MaNIACS: Approximate Mining of Frequent Subgraph Patterns through Sampling*

GIULIA PRETI, CENTAI, Italy
GIANMARCO DE FRANCISCI MORALES, CENTAI, Italy
MATTEO RIONDATO, Amherst College, USA

"And we're zany to the max!" — *Animaniacs* theme song

We present MaNIACS, a sampling-based randomized algorithm for computing high-quality approximations of the collection of the subgraph patterns that are frequent in a single, large, vertex-labeled graph, according to the Minimum Node Image-based (MNI) frequency measure. The output of MaNIACS comes with strong probabilistic guarantees, obtained by using the empirical Vapnik-Chervonenkis (VC) dimension, a key concept from statistical learning theory, together with strong probabilistic tail bounds on the difference between the frequency of a pattern in the sample and its exact frequency. MaNIACS leverages properties of the MNI-frequency to aggressively prune the pattern search space, and thus to reduce the time spent in exploring subspaces that contain no frequent patterns. In turn, this pruning leads to better bounds to the maximum frequency estimation error, which leads to increased pruning, resulting in a beneficial feedback effect. The results of our experimental evaluation of MaNIACS on real graphs show that it returns high-quality collections of frequent patterns in large graphs up to two orders of magnitude faster than the exact algorithm.

CCS Concepts: • **Mathematics of computing → Graph enumeration**; **Approximation algorithms**; **Probabilistic algorithms**; • **Information systems → Data mining**; • **Theory of computation → Sketching and sampling**; **Sample complexity and generalization bounds**.

Additional Key Words and Phrases: Minimum Node Image, Pattern mining, VC-dimension

ACM Reference Format:
Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. 2023. MaNIACS: Approximate Mining of Frequent Subgraph Patterns through Sampling. *ACM Trans. Intell. Syst. Technol.* 14, 3, Article 54 (April 2023), 28 pages. https://doi.org/10.1145/3587254

1 INTRODUCTION

A subgraph pattern (sometimes called "graphlet") is a small graph, possibly with labeled vertices. Frequent Subgraph Pattern Mining (FSPM), i.e., finding the patterns that appear frequently in a single graph, has many applications, from the discovery of protein functionality in computational biology [40, 57], to the development of recommender systems for video games [2], to social media marketing [20], to software engineering [26]. It is also a primitive for graph mining tasks such as classification [18] and clustering [24].

The FSPM task is computationally challenging, for two main reasons: *(i)* the number of possible patterns experiences a combinatorial explosion with the maximum number of vertices in a pattern and with the number of possible vertex labels; and *(ii)* the subgraph isomorphism operation needed

*A preliminary version of this work [47] appeared in the proceedings of ACM KDD'21.

Authors' addresses: Giulia Preti, CENTAI, Corso Inghilterra 3, Turin, 10138, Italy, giulia.preti@centai.eu; Gianmarco De Francisci Morales, CENTAI, Corso Inghilterra 3, Turin, 10138, Italy, gdfm@acm.org; Matteo Riondato, Department of Computer Science, Amherst College, Box #2232, Amherst College, Amherst, MA, 01002, USA, mriondato@amherst.edu.

to find a pattern in the graph is in general NP-complete. Ingenious exact algorithms exist, but they tend to scale poorly with the size of the graph and with the maximum size of a pattern.

The use of *random sampling* is a common solution to speed up time-consuming data analytics tasks, from approximate database query processing [17], to itemset mining [53], to other tasks on graphs [54]. It however comes at the price of obtaining an *approximate solution* to the task at hand. Such solutions are acceptable when they come with *stringent theoretical guarantees on their quality*. A typical approach for sampling algorithms relies on evaluating the function of interest only on a randomly chosen subset of the input domain. In FSPM, one can, e.g., create a small random sample of vertices, and evaluate the presence of subgraph isomorphisms between patterns and only those subgraphs of the graph that include at least one of the sampled vertices.

Random sampling and approximate solutions are *necessary* when access to the graph is restricted, as in online social networks, where one cannot inspect the whole graph, but only query a vertex and its neighborhood through an API. By using vertex sampling schemes [15, 16], an approximation algorithm enables FSPM in this scenario.

The key challenge in using random sampling is understanding the trade-off between the sample size, the time needed to analyze the sample (which depends on the sample size, but also on the analytics task at hand), and the quality that can be obtained from a sample of the specific size. Large deviation bounds can be applied when there is only one function to be estimated, but in FSPM, like in most data analytics tasks, we need to accurately estimate the frequencies of many patterns from the same sample. Classic simultaneous deviation bounds tools such as the union bound, if applied naïvely, are inherently loose, so more sophisticated techniques must be employed. For FSPM, having tight bounds to the maximum estimation error is particularly important, as they are used not only to decide which patterns to include in the approximate solution, but also to *prune the search space* via an apriori-like argument, thus avoiding the expensive step of evaluating the frequency of patterns that are not sufficiently frequent to be included in the output.

Contributions. We present MaNIACS (for "MNI Approximate Computation through Sampling"), an algorithm to compute high-quality approximations of the collection of frequent subgraph patterns from a single, large, vertex-labeled graph, according to the MNI-frequency measure [10].

