorname{adm}_r(\mathbb{G}) = \max_{v\in G} |\operatorname{adm}_r^{\mathbb{G}}(v)|,$$

and it is not too difficult to see that $\operatorname{adm}_r(\mathbb{G}) \leq \operatorname{col}_r(\mathbb{G})$. As with the other generalized coloring numbers, the admissibility of an unordered graph is taken over all its possible orderings:

$$\operatorname{adm}_r(G) = \min_{\mathbb{G}\in\Pi(G)} \operatorname{adm}_r(\mathbb{G}).$$

In the other direction, we have the following results:

Proposition 3 (cf. Dvořák [5]) *For any linear ordering $\mathbb{G}$ of G and $r \in \mathbb{N}$ it holds that*

$$\operatorname{col}_r(\mathbb{G}) \leq \operatorname{adm}_r(\mathbb{G})(\operatorname{adm}_r(\mathbb{G}) - 1)^{r-1} + 1.$$

Proposition 4 (cf. Dvořák [6]) *For any linear ordering $\mathbb{G}$ of G and $r \in \mathbb{N}$ it holds that*

$$\operatorname{wcol}_r(\mathbb{G}) \leq \left(r^2 \operatorname{adm}_r(\mathbb{G})\right)^r.$$

Importantly, a linear-time algorithm to compute the admissiblity exists[6].

[6] The algorithm relies on heavy machinery and is in its current formulation probably not practical. See Sect. 5 below for a discussion of this issue.

Proposition 5 (cf. Dvořák [5]) *Let $\mathcal{G}$ be a class with bounded expansion and $r \in \mathbb{N}$. There exists a linear-time algorithm that for each $G \in \mathcal{G}$ computes an ordering $\mathbb{G}$ with* $\mathrm{adm}_r(\mathbb{G}) = \mathrm{adm}_r(G)$.

As a corollary to these three proposition, we can compute an ordering $\mathbb{G}$ of G in linear time with

$$\mathrm{col}_r(\mathbb{G}) \leq \mathrm{col}_r(G)(\mathrm{col}_r(G) - 1)^{r-1} + 1 = O(\mathrm{col}_r(G)^r)$$

and

$$\mathrm{wcol}_r(\mathbb{G}) \leq \left(r^2 \,\mathrm{wcol}_r(G)\right)^r.$$

2.6 Conventions

In the remainder, we fix a linear graph $\mathbb{G}$, the *host graph*, and a *pattern graph* H. Our goal is to count how often H appears as an induced subgraph in the underlying graph G of $\mathbb{G}$. For ease of presentation, we will assume that H is connected and discuss later how the algorithms can be modified for disconnected patterns.

3 Pattern Decomposition

We will be counting the pattern by considering the possible orderings in which it may appear in the host graph. However, it turns out that some of these orderings need to be treated as a unit with our approach, namely those orderings that result in the same pattern relaxation. In that sense, we count the number of embeddings only for members of the following set:

Definition 4 (*Pattern relaxation*) For the pattern graph H we define its *pattern relaxations* as the set

$$\mathcal{H} := \{\mathsf{relax}((H, \leq_\pi)) \mid \pi \in \pi(V(H))\}.$$

Each pattern relaxation will be decomposed further until we arrive at an object that is easily countable. To that end, we define the following:

Definition 5 (*Pieces, linear pieces*) Given a pattern relaxation $\mathbf{H} \in \mathcal{H}$ and a subset of its leaves $S \subseteq \mathrm{leaves}(\mathbf{H})$, the *piece* induced by S is the induced subtog

$$\mathbf{H}\Big[\bigcup_{x \in S} \mathrm{rpath}(x)\Big].$$

If $|S| = 1$, the resulting piece is a linear graph and we refer to it as a *linear piece*.

With that, we define the decomposition of a pattern relaxation via piece sums (see Fig. 1 for examples):

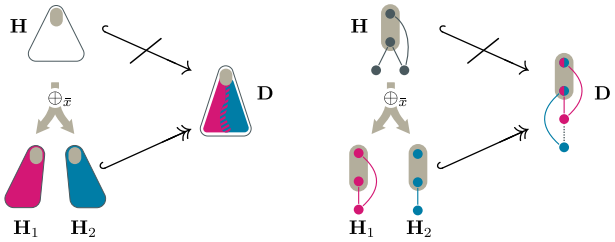


Fig. 1 The tog $\mathbf{H}$ is decomposed into pieces $\mathbf{H}_1$ and $\mathbf{H}_2$ along its stem $\bar{x}$. $\mathbf{D}$ is a defect of $\mathbf{H}$ as $\mathbf{H} \not\hookrightarrow \mathbf{D}$ but there exists a mapping ϕ such that $\mathbf{H}_1 \overset{\phi}{\hookrightarrow} \mathbf{D}$ and $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$. On the right the same situation with a concrete example where $\mathbf{H}$ is an etog of P_4

Definition 6 (*Piece sum*) Let $\mathbf{H}$ be a tog with stem $\bar{x}$. We write $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ to denote that $\mathbf{H}_1$ and $\mathbf{H}_2$ are pieces of $\mathbf{H}$ with the properties that

1. leaves($\mathbf{H}_1$) and leaves($\mathbf{H}_2$) are both non-empty and partition leaves($\mathbf{H}$),
2. and $\mathbf{H}_1 \cap \mathbf{H}_2 = \bar{x}$.

Note that pieces of a connected tog are not necessarily connected. However, in the case of *etogs*, some connectivity is maintained, namely between non-stem vertices:

Lemma 2 *Let* $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ *be an* etog *and let* $u \in \mathbf{H}_1 - \bar{x}$. *Then any vertex* v *with* $u \preccurlyeq_{\mathbf{H}} v$ *is contained in* $\mathbf{H}_1 - \bar{x}$. *Moreover, there exists a* u–v*-path* $P \subseteq \mathbf{H}_1 - \bar{x}$ *with* $\min_{\mathbf{H}} P = \min_{\mathbf{H}_1} P = u$.

Proof Since $\mathbf{H}$ is an etog, there exists an elimination tree T whose ancestor relationship defines $\preccurlyeq_{\mathbf{H}}$. Consider the subtree T_u rooted at u, because $u \preccurlyeq_{\mathbf{H}} v$ we have that $v \in T_u$. Recall that by the definition of elimination trees $H[T_u]$ is connected, therefore $H[T_u]$ contains a u–v-path $P \subseteq T_u$.

Because $u \in \mathbf{H}_1 - \bar{x}$ it follows that $T_u \subseteq \mathbf{H}_1 - \bar{x}$, thus the same is true for P. It immediately follows that $\min_{\mathbf{H}_1} P = u$. □

We now show that linear pieces can be enumerated or counted in linear time given a suitable vertex ordering of the host graph with constant-sized weak/strong r-neighborhoods.

3.1 Counting (Relevant) Linear Pieces

We first prove that all relevant linear pieces (those that can be completed to the full pattern) are completely contained in weakly reachable sets and therefore can be counted easily in time $O(\mathrm{wcol}_{|\mathbf{H}|}(\mathbb{G})^{|\mathbf{H}|-1} \cdot |\mathbb{G}|)$, see Sect. 4 for details.

Lemma 3 *Let* $\mathbb{K}$ *be a linear piece for some pattern relaxation* $\mathbf{H} \in \mathcal{H}$ *and let* $z = \max(\mathbb{K})$. *Then for every* $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbb{G}$ *it holds that* $\phi(\mathbb{K})$ *is contained in* $W_{\mathbb{G}}^{|\mathbf{H}|}[\phi(z)]$.

Proof Let ϕ be such an embedding and fix any $x \in \mathbb{K}$. We need to show that $\phi(x) \in W_{\mathbb{G}}^{|\mathbf{H}|}[\phi(z)]$. Since we assumed that H is connected, so is $\mathbf{H}$. Then by Lemma 1, there

exists a path P path from x to z in $\mathbf{H}$ with $\min_{\mathbf{H}} P = x$. Since ϕ is an embedding, by Observation 1 it holds that

$$\min_{\mathbb{G}} \phi(P) = \phi(\min_{\mathbf{H}} P) = \phi(x).$$

We conclude that $\phi(x) \in W_{\mathbb{G}}^{|P|}[\phi(z)] \subseteq W_{\mathbb{G}}^{|\mathbf{H}|}[\phi(z)]$. □

The above does not hold if we replace weak reachability by strong reachability, however, the following statement already suffices to build the strong-reachability variant of our algorithm:

Lemma 4 *Let $\mathbb{K}$ be a linear piece for some pattern relaxation $\mathbf{H} \in \mathcal{H}$. Let $z = \max(\mathbb{K})$ and let $x \prec_{\mathbb{K}} z$ be an arbitrary vertex of $\mathbb{K}$. There exists a vertex $y \in \mathbb{K}$, $x \prec_{\mathbb{K}} y \preceq_{\mathbb{K}} z$ such that for every embedding $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbb{G}$ it holds that $\phi(x) \in S_{\mathbb{G}}^{|H|}[\phi(y)]$.*

Proof Let ϕ be such an embedding. Again, since $\mathbf{H}$ is connected there exists a path P from x to z in $\mathbf{H}$ with $\min_{\mathbf{H}} P = x$. Let $y = \min_{\mathbf{H}}((P - x) \cap V(\mathbb{K}))$ be the smallest vertex of $\mathbb{K}$ which lies on P; since z lies in this intersection this minimum must exist. Let P' be the portion of P which goes from x to y.

Claim $y = \min_{\mathbf{H}}(P' - x)$.

Proof Assume towards a contradiction that $y' = \min_{\mathbf{H}}(P' - x)$ with $y' \neq y$. Note that by our choice of P it holds that $y' \prec_{\mathbf{H}} z$.

First consider the case that $y' \prec_{\mathbf{H}} y$. Hence y' must lie somewhere on the path from x to y in $T(\mathbf{H})$. But then y' is contained in the piece $\mathbb{K}$ and hence $(P - x) \cap V(\mathbb{K})$, contradicting our choice of y.