- MaNIACS relies on *uniform random sampling* of vertices and on computing the patterns to which these vertices belong. MaNIACS is scalable w.r.t. the size of the graph and is the first FSPM algorithm on graphs with restricted access. Sampling allows MaNIACS to be easily parallelized on, e.g., Arabesque [61].
- MaNIACS is the first sampling-based algorithm for the task of FSPM that comes with *strong probabilistic guarantees* on the quality of its output. These guarantees are obtained by leveraging *sample-dependent quantities*: MaNIACS extracts information from the sample to determine the quality of the approximation in terms of the maximum frequency estimation error, which is used to avoid false negatives. The estimated quality is output together with the approximate collection of frequent patterns. To upper bound the maximum estimation error, MaNIACS relies on the *empirical Vapnik-Chervonenkis (eVC) dimension* [64], a fundamental concept from statistical learning theory [59]. The eVC-dimension leads to much better quality guarantees than could be obtained by using classic approaches such as the union bound. We show that the eVC-dimension of the task at hand is *independent from the number of vertex labels*, and we show how to efficiently compute a tight upper bound to this quantity.
- MaNIACS aggressively leverages the *anti-monotonicity* of the MNI-frequency measure (Facts 2 and 3) to prune parts of the search space that provably do not contain any frequent pattern, and to focus the exploration only on the "promising" subspaces, therefore avoiding expensive-but-useless computations. Pruning also leads to better bounds to the maximum

frequency estimation error, which enables additional pruning, thus creating a virtuous cycle that improves both the computational and statistical properties of MaNIACS.
- The results of our experimental evaluation of MaNIACS on real datasets show that it returns a high-quality output very quickly, with even lower error than guaranteed by the theory.

The present article extends the conference version [47] in multiple ways, including:

- We introduce a new variant of the algorithm (Sect. 4.5), which uses different sample sizes to mine patterns of different sizes. This strategy leads to additional pruning at the early levels of the search tree, which results in less work done later, and thus in a significant reduction of the running time of the algorithm, as verified by a new experimental evaluation.
- We compare our approach with Peregrine [29], the state-of-the-art in exact frequent pattern mining in graphs, to prove the effectiveness of our pruning and sampling strategies. Results show that the larger the search space is (e.g., at lower frequency thresholds or when the number of labels is large), the higher the improvement in performance over Peregrine is.
- We clarify a number of extensions of our algorithm to other settings, such as multigraphs, and edge-induced pattern mining, and why we cannot easily extend our approach to other anti-monotonic measures which rely on the concept of *overlap graph* [11].
- We add a discussion on the size of the pattern search space (Sect. 4.2), that, to our knowledge, is the first contribution to the study of the properties of this space for *labeled* patterns.
- We include all the proofs of theorems and lemmas, after carefully tweaking their hypotheses and assumptions, to make our theoretical contributions as general and as strong as possible. We add examples and figures to help the understanding of important concepts.

2 RELATED WORK

There is a vast body of work on subgraph extraction and counting. In the interest of brevity and clarity, we focus on the single, static graph setting, and we omit others (e.g., transactional, dynamic, or stream). For a discussion of these many others areas, we refer the reader to the tutorial by Seshadhri and Tirthapura [58].

The patterns we consider are *connected, unweighted, undirected, vertex-labeled graphs with up to k vertices*. The assignment of the labels to the vertices of the pattern is important, and different assignments (up to automorphisms of the patterns) generate different patterns (see formal definitions in Sect. 3.1). The collection of patterns is therefore different from the collections of *colored graphlets* [51] and *heterogeneous graphlets* [56], which respectively only consider the set or the multiset of vertex labels. Graphlets [48] are a special case of patterns with a single label.

FSPM requires finding patterns with a *global* frequency, for instance as quantified by the popular *Minimum Node Image (MNI)* frequency measure [10] (see eq. (4)), at least as large as a user-specified minimum threshold (see eq. (5)). The MNI measure has also been employed in multigraph settings [27], where edges have multiple labels; adapting MaNIACS to this scenario is straightforward.

The presence of a minimum frequency threshold and the use of the MNI measure distinguish this task from the well-studied task of *counting graphlets or motifs*, which require to compute the global number of vertex- or edge-induced instances of a pattern [4, 8, 9, 25, 28, 42, 45, 50, 65–67]. FSPM is also different from computing the *local* counts, i.e., the number of instances of each pattern in which an edge/vertex participates [43, 56]. The techniques used in these tasks cannot be easily adapted to FSPM. Measures of global frequency for FSPM other than MNI exist [63], such as Harmful Overlap (HO) [22], Maximum Independent Set (MIS) [32], and Minimum Instance (MI) [39]. They differ by the amount of allowed overlap between different pattern instances . Similarly to MNI, they are *anti-monotonic* (see Sect. 3.1), so the pattern search space can be pruned efficiently (see Sect. 4), although the computation of HO and MIS is NP-hard [22, 39], and consequently they have found

less use than MNI [19]. These measures use the concept of *overlap graph of a pattern*, which is the graph whose vertex set is the set of embeddings (subgraph isomorphisms) of a pattern in a given graph, and whose edges depend on the overlap between pairs of embeddings. The frequency is then defined by using properties of this graph (e.g., the size of the maximum independent set, for the MIS [32]). Our approach *cannot* be extended to these measures, because we draw a uniform sample of vertices of the original graph, and even if we could use this sample to build a (non-uniform) sample of the embeddings, i.e., of vertices of the overlap graph, such a sample would not preserve, even approximately, the properties of the overlap graph needed to compute the support measures. Using sampling approaches to compute accurate estimations of these measures with quality guarantees is a very interesting direction for future work.

Algorithms. Elseidy et al. [19] present GraMi, an exact algorithm for FSPM. GraMi transforms the subgraph isomorphism problem into a constraint-satisfaction problem, and uses ingenious computation organization to speed up finding the edge-induced frequent patterns, although not their frequencies. Frequencies are important in pattern mining: since the minimum threshold is often set somewhat arbitrarily, it is important to be able to distinguish between patterns with frequency much greater than the threshold and those that are "barely" frequent. We define the FSPM task to include their exact frequencies (see eq. (5)), which is inherently more difficult. GraMi requires complete access to the whole graph. This assumption is often unrealistic when dealing with online social networks, in addition to being extremely time consuming. In this setting, approximations of the collection of frequent patterns are necessary, and sufficient when they come with stringent quality guarantees, such as the ones provided by MaNIACS (see Thm. 4.7).

Parallel and distributed systems for FSPM try to address the scalability issue of mining frequent patterns from very large graphs or when the pattern search space is huge [1, 13, 29, 60, 61, 68]. MaNIACS can be used as a primitive inside these systems, similarly to how sampling-based approximation algorithms for frequent itemset mining [53] have been integrated in MapReduce [52].

Early works in approximate FSPM include the use of graph summaries [23] or heuristics for space pruning [31], but they offer no guarantees. Other works tackle the problem via graph sampling [5, 49, 69], but they also come with no quality guarantees.

Our algorithm samples a set of vertices, but it does *not* use them to build a graph from the sample. Neither does it sample subgraphs, which is the approach taken by other works on subgraph counting [3, 6–9, 41], nor focuses on *output sampling* [12]. To the best of our knowledge, our work is the first to use concepts from statistical learning theory [64] for FPSM. Other works used VC-dimension or other concepts from statistical learning theory for centrality computations [54], for subgraph counting [41], or for itemsets mining [55], but these approaches cannot be easily adapted to FSPM, because this problem is clearly very different from centrality computation, and because the itemsets space is less complex and much easier to "navigate" than the subgraph space that we consider. In particular, the evaluation of the frequency of an itemset is straightforward and much cheaper than computing the frequency of a subgraph pattern (see Sect. 4). Thus, approaches relying on Rademacher averages for generic pattern families [44] do not perform well for FPSM.

3 PRELIMINARIES

Let us now formally define the important concepts used throughout this work, and the task we are interested in.

3.1 Graph theory concepts

Any graph G we consider is simple (no self loops, no multi-edges),[1] unweighted, undirected, and vertex-labeled, i.e., $G = (V, E, \mathsf{L})$ where L is a function that assigns labels from a fixed set $L = \{\lambda_1, \ldots, \lambda_m\}$ to vertices (unlabeled graphs can be seen as labeled graphs with a single label). For brevity, we usually drop L from the notation, and do not repeat "labeled", but all the graphs we consider are labeled, unless otherwise specified. A graph G is connected iff, for each pair of vertices $v \neq u \in V$, there exists a sequence of vertices $u, w_1, \ldots, w_n, v \in V$ and a sequence of edges $(u, w_1), \ldots, (w_i, w_{i+1}), (w_n, v) \in E$ for $1 \leq i \leq n-1$.

For a fixed $k \in \mathbb{N}$, let $\mathcal{P}$ be the set of all possible *connected* graphs with up to k vertices and whose vertices have labels in L. We call *patterns* the elements of $\mathcal{P}$. Let $S \subseteq V$ be a subset of vertices of a graph $G = (V, E)$, and let $E(S) \doteq \{(u, v) \in E \ : \ u, v \in S\}$. We say that $G_S \doteq (S, E(S))$ is the subgraph of G *induced by* S.[2] For $k > 0$, we define $\mathcal{C}$ to be the set of all *connected induced subgraphs with up to k vertices* in G.[3] All subgraphs we consider are connected induced subgraphs, unless stated otherwise.

Two graphs $G' = (V', E', \mathsf{L}')$ and $G'' = (V'', E'', \mathsf{L}'')$ are *isomorphic* if there exists a bijection $\mu : V' \to V''$ such that $(u, v) \in E'$ iff $(\mu(u), \mu(v)) \in E''$ and the mapping μ *preserves the vertex labels*, i.e., $\mathsf{L}'(u) = \mathsf{L}''(\mu(u))$, for all $u \in V'$. Isomorphisms from a graph G to itself are called *automorphisms* and their set is denoted as $\mathsf{Aut}(G)$.

Given a pattern $P = (V_P, E_P)$ in $\mathcal{P}$ and a vertex $v \in V_P$, the *orbit* $B_P(v)$ *of* v *in* P is the subset of V_P that is mapped to v by any automorphism of P, i.e.,

$$B_P(v) \doteq \{u \in V_P \ : \ \exists \mu \in \mathsf{Aut}(P) \text{ s.t. } \mu(u) = v\} \ . \tag{1}$$

The orbits of P form a partitioning of V_P, for each $u \in B_P(v)$, it holds $B_P(u) = B_P(v)$, and all vertices in $B_P(v)$ have the same label. In Fig. 1 we show examples of two patterns with their orbits.