Otherwise, y' and y are incomparable under $\preccurlyeq_{\mathbf{H}}$ and in particular y' cannot lie anywhere on $\mathrm{rpath}_{T(\mathbf{H})}(y)$ or anywhere below y in $T(\mathbf{H})$. Since $\preccurlyeq_{\mathbf{H}}$ guards $E(H)$ the path P can only go from y' to y by intersecting $\mathrm{rpath}_{T(\mathbf{H})}(y)$ in some vertex y''. But then $y'' \in (P - x) \cap V(\mathbb{K})$, contradicting our choice of y. □

Finally, we apply Observation 1 and find that

$$\min_{\mathbb{G}} \phi(P' - x) = \phi(\min_{\mathbf{H}}(P' - x)) = \phi(y)$$

from which we conclude that indeed $\phi(x) \in S_{\mathbb{G}}^{|P|}[\phi(y)] \subseteq S_{\mathbb{G}}^{|H|}[\phi(y)]$. □

We call such a vertex y a *hint* and introduce the following notation to speak about it more succinctly:

Definition 7 (hint). Let $\mathbb{K}$ be a linear piece of $\mathbf{H} \in \mathcal{H}$ with vertices $x_1, \ldots, x_p$. For every index $i \in [p]$ we define the function $\mathsf{hint}_{\mathbb{K}}^{\mathbf{H}}(i)$ to be the largest index $j > i$ such that for every embedding $\mathbf{H} \overset{\phi}{\hookrightarrow} \mathbb{G}$ it holds that $\phi(x_i) \in S_{\mathbb{G}}^{|H|}[\phi(x_j)]$.

To use these hints algorithmically, we proceed as follows. Given a linear piece $\mathbb{K}$ with vertices $x_1, \ldots, x_p$, we first match the last vertex x_p to a vertex $\hat{x}_p$ in $\mathbf{G}$. By Lemma 4,

we can now restrict our search for $\hat{x}_{p-1}$ to vertices in the set $S^p_{\mathbb{G}}[\hat{x}_p]$. For any match of x_{p-1} to $\hat{x}_{p-1}$ in $\mathbf{G}$, we then can attempt to find a match for x_{p-2} by searching the set $S^p_{\mathbb{G}}[\hat{x}_j]$ where $j = \mathsf{hint}(p-2) \in \{p-1, p\}$. This algorithm is described in more detail in the last section.

3.2 Combining Counts

In order to succinctly describe our approach, we need to introduce the following notation for counting embeddings of a pattern graph $\mathbf{H}$ into a host graph $\mathbf{G}$ where we already fix the embedding of a prefix of $\mathbf{H}$'s stem vertices.

Definition 8 (*(Strict) embedding count*) For togs $\mathbf{H}, \mathbf{G}$ with $\bar{x}$ a stem prefix of $\mathbf{H}$ and $\bar{y} \subseteq \mathbf{G}$ an ordered vertex set with $|\bar{x}| = |\bar{y}|$, we define

$$\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbf{G}) := \left|\{\phi \mid \mathbf{H} \overset{\phi}{\hookrightarrow} \mathbf{G} \text{ and } \phi(\bar{x}) = \bar{y}\}\right|.$$

The central idea is now that in order to count $\mathbf{H}$, we instead count the occurrences of two pieces $\mathbf{H}_1 \oplus_{\mathsf{stem}\,\mathbf{H}} \mathbf{H}_2$ and compute $\#_{\mathsf{stem}\,\mathbf{H} \mapsto \bar{y}}(\mathbf{H}, \mathbb{G})$ by taking the product $\#_{\mathsf{stem}\,H \mapsto \bar{y}}(\mathbf{H}_1) \cdot \#_{\mathsf{stem}\,H \to \bar{y}}(\mathbf{H}_2)$. Of course, the latter quantity over-counts the former, as we will discuss below. First, let us introduce the following notation for this 'estimate' embeddings count:

Definition 9 (*Relaxed embedding*) For togs $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2, \mathbf{G}$ with $\bar{x}$ a stem prefix of $\mathbf{H}$ we write $(\mathbf{H}_1, \mathbf{H}_2) \overset{\phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{G}$ to denote that the mapping $\phi \in V(\mathbf{G})^{V(\mathbf{H})}$ has the following properties:

- $\mathbf{H}_1 \overset{\phi}{\hookrightarrow} \mathbf{G}$ and $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{G}$,
- $\phi(\mathbf{H}) = V(\mathbf{G})$.

Definition 10 (*Relaxed embedding count*) For togs $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2, \mathbf{G}$ with $\bar{x}$ a stem prefix of $\mathbf{H}$ and $\bar{y} \subseteq \mathbf{G}$ an ordered vertex set with $|\bar{x}| = |\bar{y}|$ we write $\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbf{G} \mid \mathbf{H}_1, \mathbf{H}_2)$ for the number of embeddings $(\mathbf{H}_1, \mathbf{H}_2) \overset{\phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{G}$ with $\phi(\bar{x}) = \bar{y}$.

Now, how does a mapping which embeds $\mathbf{H}_1$ and $\mathbf{H}_2$ fail to embed $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$? We either must have that the images of $\mathbf{H}_1$ and $\mathbf{H}_2$ intersect or that there exists an edge between their images which does not belong to $\mathbf{H}$. We will call such pair of embeddings a *defect*:

Definition 11 (*Defect*) Let $\mathbf{H} \in \mathcal{H}$ be a pattern relaxation and let $\mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2 = \mathbf{H}$. A *defect* of $\mathbf{H}_1, \mathbf{H}_2$ is any etog $\mathbf{D}$ that satisfies the following properties:

1. $\mathbf{H} \not\hookrightarrow \mathbf{D}$,
2. $\mathbf{H}_1 \overset{id}{\hookrightarrow} \mathbf{D}$,

3. $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$ where ϕ is the identity on the set $(V(\mathbf{H}_2) \setminus V(\mathbf{H}_1)) \cup \bar{x}$,
4. and $V(\mathbf{D}) = V(\mathbf{H}_1) \cup \phi(\mathbf{H}_2)$.

We will write $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$ to denote the set of all defects for the pair $\mathbf{H}_1, \mathbf{H}_2$.

Note that several of the above properties are for convenience only: we insist that $\mathbf{H}_1$ is a subgraph of $\mathbf{D}$ to avoid handling yet another embedding and we make ϕ preserve all vertices that it possibly can for the same reason. Importantly, all the togs $\mathbf{H}$, $\mathbf{H}_1$, $\mathbf{H}_2$, and $\mathbf{D}$ share the ordered set $\bar{x}$ as a stem prefix.

At this point we should point out that it is not a priori clear that it is enough to consider defects that are etogs themselves, it could very well be the case that defects are arbitrary tree-ordered or just 'ordered' graphs. Note that what we *really* want to count are linear subgraphs $\mathbb{D} \subseteq \mathbb{G}$ into which $\mathbf{H}_1$ and $\mathbf{H}_2$ embed, but $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ does not (as these are precisely the cases that we over-count in the product $\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}_1) \cdot \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}_2)$, for some prefix $\bar{y}$ of $\mathbb{D}$). The next lemma shows that instead of trying to find these linear subgraphs, we can instead recourse to counting their relaxations, thus circling back to etogs:

Lemma 5 *Let $\mathbf{H}$ be a connected etog with pieces $\mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2 = \mathbf{H}$. Let $\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$. Then for every linear graph $\mathbb{D}$ with $\mathsf{relax}(\mathbb{D}) = \mathbf{D}$ it holds that*

$$\#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2) = \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbb{D} \mid \mathbf{H}_1, \mathbf{H}_2)$$

Proof Fix a linear graph $\mathbb{D}$ with $\mathbf{D} = \mathsf{relax}(\mathbb{D})$ in the following. Let $\xi_1, \ldots, \xi_p$ be all strict embeddings of $\mathbf{D}$ into $\mathbb{D}$ with fixed points $\bar{x}$, that is, $\mathbf{D} \overset{\xi_i}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$ and $\xi_i(\bar{x}) = \bar{x}$ for all $i \in [p]$. Recall that the mappings ξ_i are in particular automorphisms of the underlying graph D and therefore the inverse mappings ξ_i^{-1} exist.

Claim *Let $(\mathbf{H}_1, \mathbf{H}_2) \overset{\phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{D}$ with $\phi(\bar{x}) = \bar{x}$. Then $(\mathbf{H}_1, \mathbf{H}_2) \overset{\xi_i \circ \phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$ and $(\xi_i \circ \phi)(\bar{x}) = \bar{x}$ for each $i \in [p]$.*

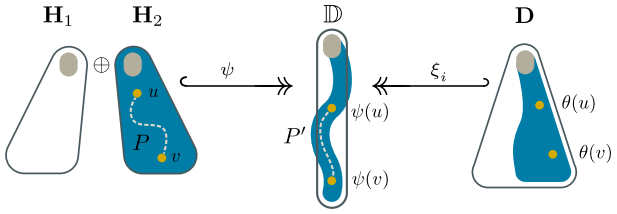
Proof For each $\mathbf{H}_j$, $j \in \{1, 2\}$, and $i \in [p]$ we have that $\mathbf{H}_j \overset{\phi}{\hookrightarrow} \mathbf{D} \overset{\xi_i}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$, therefore by transitivity $\mathbf{H}_j \overset{\xi_i \circ \phi}{\hookrightarrow} \mathbb{D}$. Since $\phi(\mathbf{H}) = V(\mathbf{D})$ and each ξ_i is an automorphism of the underlying graph D, it follows that $(\xi \circ \phi)(\mathbf{H}) = V(\mathbf{D}) = V(\mathbb{D})$. We conclude that indeed $(\mathbf{H}_1, \mathbf{H}_2) \overset{\xi_i \circ \phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$. Finally, since both ϕ and each ξ_i have $\bar{x}$ as fixed points, $(\xi \circ \phi)(\bar{x}) = \bar{x}$. □

Claim *Let $(\mathbf{H}_1, \mathbf{H}_2) \overset{\psi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$ with $\psi(\bar{x}) = \bar{x}$. Then $(\mathbf{H}_1, \mathbf{H}_2) \overset{\xi_i^{-1} \circ \psi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{D}$ and $(\xi_i^{-1} \circ \psi)(\bar{x}) = \bar{x}$ for each $i \in [p]$.*

Proof Fix $\xi_i, i \in [p]$ and let $\theta := \xi_i^{-1} \circ \psi$. We first show that $\mathbf{H}_j \overset{\theta}{\hookrightarrow} \mathbf{D}$ for $j \in \{1, 2\}$. Clearly θ is an induced subgraph isomorphism from H_j to D with fixed points $\bar{x}$, it remains to show that the ordering $\preccurlyeq_{\mathbf{H}_j}$ is properly embedded into $\preccurlyeq_{\mathbf{D}}$.