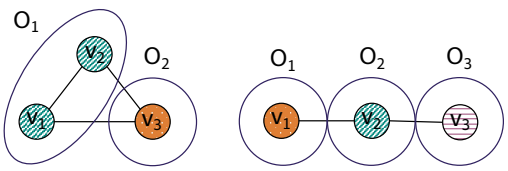


Fig. 1. Examples of patterns and orbits. Colors represent vertex labels, while circles represent orbits. In the pattern on the left, v_1 and v_2 belong to the same orbit O_1. On the right, each vertex belongs to its own orbit.

3.2 Frequent patterns

Among the many measures of frequency for subgraphs [22, 38, 39], we adopt the *minimum node image-based* (MNI) support [10] metric to count the occurrences of the patterns. MNI is *anti-monotonic*: any pattern (e.g., a triangle) has MNI support no larger than any of its subgraphs (e.g., an edge) (see Sect. 4.1), which avoids counter-intuitive results. Computationally, anti-monotonicity enables apriori-like algorithms [62] to prune the pattern space.

Let $G = (V, E)$ be a graph, and let $S \subseteq V$ be a subset of vertices. For any orbit A of any pattern $P \in \mathcal{P}$, let the *image set* $\mathsf{Z}_S(A)$ *of* A *on* S be the subset of S containing all and only the vertices $v \in S$

[1] Our work can easily be extended to handle both self loops and to work on multigraphs. See the work by Ingalalli et al. [27] for mining frequent subgraph patterns on multigraphs.

[2] Our algorithm can also handle *edge-induced* subgraphs, with minor modifications. See Alg. 4 and discussion in Sect. 4.

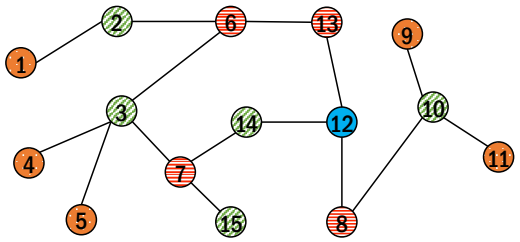
[3] $\mathcal{C}$ depends on k and G but we do not use them in the notation to keep it light.

for which there exists an isomorphism μ from an induced subgraph $G' = (V', E') \in \mathcal{C}$ with $v \in V'$ to P such that $\mu(v) \in A$. Formally,

$$\mathsf{Z}_S(A) \doteq \{v \in S \ : \ \exists \text{ isomorphism } \mu : (V', E') \to P \text{ s.t.} \\ (V', E') \in \mathcal{C} \wedge v \in V' \wedge \mu(v) \in A\} \ . \tag{2}$$

Figure 2a shows an example graph. The pattern on the left of Fig. 1 does not appear in this graph, thus the image sets of all its orbits are empty. For the pattern on the right of Fig. 1, we report the image sets of its orbits in Fig. 2b.

(a) Example graph.



(b) Image sets for the orbits of the pattern in Fig. 1 (right).

Orbit	Image set
O_1	$\{1, 4, 5, 9, 11\}$
O_2	$\{2, 3, 10\}$
O_3	$\{6, 7, 8\}$

Fig. 2. Example graph and image sets for the orbits of the pattern on the right of Fig. 1.

The *orbit frequency* $\mathsf{c}_S(A)$ *of* A *on* S is the ratio between the size of its image set $\mathsf{Z}_S(A)$ and the size of S, i.e.,

$$\mathsf{c}_S(A) \doteq \frac{|\mathsf{Z}_S(A)|}{|S|} \ . \tag{3}$$

The *(relative) MNI-frequency* $\mathsf{f}_S(P)$ *of* $P \in \mathcal{P}$ *on* S is the minimum orbit frequency on S for any orbit of P, i.e.,

$$\mathsf{f}_S(P) \doteq \min\{\mathsf{c}_S(A) \ : \ A \text{ is an orbit of } P\} \ . \tag{4}$$

For example, the MNI-frequency of the pattern on the right of Fig. 1 is, using Fig. 2b and the fact that the example graph in Fig. 2a has 15 vertices, $\min\{5/15, 3/15, 3/15\} = 1/5 = 0.2$.

When dealing with approximations, it is more straightforward to reason about this quantity than about the (absolute) MNI-support (i.e., the minimum size of the image set of any orbit of P).[4]

Given a (large) graph $G = (V, E)$, and a *minimum frequency threshold* $\tau \in (0, 1)$, for any $S \subseteq V$, the set $\mathsf{FP}_S(\tau)$ of *τ-frequent patterns on S* contains all and only the patterns with frequency on S greater than or equal to τ, *together with their frequencies*, i.e.,

$$\mathsf{FP}_S(\tau) \doteq \{(P, \mathsf{f}_S(P)) \ : \ P \in \mathcal{P} \ \wedge \mathsf{f}_S(P) \geq \tau\} \, . \tag{5}$$

The task we are interested in requires finding $\mathsf{FP}_V(\tau)$. Due to the exponential number of candidate patterns, and to the hardness of evaluating the subgraph isomorphisms, finding this collection is challenging. An *approximate solution* $\mathcal{Q}$ is sufficient, in many cases, provided it comes with stringent quality guarantees, such as *(i)* the lack of false negatives, i.e., every pattern in $\mathsf{FP}_V(\tau)$ also appears in $\mathcal{Q}$, and *(ii)* guarantees on the frequency estimation error. MaNIACS, outputs a set $\mathcal{Q}$ with such guarantees (see Thm. 4.7), by sampling a subset of vertices from V, which are then used to approximate the frequency of patterns of increasing size, while exploiting the anti-monotonicity of the frequency measure to prune the search space. To understand the trade-off between the sample size and the accuracy ε of the approximation, we use concepts and results from statistical learning theory [64], described next.

[4]Henceforth, we use "frequency" to refer to the MNI-frequency.

3.3 Empirical VC-dimension and η-samples

We give here the main definitions and results about empirical VC-dimension, tailored to our setting. For a general discussion, see the textbook by Shalev-Shwartz and Ben-David [59, Ch. 6].

A *range space* is a pair $(\mathcal{D}, \mathcal{R})$ where $\mathcal{D}$ is a finite ground set of elements called *points* and $\mathcal{R}$ is a family of subsets of $\mathcal{D}$ called *ranges*. For any $A \subseteq \mathcal{D}$, let the *projection* $\mathsf{P}_{\mathcal{R}}(A)$ *of* $\mathcal{R}$ *on* A be the set $\mathsf{P}_{\mathcal{R}}(A) \doteq \{r \cap A \ : \ r \in \mathcal{R}\} \subseteq 2^A$. When $\mathsf{P}_{\mathcal{R}}(A) = 2^A$, i.e., when the projection contains all the proper and improper subsets of A, then we say that A is *shattered* by $\mathcal{R}$. Given a subset $Y \subseteq \mathcal{D}$, the *empirical Vapnik-Chervonenkis (eVC) dimension* $\mathsf{E}_Y(\mathcal{R})$ *of* $\mathcal{R}$ *on* Y is the size of the *largest shattered subset of* Y [64]. The *VC-dimension* of $\mathcal{R}$ is the empirical VC-dimension of $\mathcal{R}$ on $\mathcal{D}$. For example, let $\mathcal{D}$ be the subset of $\mathbb{Z}$ delimited by two finite integers $a \leq b$, and let

$$\mathcal{R} = \{[c, d] \cap \mathbb{Z} \ : \ a \leq c \leq d \leq b\}$$

be the set of discrete intervals in $\mathcal{D}$. It is easy to see that shattering any set of two elements of $\mathcal{D}$ is easy, while it is impossible to shatter any set $\{e, f, g\}$ of three distinct elements $e < f < g$ from $\mathcal{D}$, as there is no range r in $\mathcal{R}$ such that $r \cap \{e, f, g\} = \{e, g\}$, as any r that contains e and g must also contains f by definition. Thus the VC-dimension of this $(\mathcal{D}, \mathcal{R})$ is 2.

The concept of η-*sample for* $(\mathcal{D}, \mathcal{R})$ is crucial for our work. For $0 < \eta < 1$, a subset $A \subseteq \mathcal{D}$ is an η-*sample for* $(\mathcal{D}, \mathcal{R})$ if it holds

$$\left| \frac{|R|}{|\mathcal{D}|} - \frac{|A \cap R|}{|A|} \right| \leq \eta, \text{ for every } R \in \mathcal{R} \ . \tag{6}$$

Given an η-sample A, we can estimate the relative sizes of any range $R \in \mathcal{R}$ w.r.t. the domain (i.e., the first term on the l.h.s.) with its relative size w.r.t. A (the second term on the l.h.s.), and the estimate is guaranteed to be no more than η-far from its exact value.

Given a sample size s, let $\mathcal{T}$ be a collection of s points sampled from $\mathcal{D}$ independently and uniformly at random (with or without replacement). Knowing an upper bound d to the empirical VC-dimension of $(\mathcal{D}, \mathcal{R})$ on $\mathcal{T}$ allows the computation of an η such that, probabilistically, $\mathcal{T}$ is an η-sample for $(\mathcal{D}, \mathcal{R})$.

Theorem 3.1 (34). *Let $\phi \in (0, 1)$ be an acceptable failure probability. For $(\mathcal{D}, \mathcal{R})$, s, $\mathcal{T}$, and d as above, it holds that, with probability at least $1 - \phi$ (over the choice of $\mathcal{T}$), $\mathcal{T}$ is an η-sample for $(\mathcal{D}, \mathcal{R})$ for*

$$\eta = \sqrt{\frac{c\left(d + \ln \frac{1}{\phi}\right)}{s}}, \tag{7}$$

where c is a universal constant.[5]

When the upper bound d to $\mathsf{E}_{\mathcal{T}}(\mathcal{R})$ is computed from $\mathcal{T}$, the value η from eq. (7) depends only on $\mathcal{T}$ and on ϕ, i.e., it is a *sample-dependent* upper bound to the maximum difference, over all ranges, between the relative sizes of the ranges w.r.t. the sample and the relative sizes w.r.t. the domain, i.e., to the l.h.s. of eq. (6).

3.4 Symbols Reference

Table 1 reports the most important and common symbols used throughout the paper, as a quick reference for the reader.

[5]In our experiments, we follow Löffler and Phillips [35] and use $c = 0.5$.

Table 1. Table of symbols.