Consider a pair $u \preccurlyeq_{\mathbf{H}_j} v$. Then $\psi(u) \leq_{\mathbb{D}} \psi(v)$ and since $\mathbf{D} \overset{\xi_i}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$, we have two possibilities for the pre-images of $\psi(u)$ and $\psi(v)$ under ξ_i: either $(\xi_i^{-1} \circ \psi)(u) = \theta(u)$

and $(\xi_i^{-1} \circ \psi)(v) = \theta(v)$ satisfy $\theta(u) \preccurlyeq_{\mathbf{D}} \theta(v)$, in which case we are done, or the two are incomparable. We now argue that the latter case is impossible.



First, note that if $u \in \bar{x}$ then necessarily $\theta(u) \preccurlyeq_{\mathbf{D}} \theta(v)$: if $v \in \bar{x}$ then their order is maintained because $\theta(\bar{x}) = \bar{x}$, otherwise the claim follows because $\bar{x}$ is a stem-prefix of $\mathbf{D}$ and therefore $\bar{x} \preccurlyeq_{\mathbf{D}} \mathbf{D} - \bar{x}$.

Therefore we may assume that $u \in \mathbf{H}_j - \bar{x}$ and we apply Lemma 2 to obtain a u–v-path $P \subseteq H_j - \bar{x}$ with $\min_{\mathbf{H}_j} P = u$. Since $\mathbf{H}_j \overset{\psi}{\hookrightarrow} \mathbb{D}$, there exists a $\psi(u)$–$\psi(v)$-path $P' \subseteq \psi(P)$ in $\mathbb{D}$ with $\min_{\mathbb{D}} P' = \psi(\min_{\mathbf{H}_j} P) = \psi(u)$. But then by Corollary 1, the pre-images of $\psi(u)$ and $\psi(v)$ under ξ_i (namely $\theta(u)$ and $\theta(v)$) cannot be incomparable (Corollary 1 applies after a trivial relabelling of graphs to $\mathbf{D}$ and $\mathbb{D}$).

We conclude that indeed $\theta(u) \preccurlyeq_{\mathbf{D}} \theta(v)$ and therefore that $H_j \overset{\theta}{\hookrightarrow} \mathbf{D}$ for $j \in \{1, 2\}$, as claimed. □

Let (L, R, E) be a bipartite graph where the side $L := \{\phi \mid (\mathbf{H}_1, \mathbf{H}_2) \overset{\phi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbf{D}$ and $\phi(\bar{x}) = \bar{x}\}$ contains all relevant mappings from $\mathbf{H}$ into $\mathbf{D}$ and similarly $R := \{\psi \mid (\mathbf{H}_1, \mathbf{H}_2) \overset{\psi}{\hookrightarrow\!\!\!\!\rightarrow} \mathbb{D}$ and $\psi(\bar{x}) = \bar{x}\}$ those from $\mathbf{H}$ into $\mathbb{D}$. We draw an edge between $\phi \in L$ and $\psi \in R$ if $\xi_i \circ \phi = \psi$ for some $i \in [p]$. This is of course precisely true iff $\phi = \xi_i^{-1} \circ \psi$. Therefore, by the above observations, we find that vertices on both sides have degree exactly p. The lemma now follows by observing that in a regular bipartite graph both sides must have the same size. □

We are now ready to prove the main technical lemma of this paper, the recurrence that will allow us to compute $\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{G})$, i.e. the number of embeddings from $\mathbf{H}$ into $\mathbb{G}$ which map the stem prefix $\bar{x}$ of $\mathbf{H}$ onto the ordered subset $\bar{y}$ of $\mathbb{G}$. Note that in order to compute the number of *induced subgraphs*, we simply have to divide this value by $\#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{H})$, the number of automorphisms of $\mathbf{H}$ with fixed points $\bar{x}$.

Lemma 6 *Let $\mathbf{H} \in \mathcal{H}$ be a (non-linear) pattern relaxation and let $\mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2 = \mathbf{H}$. Fix an ordered vertex set $\bar{y} \in \mathbb{G}$ such that $\mathbf{H}[\bar{x}] \simeq \mathbb{G}[\bar{y}]$. Then*

$$\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{G}) = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}_1, \mathbb{G}) \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}_2, \mathbb{G}) - \sum_{\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)} c_{\mathbf{D}} \#_{\bar{x} \mapsto \bar{y}}(\mathbf{D}, \mathbb{G})$$

with coefficients $c_{\mathbf{D}} = \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2) / \#_{\bar{x} \mapsto \bar{x}}(\mathbf{D}, \mathbf{D})$

Proof In order to prove this equation we will rewrite it in terms of individual linear subtogs $\mathbb{K} \subseteq \mathbb{G}$ and consider how they contribute to the different terms. To that end, define $\Phi(\mathbb{K})$ to be all pairs (ϕ_1, ϕ_2) with $\phi_1(\bar{x}) = \phi_2(\bar{x}) = \bar{y}$ such that

(a) $\mathbf{H}_1 \overset{\phi_1}{\hookrightarrow} \mathbb{K}$ and $\mathbf{H}_2 \overset{\phi_2}{\hookrightarrow} \mathbb{K}$; and

(b) $V(\mathbb{K}) = V(\phi_1(\mathbf{H}_1)) \cup V(\phi_2(\mathbf{H}_2))$.

That is, $\Phi(\mathbb{K})$ contains all pairs of embeddings that minimally embed the graphs $\mathbf{H}_1$ and $\mathbf{H}_2$ into $\mathbb{K}$ while also mapping $\bar{x}$ onto $\bar{y}$.

Claim *Let* $\mathbb{K} \subseteq \mathbb{G}$ *such that* $\mathbf{H} \xhookrightarrow{\bar{x} \mapsto \bar{y}} \mathbb{K}$. *Then* $|\Phi(\mathbb{K})| = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{K})$.

Proof First note that since $\mathbf{H} \hookrightarrow \mathbb{K}$, it follows that $H \simeq K$. Fix an embedding-pair $(\phi_1, \phi_2) \in \Phi(\mathbb{K})$ and let $V_1 := V(\phi_1(\mathbf{H}_1))$ and $V_2 := V(\phi_2(\mathbf{H}_2))$ By definition, $V_1 \cup V_2 = V(\mathbb{K})$ and since $|\mathbf{H}| = |\mathbb{K}|$ it follows that V_1 and V_2 partition $V(\mathbb{K})$.

As the embeddings of $\mathbf{H}_1$ and $\mathbf{H}_2$ are vertex-disjoint and since $\|\mathbb{K}\| = \|\mathbf{H}\|$, it follows that there cannot exist an edge between $V_1 \setminus \bar{y}$ and $V_2 \setminus \bar{y}$. But then $\mathbf{H} \xhookrightarrow{\xi} \mathbb{K}$ with $\xi(u) = \phi_1(u)$ for $u \in \mathbf{H}_1$ and $\xi(u) = \phi_2(u)$ for $u \in \mathbf{H}_1 - \bar{x}$.

In the other direction, any embedding $\mathbf{H} \xhookrightarrow{\xi} \mathbb{K}$ with $\xi(\bar{x}) = \bar{y}$ can of course be trivially decomposed into an embedding-pair $(\phi_1, \phi_2) \in \Phi(\mathbb{K})$. Accordingly, we have a one-to-one correspondence and we conclude that $|\Phi(\mathbb{K})| = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{K})$, as claimed. □

We can now use Iverson bracket notation to rewrite $\#_{\bar{x} \to \bar{y}}(\mathbf{H}, \mathbb{G})$ and obtain

$$\begin{aligned}\underset{\bar{x} \mapsto \bar{y}}{\#}(\mathbf{H}, \mathbb{G}) &= \sum_{\mathbb{K} \subseteq \mathbb{G}} \underset{\bar{x} \mapsto \bar{y}}{\#}(\mathbf{H}, \mathbb{K}) = \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \to \bar{y}} \mathbb{K}]\!] |\Phi(\mathbb{K})| \\ &= \underbrace{\sum_{\mathbb{K} \subseteq \mathbb{G}} |\Phi(\mathbb{K})|}_{\clubsuit} - \underbrace{\sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}]\!] |\Phi(\mathbb{K})|}_{\spadesuit}\end{aligned}$$

which holds because $\Phi(\mathbb{K}) = \emptyset$ whenever $\mathbb{K}$ does not have $\bar{y}$ as a stem prefix. We will now consider the two sums (♣) and (♠) individually.

Claim

$$\underset{\bar{x} \mapsto \bar{y}}{\#}(\mathbf{H}_1, \mathbb{G}) \underset{\bar{x} \mapsto \bar{y}}{\#}(\mathbf{H}_2, \mathbb{G}) = \sum_{\mathbb{K} \subseteq \mathbb{G}} |\Phi(\mathbb{K})|.$$

Proof Fix an embedding-pair (ϕ_1, ϕ_2) with $\phi_1(\bar{x}) = \phi_2(\bar{x}) = \bar{y}$ as well as $\mathbf{H}_1 \xhookrightarrow{\phi_1} \mathbb{G}$ and $\mathbf{H}_2 \xhookrightarrow{\phi_2} \mathbb{G}$. Such a pair contributes exactly one to the left-hand side of the equation and we therefore have to show that there is exactly one $\mathbb{K} \subseteq \mathbb{G}$ with $(\phi_1, \phi_2) \in \Phi(\mathbb{K})$. Concretely, we claim that this subtog is $\mathbb{K} = \mathbb{G}[V(\phi_1(\mathbf{H}_1) \cup V(\phi_2(\mathbf{H}_2)))]$. Clearly $(\phi_1, \phi_2) \in \Phi(\mathbb{K})$, to see that this is the only subtog with this property, simply note that $(\phi_1, \phi_2) \in \Phi(\mathbb{K}')$ implies that $V(\mathbb{K}') = V(\phi_1(\mathbf{H}_1)) \cup V(\phi_2(\mathbf{H}_2)) = V(\mathbb{K})$ and therefore $\mathbb{K}' = \mathbb{K}$. □

The following claim will help us to rewrite the sum (♠):

Claim *Let* $\mathbb{K} \subseteq \mathbb{G}$ *be a linear subtog with* $\mathbf{H} \xhookrightarrow{\bar{x} \not\mapsto \bar{y}} \mathbb{K}$ *and* $\Phi(\mathbb{K}) \neq \emptyset$. *Then there exists a unique defect* $\mathbf{D} \in \mathcal{D}(H_1, H_2)$ *with* $\mathbf{D} \simeq \mathit{relax}(\mathbb{K})$ *and* $|\Phi(\mathbb{K})| = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2)$.