	Symbol	Description
Graph	G	The data graph
	V	Set of vertices of G
	E	Set of edges of G
	L	Set of possible labels for the vertices V of G
	L	Labeling function of G
Pattern	P	A graph pattern (a small connected graph)
	k	Maximum size (number of vertices) a pattern can have
	$\mathcal{P}$	Set of all possible patterns (connected graphs of size up to k)
	$\mathcal{C}$	Set of all connected induced subgraphs of G up to size k
	$B_P(v)$	Orbit of vertex v in pattern P (set of vertices linked by automorphism)
	$\mathsf{Z}_V(B_P)$	Image set of orbit B_P on the vertices V (set of vertices in V mapped to B_P by subgraph isomorphism)
	$\mathsf{f}_V(P)$	MNI frequency of pattern P on the graph induced by the set of vertices V
	$\mathsf{FP}_V(\tau)$	Set of patterns with MNI frequency at least τ on V
	$\mathcal{F}_i$	Set of patterns of size i in $\mathsf{FP}_V(\tau)$
VC-dimension	$\mathcal{D}$	A finite ground set of elements (corresponds to V in our case)
	$\mathcal{R}$	A family of subsets of $\mathcal{D}$ (corresponds to $\mathsf{Z}_V(B_P)$ in our case)
	$\mathsf{E}_{\mathcal{T}}(\mathcal{R})$	The empirical Vapnik-Chervonenkis (eVC) dimension of $\mathcal{R}$ on $\mathcal{T} \subseteq \mathcal{D}$ (the size of the largest shattered subset of $\mathcal{T}$)
	$\mathcal{R}_i$	Set of image sets on V of all the orbits of patterns in $\mathcal{F}_i$

4 APPROXIMATE FSPM

We now present MaNIACS, our algorithm for mining high-quality approximations to $\mathsf{FP}_V(\tau)$ through sampling. At a very high level, MaNIACS draws a sample S from V and uses the orbit frequencies, the frequency of the patterns on S, and the eVC-dimension of appropriately-designed range spaces, to derive the output quality guarantees. MaNIACS does *not* consider the subgraph of G induced by S. Rather, it always considers the whole graph G when checking the existence of isomorphisms from patterns to induced subgraphs of G. The sample is instead used to compute $\mathsf{f}_S(P)$ as an estimation of $\mathsf{f}_V(P)$, as obtaining the former is faster given that $|S| \ll |V|$.

The following fact is at the basis of MaNIACS, and it is immediate from the definition of MNI-frequency (see eq. (4)).

Fact 1. *Given $P \in \mathcal{P}$ and $S \subseteq V$, let ε be such that it holds*

$$|\mathsf{c}_S(A) - \mathsf{c}_V(A)| \leq \varepsilon, \text{ for every orbit } A \text{ of } P. \tag{8}$$

Then it must be $|\mathsf{f}_S(P) - \mathsf{f}_V(P)| \leq \varepsilon$.

This corollary suggests how to identify patterns that cannot be frequent, and that can therefore be pruned.

Corollary 4.1. *Let P, S, and ε as in Fact 1. If it holds $\mathsf{f}_S(P) < \tau - \varepsilon$, then it must be $\mathsf{f}_V(P) < \tau$, i.e., $P \notin \mathsf{FP}_V(\tau)$.*

Statistical learning theory gives us the tools to compute values ε which satisfy the condition from eq. (8). Given the exponential number of patterns, it would be unfeasible to compute $\mathsf{f}_S(P)$ for every $P \in \mathcal{P}$. Thus, we rely on properties of the orbit frequency and of the MNI-frequency

functions (see Sect. 4.1) to *prune the space of patterns*, in an *apriori-like way*, and therefore to avoid computing the frequencies of orbits whose pattern is not in $\mathsf{FP}_V(\tau)$.

4.1 Search space and frequency properties

We now define a partial order between patterns in $\mathcal{P}$: we say that P'' is a child of P' if *(i)* P'' has exactly one more vertex than P'; and *(ii)* there exists an isomorphism between P' and some induced subgraph of P''. When P'' is a child of P' we say that P' is a *parent* of P''. A pattern may have multiple parents, while patterns with a single vertex have no parent. The anti-monotone property of the MNI-frequency gives the following fact.

Fact 2 ([10]). *For any pattern $P \in \mathcal{P}$, any pattern $Q \in \mathcal{P}$ that is a child of P, and any $S \subseteq V$, it holds that $\mathsf{f}_S(Q) \leq \mathsf{f}_S(P)$.*

We define a similar parent-child relation between pairs of orbits. Given two distinct patterns $P, Q \in \mathcal{P}$ and two orbits B_P and B_Q of each respectively, we say that B_Q is the *child* of B_P iff Q is a child of P and there is a subgraph isomorphism from P to Q that maps at least one vertex of B_P to a vertex of B_Q. When B_Q is the child of B_P, we say that B_P is the *parent* of B_Q, and denote all the children of B_P as $\mathsf{C}(B_P)$. Figure 3 shows some examples of the parent-child relationships. An orbit can have multiple parents, and the orbits of patterns containing a single vertex have no parent. Our algorithm leverages the following important property of this relationship, which is immediate from the definition, to quickly prune the search space of patterns.