Proof Since $\Phi(\mathbb{K})$ is non-empty there exists at least one embedding-pair (ϕ_1, ϕ_2) with $\mathbf{H}_1 \xhookrightarrow{\phi_1} \mathbb{K}$, $\mathbf{H}_2 \xhookrightarrow{\phi_2} \mathbb{K}$ and $V(\mathbb{K}) = V(\phi_1(\mathbf{H}_1)) \cup V(\phi_2(\mathbf{H}_2))$. Define $\xi : V(\mathbf{H}) \to V(\mathbb{K})$ to be $\xi(u) = \phi_1(u)$ for $u \in \mathbf{H}_1$ and $\xi(u) = \phi_2(u)$ for $u \in \mathbf{H}_2 \setminus \mathbf{H}_1$. Then ξ fulfils all the properties as in Definition 10, meaning that

$$|\Phi(\mathbb{K})| = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{K} \mid \mathbf{H}_1, \mathbf{H}_2) = \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbf{K} \mid \mathbf{H}_1, \mathbf{H}_2),$$

where we applied Lemma 5 with $\mathbf{K} := \mathsf{relax}(\mathbb{K})$ in the second step.

Now note that the etog $\mathbf{K}$ satisfies $\mathbf{H} \not\hookrightarrow \mathbf{K}$ and $V(\mathbf{K}) = V(\phi_1(\mathbf{H}_1)) \cup V(\phi_2(\mathbf{H}_2))$. Therefore $\mathbf{K}$ is, up to relabelling, a defect of $\mathbf{H}_1, \mathbf{H}_2$. Concretely, there exists a unique $\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$ such that $\mathbf{D} \simeq \mathbf{K}$ and accordingly

$$\#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbf{K} \mid \mathbf{H}_1, \mathbf{H}_2) = \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2)$$

from which the claim follows. □

With this claim, we can now rewrite the sum (♠) as follows:

$$\begin{aligned}
&\sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}]\!] |\Phi(\mathbb{K})| && \text{(Lemma 5)} \\
&= \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}]\!] \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathbb{K} \mid \mathbf{H}_1, \mathbf{H}_2). \\
&= \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}]\!] \#_{\bar{x} \mapsto \bar{y}}(\mathbf{H}, \mathsf{relax}(\mathbb{K}) \mid \mathbf{H}_1, \mathbf{H}_2) \\
&= \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}]\!] \sum_{\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)} [\![\mathbf{D} \simeq \mathsf{relax}(\mathbb{K})]\!] \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2)
\end{aligned}$$

Since we are now summing only over subtogs whose relaxation is isomorphic to a defect $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$, we can drop the condition that $\mathbf{H} \xhookrightarrow{\bar{x} \not\to \bar{y}} \mathbb{K}$:

$$\begin{aligned}
♠ &= \sum_{\mathbb{K} \subseteq \mathbb{G}} \sum_{\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)} [\![\mathbf{D} \simeq \mathsf{relax}(\mathbb{K})]\!] \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2) \\
&= \sum_{\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)} \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{D} \simeq \mathsf{relax}(\mathbb{K})]\!] \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2) \\
&= \sum_{\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)} \#_{\bar{x} \mapsto \bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2) \sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{D} \simeq \mathsf{relax}(\mathbb{K})]\!]
\end{aligned}$$

The sum $\sum_{\mathbb{K} \subseteq \mathbb{G}} [\![\mathbf{D} \simeq \mathsf{relax}(\mathbb{K})]\!]$ counts how many subtogs of $\mathbb{G}$ are isomorphic to $\mathbf{D}$. This is the same as counting how often $\mathbf{D}$ embeds into $\mathbb{G}$ divided by the number of automorphisms $\mathbf{D}$ has. We finally arrive at

$$\spadesuit = \sum_{\mathbf{D}\in\mathcal{D}(\mathbf{H}_1,\mathbf{H}_2)} \#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{D}\mid\mathbf{H}_1,\mathbf{H}_2)\frac{\#_{\bar{x}\mapsto\bar{y}}(\mathbf{D},\mathbb{G})}{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{D},\mathbf{D})}.$$

With these alternative computations of the sums (♣) and (♠) we finally arrive at the claimed equality

$$\#_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G}) = \#_{\bar{x}\mapsto\bar{y}}(\mathbf{H}_1,\mathbb{G})\,\#_{\bar{x}\mapsto\bar{y}}(\mathbf{H}_2,\mathbb{G}) - \sum_{\mathbf{D}\in\mathcal{D}(\mathbf{H}_1,\mathbf{H}_2)} \frac{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{D}\mid\mathbf{H}_1,\mathbf{H}_2)}{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{D},\mathbf{D})}\,\#_{\bar{x}\mapsto\bar{y}}(\mathbf{D},\mathbb{G}).$$

□

Note

For practical purposes, it is preferable to compute embedding-counts which exclude automorphisms. Define $\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G}) := \#_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G})/\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{H})$ to be this automorphism-corrected count, then the equation in Lemma 6 becomes

$$\begin{aligned}\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G}) =& \frac{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H}_1,\mathbf{H}_1)\,\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H}_2,\mathbf{H}_2)}{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{H})}\,\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{H}_1,\mathbb{G})\,\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{H}_2,\mathbb{G})\\ &- \sum_{\mathbf{D}\in\mathcal{D}(\mathbf{H}_1,\mathbf{H}_2)} \frac{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{D}\mid\mathbf{H}_1,\mathbf{H}_2)}{\#_{\bar{x}\mapsto\bar{x}}(\mathbf{H},\mathbf{H})}\,\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{D},\mathbb{G}).\end{aligned}$$

The above form is better suited for implementation as the numbers stay smaller but for the mathematical presentation the form in Lemma 6 is simpler.

We next prove that the recurrence implied by the equation in Lemma 6 is finite to prove that the decomposition algorithm does indeed terminate. Here the recurrence terminates if the first argument is a linear graph as these cannot be decomposed further and, as described in the previous sections, these graphs can be counted in linear time. We are not interested in precise bounds here, instead we provide relevant measurements for small graphs of interest in Table 1 in the last section.

Lemma 7 *The recurrence for* $\#_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G})$ *as stated in Lemma 6 has depth at most* $|\mathbf{H}|^2$.

Proof Let $h := |\mathbf{H}|$. We argue that the measure

$$f(\mathbf{G}) = (h - |\,\mathsf{stem}(\mathbf{G})|)h + \kappa(\mathbf{G} - \mathsf{stem}(\mathbf{G})),$$

where κ denotes the number of connected components, strictly decreases for all graphs involved in the right hand side of the recurrence. That is, we show that $f(\mathbf{K}) < f(\mathbf{H})$ for all togs $\mathbf{K} \in \{\mathbf{H}_1,\mathbf{H}_2\} \cup \mathcal{D}(\mathbf{H}_1,\mathbf{H}_2)$. All these graphs have at most h vertices, therefore $\kappa(\mathbf{K} - \mathsf{stem}(\mathbf{K})) \le h$. Note therefore that the measure already decreases if $|\,\mathsf{stem}(\mathbf{K})| > |\,\mathsf{stem}(\mathbf{H})|$, independent of the value of $\kappa(\mathbf{K} - \mathsf{stem}(\mathbf{K}))$.

First consider $\mathbf{H}_1, \mathbf{H}_2$. Since $\mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2 = \mathbf{H}$, we have that $\kappa(\mathbf{H}_i - \mathsf{stem}(\mathbf{H})) < \kappa(\mathbf{H} - \mathsf{stem}(\mathbf{H}))$ for $i \in \{1,2\}$. Thus for $\mathsf{stem}(\mathbf{H}_i) = \mathsf{stem}(\mathbf{H})$ we have $f(\mathbf{H}_i) < f(\mathbf{H})$, as observed above the same holds true when $\mathsf{stem}(\mathbf{H}_i) > \mathsf{stem}(\mathbf{H})$.

Thus we are left to prove that the measure decreases for all $\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$. Let ϕ be the embedding $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$ with fixed points $\bar{x}$ (recall that $\mathbf{H}_1$ is simply a subtog of $\mathbf{D}$). Since $\mathbf{H} \not\hookrightarrow \mathbf{D}$, we conclude that $\mathbf{H}_1 - \bar{x}$ and $\phi(\mathbf{H}_2) - \bar{x}$ must either be connected by an edge or share a vertex in $\mathbf{D}$. In either case, $\kappa(\mathbf{H} - \bar{x}) < \kappa(\mathbf{D} - \bar{x})$. Accordingly, the measure decreases if $\bar{x} = \mathsf{stem}(\mathbf{D})$. If that is not the case, we necessarily have that $|\mathsf{stem}(\mathbf{D})| > |\mathsf{stem}(\mathbf{H})|$. We conclude that the measure indeed decreases in either case.

Finally, note that once the measure is 0 we have that $\mathsf{stem}(\mathbf{K}) = h$ and the etog in question is therefore linear. Since we are only considering connected patterns, the maximum value the measure can possibly take is $(h-1)h + h - 1 = h^2 - 1$. We conclude that the recurrence has depth at most $|\mathbf{H}|^2$. □

3.3 Computing Defects

For the remainder of this section, fix $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ where $\bar{x} := \mathsf{stem}(\mathbf{H})$. Let also $V_1 := V(\mathbf{H}_1) - \bar{x}$ and $V_2 := V(\mathbf{H}_2) - \bar{x}$ be the vertex sets exclusive to $\mathbf{H}_1$ and $\mathbf{H}_2$.

Definition 12 (*Monotone*) Let $\preccurlyeq$ be a partial order over a set S and let $M \subseteq \binom{S}{2}$ be a matching. Let further $\vec{D}$ be the digraph representation of $\preccurlyeq$. We say that M is *monotone* with respect to $\preccurlyeq$ if the digraph obtained from $\vec{D}$ by identifying the pairs in M is a dag.

Definition 13 (*Defect map*) A *defect map* is a bijection $\kappa : \bar{x} \cup \tilde{V}_1 \to \bar{x} \cup \tilde{V}_2$ for subsets $\tilde{V}_1 \subseteq V_1$ and $\tilde{V}_2 \subseteq V_2$ with the following properties:

- κ is an isomorphism between $H[\bar{x} \cup \tilde{V}_1]$ and $H[\bar{x} \cup \tilde{V}_2]$,
- the matching $\{x\kappa(x) \mid x \in \tilde{V}_1\}$ is monotone with respect to $\preccurlyeq_{\mathbf{H}}$.

In the following we construct a set of etogs $\mathcal{D}'$ and prove that it is precisely $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$. Given the decomposition $\mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ of $\mathbf{H}$, we generate the etogs in $\mathcal{D}'$ as follows:

1. Select appropriate subsets $\tilde{V}_1 \subseteq V_1$ and $\tilde{V}_2 \subseteq V_2$ and a defect map $\kappa : \bar{x} \cup \tilde{V}_1 \to \bar{x} \cup \tilde{V}_2$. Let $M := \{x\kappa(x) \mid x \in \tilde{V}_1\}$.
2. Identify the pairs matched by M in H to create the (unordered) graph H' and create the relation $\preccurlyeq_M$ from $\preccurlyeq_{\mathbf{H}}$ by the same process.
3. Select a set $E^+ \subseteq (V_1 - \tilde{V}_1) \times (V_2 - \tilde{V}_2)$ with $E^+ \cap E(H) = \emptyset$ and add it to H'; we only allow $E^+ = \emptyset$ if $\tilde{V}_1, \tilde{V}_2 \neq \emptyset$.
4. For every linear ordering $\leq$ of $V(H)$ that is compatible with $\preccurlyeq_M$, add the graph $\mathsf{relax}((H', \leq))$ to $\mathcal{D}'$.

For compatibility with Definition 11, whenever we identify vertices $xy \in M$, we label the resultant vertex x, thus $V_1 \subseteq V(H')$.

Theorem 1 *The above process generates exactly* $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$.

We prove Theorem 1 by showing the following two lemmas.

Lemma 8 $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2) \subseteq \mathcal{D}'$.

Proof Consider $\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$ and let $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$ such that ϕ is the identity on the set $V(\mathbf{H}_2) \setminus V(\mathbf{H}_1) \cup \bar{x}$. Recall that, by convention, $\mathbf{H}_1 \overset{id}{\hookrightarrow} \mathbf{D}$. Let $\tilde{V} := V_1 \cap \phi(V_2)$ and define the mapping $\kappa\colon \bar{x} \cup \tilde{V} \to \bar{x} \cup \phi^{-1}(\tilde{V})$ as the identity on $\bar{x}$ and $\kappa(x) := \phi^{-1}(x)$ for $x \in \tilde{V} \subseteq V_1$. Let further $M := \{x\kappa(x) \mid x \in \tilde{V}\}$.

Claim *κ is a defect map.*

Proof Since $\mathbf{H}_1[\bar{x} \cup \tilde{V}] \overset{id}{\hookrightarrow} \mathbf{H}[\bar{x} \cup \tilde{V}]$ and $\mathbf{H}_2[\kappa(\bar{x} \cup \tilde{V})] \overset{\kappa^{-1}}{\hookrightarrow} \mathbf{H}[\bar{x} \cup \tilde{V}]$ we conclude that κ is an isomorphism of the underlying graphs $H_1[\bar{x} \cup \tilde{V}]$ and $H_2[\kappa(\bar{x} \cup \tilde{V})]$.

Let $\vec{O}$ be the digraph representation of $\preccurlyeq_{\mathbf{H}}$ and $\vec{O}''$ of $\preccurlyeq_{\mathbf{D}}$. By construction, $\vec{O}''$ is precisely the digraph obtained from $\vec{O}$ by identifying the pairs matched in M. Since $\mathbf{D}$ is a tree-ordered graph, $\preccurlyeq_{\mathbf{D}}$ is a partial order and thus $\vec{O}''$ is a dag. In other words, the matching M is monotone with respect to $\preccurlyeq_{\mathbf{H}}$ and we conclude that κ is a defect map. □

Let $\tilde{V}_1 := \tilde{V}$ and $\tilde{V}_2 := \kappa(\tilde{V})$ in the following. Define $E^+ := E(\mathbf{D}) \cap ((V_1 - \tilde{V}_1) \times (\phi(V_2) - \tilde{V}_1))$. Let H' be the graph obtained from H by identifying the pairs matched by M and adding E^+ to it. Let further $\preccurlyeq_M$ be the relation obtained from $\preccurlyeq_{\mathbf{H}}$ by identifying the pairs matched by M. It is left to show that there exists a linear order $\leq$ which is compatible with $\preccurlyeq_M$ and satisfies $D = \mathsf{relax}((H', \leq))$. Let $\vec{O}'$ be the digraph representation of $\preccurlyeq_M$ and let $\vec{O}''$ be again the digraph representation of $\preccurlyeq_{\mathbf{D}}$. Note that the difference between $\vec{O}'$ and $\vec{O}''$ are arcs corresponding to an orientation $\vec{E}^+$ of E^+ and transitive arcs resulting from the addition of $\vec{E}^+$. Since all edges in E^+ are between $V_1 - \tilde{V}$ and $\phi(V_2) - \tilde{V}$ and those two sets are disjoint, we can choose, for example, to orient $\vec{E}^+$ by letting all arcs point towards $\phi(V_2) - \tilde{V}$. Then $\vec{O}' \cup \vec{E}^+$ is a digraph and so is its transitive closure $\tilde{O}''$. Now note that every topological ordering $\leq$ of $\vec{O}''$ is also a topological ordering of $\vec{O}'$ and we conclude that $\preccurlyeq_M$ is compatible with $\preccurlyeq_D$. Since $\mathbf{D}$ is an etog, these orderings also all satisfy $\mathsf{relax}((H', \leq)) = D$. We conclude that $D \in \mathcal{D}'$ and therefore $\mathcal{D}(\mathbf{H}_1, \mathbf{H}_2) \subseteq \mathcal{D}'$. □

Lemma 9 $\mathcal{D}' \subseteq \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$.

Proof Consider $\mathbf{D} \in \mathcal{D}'$ and let $\tilde{V}_1, \tilde{V}_2, \kappa, E^+$ and $\leq$ be those choices that generated $\mathbf{D}$. Let also H' be the graph generated by identifying the pairs matched by $M := \{x\kappa(x) \mid x \in \tilde{V}_1\}$ in H and adding E^+ to the resulting graph. We need to show that $\mathbf{D}$ is indeed a defect; note that by the last step of the construction it is necessarily an etog.

First, let us convince ourselves that $\mathbf{H} \not\hookrightarrow \mathbf{D}$. If $\tilde{V}_1 \neq \emptyset$, then $\mathbf{D}$ has less vertices than $\mathbf{H}$ and thus no embedding can exist. Otherwise, we have that E^+ is non-empty and therefore $\mathbf{D}$ has more edges than $\mathbf{H}$, again no embedding can exist.

Next, we need to show that $\mathbf{H}_1 \overset{id}{\hookrightarrow} \mathbf{D}$. We chose to label the vertices from identifying the pairs in M by their respective endpoint in $\tilde{V}_1$. Furthermore, no edge in E^+ has both its endpoints in $\bar{x} \cup V_1$, therefore $\mathbf{D}[\bar{x} \cup V_1] = \mathbf{H}[\bar{x} \cup V_1]$ and therefore $\mathbf{H}_1 \overset{id}{\hookrightarrow} \mathbf{D}$.

Similarly, we need to show that $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$. Define ϕ to be the identity on $\bar{x} \cup V_2 \setminus V_1$ and κ^{-1} on $\tilde{V}_2$. Again, no edge in E^+ has both its endpoints in $\phi(\mathbf{H}_2)$ and hence $\mathbf{H}[V_2] \overset{id}{\hookrightarrow} \mathbf{D}[\phi(\mathbf{H}_2)]$ and therefore $\mathbf{H}_2 \overset{\phi}{\hookrightarrow} \mathbf{D}$.

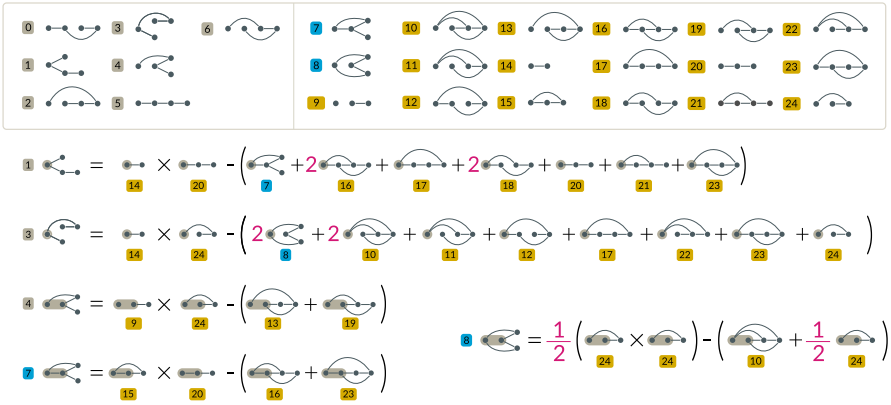


Fig. 2 Top: Patterns (0–6), defects (7–8) and pieces (9–24) needed to count a path on four vertices. The arrangements indicate the tree order, all pattern relaxations except 1,3,4,7 and 8 are linear. Bottom: Algebraic expressions to compute non-linear patterns, gray boxes indicate the stems. The graphs are understood as automorphism-corrected embedding counts $\widehat{\#}_{\bar{x}\mapsto\bar{y}}(\mathbf{H},\mathbb{G})$ (see note below Lemma 6). For example, in order to compute the number of embeddings of defect 8 which map its stem $\bar{x}$ onto a vertex pair $\bar{y}$ in the host graph, we first need to compute the number of embeddings of the pieces 24 and 10 which likewise map the first two vertices of their stem prefix onto $\bar{x}$