Fact 3. *Let A and D be two orbits such that D is a child of A. Then, for any $S \subseteq V$, it holds $\mathsf{Z}_S(D) \subseteq \mathsf{Z}_S(A)$.*

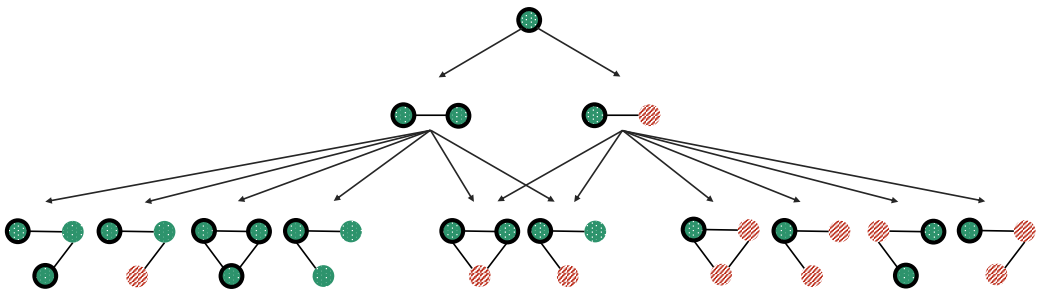


Fig. 3. Examples of parent-child relations for orbits (labels represented as colors). We represent each orbit by using its pattern with the vertices of the orbit in a thicker border.

4.2 The number of labeled patterns

The number of *unlabeled* (or, equivalently, single-labeled, when $|L| = 1$) connected patterns with k vertices is well known,[6] as is the number of orbits in this case [50, Table 1]. Conversely, to the best of our knowledge, the number of *labeled* patterns and of *labeled orbits* with m labels and k vertices is *not known*. We show how to compute the number of labeled *patterns* by using the Redfield–Pólya (enumeration) theorem [46] (Thm. 4.2). A similar analysis for the number of *orbits* in the presence of multiple labels seems quite complicated, due to the symmetries that are broken in different ways

[6] http://oeis.org/A001349

when assigning different labels to vertices belonging to the same unlabeled orbit. Nevertheless, it remains a very interesting direction for future work.

Let $\{p_{1,k} = (V_{1,k}, E_{1,k}), \ldots, p_{u_k,k} = (V_{u_k,k}, E_{u_k,k})\}$ be the set of the u_k *unlabeled* patterns of size k, with $p_{u_k,k}$ being the clique, and all other patterns given arbitrarily distinct values for i, $1 \leq i < u_k$. Our goal is to compute the number of different possible assignments of the m labels to the vertices of $p_{i,k}$, $1 \leq i \leq u_k$. To do so, we use the following celebrated combinatorics result.

Theorem 4.2 (Redfield–Pólya theorem [46]). *Let X be a finite set and Z be a group of permutations of X. Let Y be a finite set with $|Y| = q$, and Y^X be the set of functions from Y to X (the group Z acts on Y^X). The number of* group orbits[7] *under Z is*

$$\left|Y^X/Z\right| = \frac{1}{|Z|} \sum_{g \in Z} q^{\mathrm{c}(g)}, \tag{9}$$

where $\mathrm{c}(g)$ is the number of (permutation) cycles in the permutation $g \in Z$.[8]

We use this result as follows. We fix $Y = L$ and $q = m$. For each unlabeled pattern $p_{i,k}$, we set $X = V_{i,k}$, and Z as the *automorphism group of* X, i.e., the set $\mathrm{Aut}(p_{i,k})$ of graph automorphisms of $p_{i,k}$, seen as permutations on X, with the composition operator. The quantity $\left|Y^X/Z\right|$ is then the number of different ways of labeling the vertices of $p_{i,k}$ by using the labels in L, up to automorphisms. Therefore, it is the number of labeled patterns that, up to the labels, are isomorphic to $p_{i,k}$. To compute this quantity by using eq. (9), we therefore need to analyze each $g \in Z$ to obtain $\mathrm{c}(g)$.

For example, for the *clique* $p_{u_k,k}$, Z contains all $k!$ permutations. The number of permutations of k objects with i cycles is known as the *unsigned Stirling number of the first kind* $\left[{k \atop i}\right]$. For the clique, we can then rewrite (9) as

$$\left|Y^X/Z\right| = \frac{1}{k!} \sum_{i=1}^{k} \left[{k \atop i}\right] m^i \quad .^9$$

The value of this expression may not be immediately evident, but it can be derived by using the Stars and Bars theorem [21], which tells us that the number of distinct ways to arrange k indistinguishable objects (the vertices) into m distinguishable boxes (the labels) is $\binom{k+m-1}{k}$. Thus, the number of labeled cliques of size k that can be created with m labels is

$$\mathsf{patterns}(u_k, k, m) \doteq \binom{k+m-1}{k} \ . \tag{10}$$

For each non-clique unlabeled pattern $p_{i,k}$, we can explicitly identify the permutations in $\mathrm{Aut}(p_{i,k})$ (i.e., the automorphisms), and for each permutation $g \in \mathrm{Aut}(p_{i,k})$, compute the number $\mathrm{c}(g)$ of cycles of g. To obtain this information, we analyze the *rotational and reflection symmetries* of the (graphical representation of the) pattern $p_{i,k}$, as follows. In Fig. 7 in App. A.1 (inspired by [36, Fig. 1]) we show, for $k \in \{3, 4, 5\}$, each *unlabeled* (i.e., single-labeled) pattern, where nodes of the same color belong to the same orbit (in the *unlabeled* pattern). A pattern has a *d-degree rotational symmetry* if, by rotating its graphical representation by d degrees counterclockwise, we obtain the same graphical representation. A pattern has a *reflection symmetry* around a certain axis if, by reflecting its graphical representation around the axis, we obtain the same graphical representation.