Finally, it follows directly from the construction of $\mathbf{D}$ that indeed $V(\mathbf{D}) = \bar{x} \cup V_1 \cup \phi(V_2) = V(\mathbf{H}_1) \cup \phi(\mathbf{H}_2)$, thus we conclude that $\mathbf{D}$ is indeed a defect. It follows that $\mathcal{D}' \subseteq \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$, as claimed (Fig. 2). □

4 The Algorithms

In order to efficiently implement the counting algorithm we need a data structure C which acts as a map from ordered vertex sets to integers; the idea being that for a fixed pattern relaxation $\mathbf{H} \in \mathcal{H}$ with stem $\bar{x}$ we store in $\mathsf{C}[\bar{y}]$, $\bar{y} \subset \mathbb{G}$ how many embeddings $\mathbf{H} \xhookrightarrow{\bar{x}\mapsto\bar{y}} \mathbb{G}$ exist. We use Lemmas 3 or 4 to populate these counters for all linear pieces of $\mathbf{H}$ and then use Lemma 6 to progressively compute counts for larger and larger pieces of $\mathbf{H}$ until we arrive at a count for $\mathbf{H}$ itself. We organize the progressive decompositions of $\mathbf{H}$ and the coefficients resulting from the application of Lemma 6 in a *counting dag*. Leaves of the counting dag correspond to linear pieces of $\mathbf{H}$, the single source to $\mathbf{H}$ itself. The computation then proceeds from the leaves upwards; a task can be completed as soon as all its out-neighbors have been completed (leaf nodes are completed by applying Lemmas 3 or 4).

Repeating this procedure for all pattern relaxations in $\mathcal{H}$ and correcting the sum by the number of automorphisms of H then gives us the total number of times H appears as an induced subgraph of G. For convenience, we compute a joint counting dag for all relaxations $\mathcal{H}$ and read of the final value from all its source nodes—note that in practice this will save some computations since the counting dags likely have nodes in common.

We first outline the notation and necessary operations of C and then discuss how it can be implemented, then we describe the counting dag and then finally provide

the algorithms. We will assume in the following that $\mathbb{G}$ is a linear ordering of G, we present the algorithm with a dependence on $\mathrm{wcol}_{|H|}(\mathbb{G})$ and show what modifications have to be made to arrive at an algorithm depending on $\mathrm{col}_{|H|}(\mathbb{G})$ instead. At the end of the section we will use Propositions 3 and 5 to express the running times in terms of $\mathrm{wcol}_{|H|}(G)$ and $\mathrm{col}_{|H|}(G)$, respectively.

4.1 Counting Data Structure

The *counting data structure* C *of depth* d is a map from d-length ordered vertex sets $\bar{y} \subseteq \mathbf{G}$ to positive integers $\mathsf{C}[\bar{y}]$. Initially, the counting data structure contains a count of zero for every possible key. We write $|\mathsf{C}|$ to denote the number of keys stored in C with non-zero counts. The data structure supports the following queries and modifications:

- *Increment* count $\mathsf{C}[\bar{y}]$ by any integer for tuples $\bar{y}$ of length d in time $O(d)$;
- Answer the *prefix query*

$$\mathsf{C}[\bar{y}] := \sum_{\substack{\bar{z}:|\bar{z}|=d \\ \text{and } \bar{z}|_r=\bar{y}}} \mathsf{C}[\bar{z}]$$

  for tuples $\bar{y}$ of length $r \leq d$ in time $O(r)$;
- for $\gamma \in \mathbb{R}$ we can compute the *scalar product* $\gamma\mathsf{C}$ with

$$(\gamma\mathsf{C})[\bar{y}] := \gamma \cdot \mathsf{C}[\bar{y}] \quad \forall \bar{y} \in V(\mathbb{G})^d$$

  in time $O(d|\mathsf{C}|)$.

Given two counting data structures $\mathsf{C}_1, \mathsf{C}_2$ of depth $\geq r$ the following two operations must be supported:

- The r*-depth difference* $\mathsf{C}_1 -_r \mathsf{C}_2$ with

$$(\mathsf{C}_1 -_r \mathsf{C}_2)[\bar{y}] := \mathsf{C}_1[\bar{y}] - \mathsf{C}_2[\bar{y}] \quad \forall \bar{y} \in V(\mathbb{G})^r$$

  in time $O(r \cdot \max(|\mathsf{C}_1|, |\mathsf{C}_2|))$;
- the r*-depth product* $\mathsf{C}_1 *_r \mathsf{C}_2$ with

$$(\mathsf{C}_1 *_r \mathsf{C}_2)[\bar{y}] := \mathsf{C}_1[\bar{y}] \cdot \mathsf{C}_2[\bar{y}] \quad \forall \bar{y} \in V(\mathbb{G})^r$$

  in time $O(r \cdot \max(|\mathsf{C}_1|, |\mathsf{C}_2|))$.

A convenient way to implement C is a prefix-trie in which every node contains a counter (which contains the sum-total of all values stored below it) and a dynamically sized hash-map to store its descendants. It is trivial to update the counters during an increment and answering the prefix query $\mathsf{C}[\bar{y}]$ amounts to locating the node with prefix $\bar{y}$ in C and returning its counter in time $O(|\bar{y}|)$.

Since we can easily enumerate all keys contained in C by a depth-first traversal, implementing the scalar product can be done by first creating an empty counting data

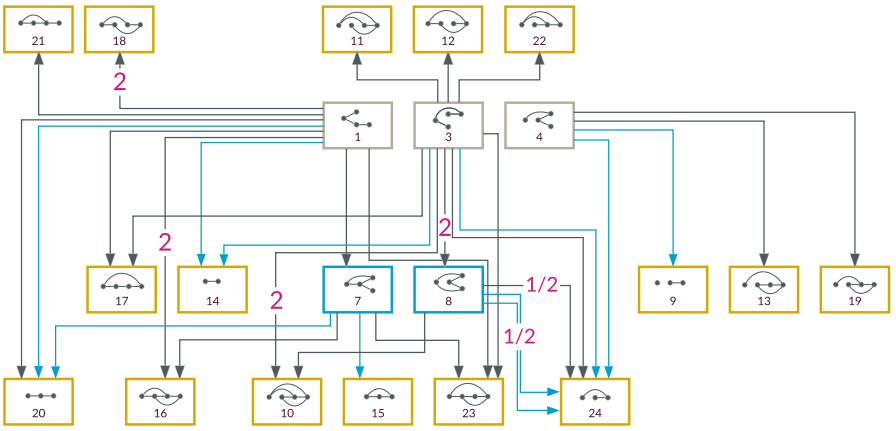


Fig. 3 Task-dag for counting a path on four vertices, as depicted in Fig. 2. Blue arcs belong to edges $E^{\times}$, gray arcs to E^{-}. Note that the coefficients are for the automorphism-corrected counts, therefore the hyperedges $E^{\times}$ need to be imbued with a weight as well (Color figure online)

structure C' and inserting all keys $\bar{x}$ contained C by incrementing the value of $\mathsf{C}'[\bar{x}]$ by $\gamma\mathsf{C}[\bar{x}]$. The DFS on C takes time $O(|\mathsf{C}|)$ and each insertion takes time $O(d)$, hence the claimed running time holds true (Fig. 3).

To perform the r-depth difference and product we traverse the two tries C_1 and C_2 in lockstep, meaning that we only descend in the DFS if the two currently active nodes x_1 in C_1 and x_2 in C_2 both have a child with the same respective key, and truncating the DFS at depth r. During this traversal, it is easy to populate a new trie to obtain the final result $(\mathsf{C}_1 -_r \mathsf{C}_2)$ or $(\mathsf{C}_1 *_r \mathsf{C}_2)$. The lockstep DFS takes time $O(\max(|\mathsf{C}_1|, |\mathsf{C}_2|))$, each insertion into the resultant trie takes time $O(r)$ and the running time follows.

4.2 The Counting Dag

A *counting dag* is a directed hypergraph $(\mathcal{V}, E^{\times}, E^{-})$ with two types of edges. $E^{\times} \subseteq \mathcal{V}^3$ contains edges of the form $(\mathbf{H}, \mathbf{H}_l, \mathbf{H}_r)$ with $\mathbf{H} = \mathbf{H}_r \oplus_{\bar{x}} \mathbf{H}_l$ which indicate that in order to compute $\#_{\bar{x}\mapsto\bullet}(\mathbf{H})$ by application of Lemma 6, we need to compute $\#_{\bar{x}\mapsto\bullet}(\mathbf{H}_l)$ and $\#_{\bar{x}\mapsto\bullet}(\mathbf{H}_r)$ first because they appear in the product on the right hand side. Every node in $\mathcal{V}$ has at most one outgoing hyperedge in $E^{\times}$. Also note that $\mathbf{H}_l = \mathbf{H}_r$ is possible.

Similarly, $E^{-} \subseteq \mathcal{V}^2 \times \mathbb{R}$ contains edges of the form $(\mathbf{H}, \mathbf{D}, \gamma)$ where $\mathbf{D}$ is a defect of $\mathbf{H}_l, \mathbf{H}_r$ and in order to compute $\#_{\bar{x}\mapsto\bullet}(\mathbf{H})$ from $\#_{\bar{x}\mapsto\bullet}(\mathbf{H}_l) \cdot \#_{\bar{x}\mapsto\bullet}(\mathbf{H}_r)$ we need to subtract $\gamma \cdot \#_{\bar{x}\mapsto\bullet}(\mathbf{D})$ for all such edges.

For two counting dags $\vec{C}, \vec{C}'$ we write $\vec{C} \cup \vec{C}'$ to denote the union of their vertex and edge sets. With that notation in mind, Algorithm 1 shows how to compute a counting dag for a given etog $\mathbf{H}$. Note that the choice of decomposition $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$ is arbitrary, reasonable choices include either letting $\mathbf{H}_1$ be as small as possible or trying to balance the size of $\mathbf{H}_1$ and $\mathbf{H}_2$.

Lemma 10 *Given a graph H on h vertices, we can construct a counting dag $\vec{C}(H)$ with $\|\vec{C}\| = O(3^{h^2})$ using Algorithm 1 in time $O(\|\vec{C}\|)$.*

Input: An etog $\mathbf{H}$.
Output: A counting dag $\vec{C}(\mathbf{H})$.

function `decompose`$(\mathbf{H})$
  Initialize $\vec{C}$ as an empty counting dag
  Let $\bar{x} = \mathsf{stem}(\mathbf{H})$
  Choose decomposition $\mathbf{H} = \mathbf{H}_1 \oplus_{\bar{x}} \mathbf{H}_2$
  $\vec{C} \leftarrow \vec{C} \cup$ `decompose`$(\mathbf{H}_1)$
  $\vec{C} \leftarrow \vec{C} \cup$ `decompose`$(\mathbf{H}_2)$
  $E^{\times}(\vec{C}) \leftarrow E^{\times}(\vec{C}) \cup \{(\mathbf{H}, \mathbf{H}_1, \mathbf{H}_2)\}$
  for $\mathbf{D} \in \mathcal{D}(\mathbf{H}_1, \mathbf{H}_2)$ **do**
    $\eta \leftarrow \#_{\bar{x}\mapsto\bar{x}}(\mathbf{H}, \mathbf{D} \mid \mathbf{H}_1, \mathbf{H}_2)$
    $\alpha \leftarrow \#_{\bar{x}\mapsto\bar{x}}(\mathbf{D}, \mathbf{D})$
    $\vec{C} \leftarrow \vec{C} \cup$ `decompose`$(\mathbf{D})$
    $E^{-}(\vec{C}) \leftarrow E^{-}(\vec{C}) \cup \{(\mathbf{H}, \mathbf{D}, \eta/\alpha)\}$
  return $\vec{C}$

Algorithm 1: Recursive computation of a counting dag.

Proof We enumerate the at most $h!$ etogs of H and run Algorithm 1, then we take the union of all resulting counting dags to obtain $\vec{C} := \vec{C}(H)$.

If we employ memoization across the calls we can upper-bound the running time and the size of the counting dag by the total number of togs on $\leq h$ vertices.

We arrive at a crude upper bound by noting that there are at most $O(2.956^h)$ tree orders on h vertices [16, chap. 2.3.4.4], each of which can contain up to $\binom{h}{2}$ edges; hence

$$\|\vec{C}\| \leq |\vec{C}|^2 = O\Big(\big(2.956^h \cdot 2^{\frac{1}{2}(h^2-h)}\big)^2\Big) = O\Big(2^{h^2+2.128h}\Big) = O\big(3^{h^2}\big).$$

We obtain the same bound on the running time:

$$O(h! \cdot \|C\|) = O(2^{h \log h} \cdot 2^{h^2+2.128h}) = O\big(3^{h^2}\big).$$

□

4.3 The Algorithms

The following proofs of the worst-case running time are not very indicative of the algorithms performance as a) the term $O(3^{h^2})$ is a (crude) upper bound on the size of the counting dag and b) not every pattern of size h needs to use the $(h-1)$-reachable sets. As an extreme example, the graph K_h only needs 1-reachable sets, e.g. only a degeneracy ordering of the host graph. We include a table with sizes of counting dags and the necessary depth for various patterns graph in the subsequent section.

Lemma 11 *Algorithm 2 computes the number of induced embeddings of H into G in time $O(\|\vec{C}\| \cdot h \operatorname{wcol}_h(\mathbb{G})^{h-1}|G|) = O(3^{h^2} \cdot h \operatorname{wcol}_h(\mathbb{G})^{h-1}|G|)$ where $h := |H|$.*

Input: A linear host graph $\mathbb{G}$ and a counting dag $\vec{C}(H)$ of a pattern H.
Output: The number of embeddings of H into induced subgraphs of G

① Initialize counting data structures
Compute topological ordering $\mathbf{H}_1, \ldots, \mathbf{H}_\ell$ of $V(\vec{C})$ such that $\mathbf{H}_1, \ldots, \mathbf{H}_s$ are source-nodes in $\vec{C}$ and $\mathbf{H}_t, \ldots, \mathbf{H}_\ell$ are sink-nodes in $\vec{C}$
Initialize counting data structures C_i of depth $|\mathsf{stem}(\mathbf{H}_i)|$ for $i \in [1, \ell]$
② Count linear patterns
for $i \in [t, \ell]$ **do**
  for $v \in \mathbb{G}$ **do**
    ③ Count patterns ending in v using weak reachability
    $W \leftarrow W_{\mathbb{G}}^{|\mathbf{H}|}(v)$
    for $\bar{y} \in W^{|\mathbf{H}_i|-1}$ **do**
      if $\mathbf{H}_i \xrightarrow{V(\mathbf{H}) \mapsto \bar{y}v} \mathbb{G}$ **then**
        $\mathsf{C}_i[\bar{y}v] \leftarrow \mathsf{C}_i[\bar{y}v] + 1$;
④ Propagate counts
for $i \in (t-1, t-2, \ldots, 1)$ **do**
  Let l, r be the indices for which $(\mathbf{H}_i, \mathbf{H}_l, \mathbf{H}_r) \in E^{\times}(\vec{C})$
  $k \leftarrow |\mathsf{stem}(\mathbf{H}_i)|$
  $\mathsf{C}_i \leftarrow \mathsf{C}_l *_k \mathsf{C}_r$
  for $(\mathbf{H}, \mathbf{H}_d, \gamma) \in E^{-}(\vec{C})$ **do**
    $\mathsf{C}_i \leftarrow \mathsf{C}_i -_k \gamma \mathsf{C}_d$
⑤ Sum up counts in source-nodes
$c \leftarrow 0$
for $i \in (1, \ldots, s)$ **do**
  $c \leftarrow c + \mathsf{C}_i[\emptyset]$
return c

Algorithm 2: The subgraph counting algorithm using weak reachability. Note that part ① is independent of $\mathbb{G}$, hence the counting dag $\vec{C}$ for any given pattern graph H can be precomputed. The part marked with ③ can be optmized further in practice by matching the vertices in $\bar{y}$ one-by-one and discarding unsuitable combinations early.

Proof In part ①, for every one of the $\ell := \text{leaves}(\vec{C})$ many sinks $\mathbf{H}_i$, $i \in [t, \ell]$, of $\vec{C}$ we fill the counting data structure C_i in time $O(\text{wcol}_h(\mathbb{G})^{h-1}|G|)$ by application of Lemma 3.

Since every counting data structure at the end of part ② contains at most $O(\text{wcol}_h(\mathbb{G})^{h-1}|G|)$ many tuples, it follows that all operations in step ④ on counting data structures (r-depth products, differences, scalar products) can be computed in time $O(h\,\text{wcol}_h(\mathbb{G})^{h-1}|G|)$. The number of such operations is proportional to $\|\vec{C}\|$, thus in total step ④ takes times $O(\|\vec{C}\| \cdot h\,\text{wcol}_h(\mathbb{G})^{h-1}|G|)$. The time taken in step ⑤ is negligible compared to the previous steps and we conclude that the total running time is as claimed.

The correctness of the algorithm follows by induction over the counting dag: the leaf counts are correct by Lemma 3 and the counts at the internal nodes are correct by Lemma 6. □

Input: A linear host graph $\mathbb{G}$ and a counting dag $\vec{C}(H)$ of a pattern H.
Output: The number of embeddings of H into induced subgraphs of G

```
⋮
(2) Count linear patterns
for i ∈ [t, ℓ] do
    for v ∈ 𝔾 do
        (3) Count patterns ending in v using strong reachability
        h ← |H_i|
        for x_{h−1} ∈ S_𝔾^{|H|}[v] do
            j ← hint^{H}_{H_i}(h − 2)
            for x_{h−2} ∈ S_𝔾^{|H|}[x_j] do
                ⋱
                j ← hint^{H}_{H_i}(2)
                for x_1 ∈ S_𝔾^{|H|}[x_j] do
                    if H_i ≃ 𝔾[x_1, ..., x_p] then
                        C_i[x_1, ..., x_p] ← C_i[x_1, ..., x_p] + 1
                ⋰
⋮
```

Algorithm 3: Modification of Algorithm 2 to use strong instead of weak reachability. For ease of presentation, the algorithm is shown as a sequence of nested loops instead of recursion or a loop with a stack of partial solutions.

Combining the above lemma with Propositions 4 and 5 , we immediately obtain the following:

Corollary 2 *There exists an algorithm that given a graph G and a graph H on h vertices computes the number of times H appears as an induced subgraph in G in total time $O((3h^2 \operatorname{wcol}_h(G))^{h^2}|G|)$.*

Note that this running time is linear in $|G|$ if G is taken from a class with bounded expansion. Further, if we consider h to be constant, then the running time is $O(|G|^{1+o(1)})$, i.e. almost linear, when G is taken from a nowhere dense class.

Exchanging Lemma 3 for Lemma 4 in the above proof shows a similar running time for the variants using strong reachability:

Lemma 12 *Algorithm 3 computes the number of induced embeddings of H into G in time $O(\|\vec{C}\| \cdot h \operatorname{col}_h(\mathbb{G})^{h-1}|G|) = O(3^{h^2} \cdot h \operatorname{col}_h(\mathbb{G})^{h-1}|G|)$ where $h := |H|$.*

Corollary 3 *There exists an algorithm that given a graph G and a graph H on h vertices computes the number of times H appears as an induced subgraph in G in total time $O((3h^2 \operatorname{col}_h(G))^{h^2}|G|)$.*

5 Discussion

The goal of our work is to design an algorithm for subgraph counting that has the potential for being useful in practice, which is why we chose to avoid certain tools from the bounded expansion toolkit which—as discussed in the introduction—would immediately render our algorithm unuseable. Outside these questions of algorithmic feasibility, we then need to ask whether real-world networks exhibit the presumed properties, namely, whether they "have bounded expansion". As cited in the Introduction, there is theoretical and empirical evidence supporting the hypothesis that some types of real-world networks can be treated as bounded expansion graphs. However, we still need to address the fact that the notion is ill-defined for single instances.

We could approach this problem by computing statistics like the weak r-coloring number for a network and then decide based on those numbers whether the network has "bounded expansion" or not. This is of course deeply unsatisfactory (though useful as a *comparative* approach) as we have to draw arbitrary boundaries around how large the statistics should be for any given r and which values of r we should consider.

We believe that the better approach is to side-step the more philosophical question of whether a single given network is better treated as "bounded expansion" or not and instead simply run the algorithm in question and see whether it appears to be useful/competitive. Algorithms designed for bounded expansion classes by design will still work on dense inputs, unlike e.g. certain algorithms that assume that the input graph is planar. Grounding the experiments in known use-cases then also provides us with a natural limit on the 'depth' r at which we operate the algorithm. In our case, practical subgraph counting is usually limited to pattern graphs of size ≤ 5 [23] which provides an upper bound on r for our purposes (see column 'd' in Table 1).

With these considerations in mind, let us now discuss our proof-of-concept implementation[7] along with preliminary experimental results. Several observations on natural extensions of this algorithm follow.

In practice, Algorithm 2 and 3 have a lot of engineering potential. In most cases, the search space for linear patterns is much smaller than the h-weak or strong neighborhoods since previously-fixed vertices will often have the sought vertices in their left neighborhood or in a weak/strong neighborhood at distance less than h. Furthermore, the task dag for a given pattern can be precomputed and optimized; in order to minimize memory use, we can process tasks in an order which enables us to delete counting data structures as soon as they have been propagated along all in-edges.

Since we view the counting dag computation as a form of pre-processing, we implemented this stage using Python and show results for several small pattern graphs in Table 1. We have not yet explored whether different decomposition strategies (i.e. which piece-sum decomposition to choose if there are multiple options) significantly impact the size of these dags. As expected, the counting dag is smaller for denser graphs—for complete graphs the algorithm essentially reduces to the well-known clique-counting algorithm for degenerate graphs.

For the subgraph counting algorithm, we choses to implement in Rust. While we recognize that there is significant additional optimization and engineering needed, it

[7] Code available under a BSD 3-clause license at http://www.github.com/theoryinpractice/mandoline.

Table 1 Size and number of leaves for counting dags for various small graphs. The final column gives the reachability-depth d necessary to count the specified pattern. Named graphs can be looked up under http://www.graphclasses.org/smallgraphs.html

n	G	$\lvert\vec{C}\rvert$ (leaves)	$\lVert\vec{C}\rVert$	d
	P_3	5 (4)	3	1
	P_4	25 (20)	26	2
	P_5	247 (186)	552	3
	C_3	1 (1)	0	1
	C_4	5 (4)	3	1
	C_5	32 (27)	27	2
	C_6	424 (338)	689	2
	S_3	14 (9)	21	1
	S_4	60 (36)	200	2
	S_5	619 (389)	4919	2
	W_3	1 (1)	0	1
	W_4	21 (18)	9	1
	W_5	141 (123)	90	2
	W_6	1707 (1395)	2332	2
	K_i	1 (1)	0	1
	$K_{2,2}$	5 (4)	3	1
	$K_{3,3}$	24 (17)	27	1
	$K_{4,4}$	132 (87)	281	2
	$K_{5,5}$	890 (620)	1570	2
4	Diamond	8 (7)	3	1
	Paw	18 (15)	12	1
5	Butterfly	56 (44)	85	2
	Gem	90 (77)	61	2
	Cricket	94 (65)	226	2
	House	110 (92)	88	2
	Dart	121 (93)	171	2
	Kite	141 (116)	175	2
	Bull	199 (154)	325	2
6	Co-net	371 (306)	441	2
	Domino	723 (572)	1110	2
	Co-domino	733 (606)	1050	2
	Co-fish	908 (734)	1515	2
	Net	1805 (1388)	4333	3
	Fish	2052 (1556)	5436	3

is notable that runtimes remain reasonable (see Table 2) on host graphs with tens of thousands of nodes even for relatively large patterns (all measurements where taken on a simple laptop with an intel i5 core and 4GB RAM).

Additionally, we chose to use a very simple heuristic for the vertex ordering by sorting the vertices by descending degree. Nadara *et al*. showed that these orderings yield acceptable results for weak coloring numbers on real-world networks [17] and

Table 2 Runtimes for counting several common patterns in five real-world networks

Network	n	m	P_5	W_5	bull	$K_{4,4}$
soc-advogato	6551	43,427	4m36s	1m17s	2m23s	1m5s
cora-citation	23,166	89,157	3m23s	2m8s	2m58s	1m33s
ca-CondMat	23,133	93,497	3m26s	1m57s	2m41s	1m21s
Google+	23,628	39,194	2m57s	1m45s	2m13s	1m18s
digg	30,398	86,312	5m50s	2m53s	3m38s	2m8s

are, of course, very fast to compute. In the future, we will investigate the impact of computing better orderings (using e.g. the more involved heuristics discussed in [17]) on the subsequent counting step and how these two phases of the algorithm should be balanced.

We plan to engineer these implementations further and compare this approach to other subgraph-counting algorithms on a larger corpus of host and pattern graphs in future work. Note that it is straightforward to extend our algorithm to edge- and vertex-labelled graphs by defining isomorphisms and embeddings appropriately. We chose not to include labels here as they add another layer of notation that would make the presentation less clear.

We also, for simplicity, assumed that the pattern graph H is connected. This is easily remedied by a labelled version of the algorithm: we add an apex vertex with a unique label to both H and G and make it the minimum in $\mathbb{G}$. Alternatively, the presented algorithm can be modified by allowing piece-sums to work on connected components. This modification does not significantly change the algorithm, but adds additional cases in many proofs.

Finally, we observe that the approach presented here can be modified to count non-induced subgraphs, subgraph homomorphisms or boolean queries instead by adjusting the notions of patterns and pattern decompositions appropriately.

Acknowledgements We thank Marc Roth for pointing out a misattribution of parameterized hardness results in an earlier version of this paper. This work was supported in part by the Gordon & Betty Moore Foundation under award GBMF4560 to Blair D. Sullivan.

References

1. Brown, C.T., Moritz, D., O'Brien, M.P., Reidl, F., Reiter, T., Sullivan, B.D.: Exploring neighborhoods in large metagenome assembly graphs using spacegraphcats reveals hidden sequence diversity. Genome Biol. **21**(1), 1–16 (2020)
2. Chen, Y., Thurley, M., Weyer, M.: Understanding the complexity of induced subgraph isomorphisms. In: International Colloquium on Automata, Languages, and Programming, pp. 587–596. Springer, (2008)
3. Demaine, E.D., Reidl, F., Rossmanith, P., Sánchez Villaamil, F., Sikdar, S., Sullivan, B.D.: Structural sparsity of complex networks: bounded expansion in random models and real-world graphs. J. Comput. Syst, Sci (2019)
4. Dobler, A., Sorge, M., Villedieu, A.: Turbocharging heuristics for weak coloring numbers. In: 30th Annual European Symposium on Algorithms (ESA 2022), volume 244 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 44:1–44:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022)

5. Dvořák, Z.: Constant-factor approximation of the domination number in sparse graphs. Eur. J. Comb. **34**(5), 833–840 (2013)
6. Dvořák, Z.: On weighted sublinear separators. J. Graph Theory **100**(2), 270–280 (2022)
7. Dvořák, Z., Tůma, V.: A dynamic data structure for counting subgraphs in sparse graphs. In: Workshop on Algorithms and Data Structures, pp. 304–315. Springer (2013)
8. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. In: International Symposium on Algorithms and Computation, pp. 403–414. Springer, (2010)
9. Farrell, M., Goodrich, T.D., Lemons, N., Reidl, F., Sánchez Villaamil, F., Sullivan, B.D.: Hyperbolicity, degeneracy, and expansion of random intersection graphs. In: Algorithms and Models for the Web Graph—12th International Workshop, WAW 2015, Eindhoven, The Netherlands, December 10–11, 2015, Proceedings, volume 9479 of Lecture Notes in Computer Science, pp. 29–41. Springer (2015)
10. Flum, J., Grohe, M.: The parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series). Springer, New York (2006)
12. Grohe, M., Kreutzer, S., Rabinovich, R., Siebertz, S., Stavropoulos, K.: Colouring and covering nowhere dense graphs. In: International Workshop on Graph-Theoretic Concepts in Computer Science, pp. 325–338. Springer (2015)
13. Grohe, M., Kreutzer, S., Siebertz, S.: Characterisations of Nowhere Dense Graphs (Invited Talk). In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013), volume 24 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 21–40. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2013)
14. Kazana, W., Segoufin, L.: Enumeration of first-order queries on classes of structures with bounded expansion. In: Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 297–308. ACM (2013)
15. Kierstead, H.A., Yang, D.: Orderings on graphs and game coloring number. Order **20**(3), 255–264 (2003)
16. Knuth, D.E.: The Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd edn. Addison-Wesley, Boston (1997)
17. Nadara, W., Pilipczuk, M., Rabinovich, R., Reidl, F., Siebertz, S.: Empirical evaluation of approximation algorithms for generalized graph coloring and uniform quasi-wideness. J. Exp. Algorithmics **103**, 14:1-14:16 (2018)
18. Nešetřil, J., de Mendez, P.O.: Grad and classes with bounded expansion I. Decompositions. Eur. J. Comb. **29**(3), 760–776 (2008)
19. Nešetřil, J., Ossona de Mendez, P.: On nowhere dense graphs. Eur. J. Comb. **32**(4), 600–617 (2011)
20. Nešetřil, J., de Mendez, P.O.: Sparsity: Graphs, Structures, and Algorithms, volume 28 of Algorithms and Combinatorics. Springer, New York (2012)
21. O'Brien, M.P., Sullivan, B.D.: Experimental evaluation of counting subgraph isomorphisms in classes of bounded expansion. *CoRR*, arXiv:1712.06690 (2017)
22. Reidl, F.: Structural sparseness and complex networks. Dr., Aachen, Techn. Hochsch., Aachen, Aachen, p. 2015. Hochsch., Diss, Techn (2016)
23. Ribeiro, P., Paredes, P., Silva, M.E.P., Aparicio, D., Silva, F.: A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. ACM Comput. Surv. **54**(2) (2021)
24. Zhu, X.: Colouring graphs with bounded generalized colouring number. Discrete Math. **309**(18), 5562–5568 (2009)