[7] The *group orbit* of $x \in X$, not to be confused with eq. (1), is $\{\, y \in X \mid \exists g \in Z \,.\, y = g * x \}$, where $*$ is the group action.
[8] A *permutation cycle* is a subset of a permutation whose elements are permuted among themselves, and not with elements not in the cycle. The *length of a cycle* is the number of elements it contains. For example, the identity permutation on z elements has z cycles of length 1, while a "circular shift" permutation of z elements has one cycle of length z.
[9] For *signed* Stirling numbers of the first kind, this expression would be equal to $\binom{m}{k}$, but it is *not* the case.

Table 2. Number of labeled patterns with m labels, for each unlabeled pattern as in Fig. 7 in App. A.1.

Pattern	patterns(i,k,m)	Explanation
1	$(m^3+m^2)/2$	The identity has three cycles, the reflection around the v-axis has two.
2	$(m^3+3m^2+2m)/6$	The pattern is a clique, so we use eq. (10).
3	$(m^4+m^2)/2$	The identity has four cycles, the reflection around the v-axis has two.
4	$(m^4+3m^3+2m^2)/6$	The identity has four cycles, the three reflections along the axis defined by each edge have three cycles each, and the two rotations by 120° and 240° have two each.
5	$(m^4+2m^3+3m^2+2m)/8$	The identity has four cycles, the two reflections along the diagonals have three each, the two reflections around the h- and v-axis have two each, the rotation by 180° has two, the rotations by 90° and 270° one each.
6	$(m^4+m^3)/2$	The identity has four cycles, the reflection along the h-axis has three.
7	$(m^4+2m^3+m^2)/4$	The identity has four cycles, the two reflections along the diagonals have three each, and the rotation by 180° has two.
8	$(m^4+6m^3+11m^2+6m)/24$	The pattern is a clique, so we use eq. (10).
9	$(m^5+m^3)/2$	The identity has five cycles, the reflection around the v-axis has three.
10	$(m^5+m^4)/2$	The identity has five cycles, the reflection around the h-axis has four.
11	$(m^5+6m^4+11m^3+6m^2)/24$	For each permutation g over four elements, there is a permutation g' of the pattern vertices obtained by applying g to the black vertices (the white one is fixed). It holds $c(g') = c(g)+1$.
12	$(m^5+m^3)/2$	The identity has five cycles, the reflection around the v-axis has three.
13	$(m^5+m^4)/2$	The identity has five cycles, the reflection along the h-axis has four.
14	$(m^5+2m^4+m^3)/4$	The identity has five cycles, the two permutations that only permute the white vertices or only the black vertices have four each, the permutation that permutes both the white and the black vertices (in pairs) has three.
15	$(m^5+5m^3+4m)/10$	The identity has five cycles, the five reflections along each of the axis going from a vertex to the mid point of the opposite edge have three cycles each, and the other four rotations by multiples of 72° have one cycle each.
16	$(m^5+m^4)/2$	The identity has five cycles, the reflection along the h-axis has four.
17	$(m^5+m^4)/2$	Same as #16.
18	$(m^5+2m^4+3m^3+2m^2)/8$	The identity has five cycles, the two permutations that only swap the two top or the two bottom black vertices (in pairs) have four cycles each, the two reflections along the h- or v-axis have three each, the rotation by 180° has three. There are then two permutations obtained by first rotating by 180° and swapping, respectively, the resulting two top or the resulting two bottom nodes, and each of these permutation has two cycles.
19	$(m^5+m^4)/2$	The identity has five cycles, the reflection along the h-axis has four.
20	$(m^5+4m^4+5m^3+2m^2)/12$	For each permutation g of three elements, there are two permutations g' and g'' of the pattern vertices, with g' applying g to the black vertices and keeping the white ones fixed, and g'' applying g to the black vertices and also permuting the two white vertices. It holds $c(g') = c(g)+2$ and $c(g'') = c(g)+1$.
21	$(m^5+m^3)/2$	The identity has five cycles, the reflection around the v-axis has three.
22	$(m^5+4m^4+5m^3+2m^2)/12$	Same as #20.
23	$(m^5+3m^4+2m^3)/6$	For each permutation g of three elements, consider the permutation g' of the pattern vertices obtained by applying g to the white vertices (the others are fixed). It holds $c(g') = c(g)+2$.
24	$(m^5+m^3)/2$	The identity has five cycles, the reflection around the v-axis has three.
25	$(m^5+2m^4+m^3)/4$	The identity has five cycles, the two permutations that only permute the white vertices or only the grey vertices have four each, the permutation that permutes both white and grey vertices (in pairs) has three.
26	$(m^5+2m^4+m^3)/4$	Same as #25.
27	$(m^5+2m^4+3m^3+2m^2)/8$	Similar to #5, but each permutation has an additional cycle for the white vertex.
28	$(m^5+4m^4+5m^3+2m^2)/12$	Same as #20.
29	$\binom{m+4}{5}$	The pattern is a clique, so we use eq. (10).

Once we have the necessary information about $\mathrm{Aut}(p_{i,k})$ and its elements, we can use eq. (9) to compute the number of distinct labeled patterns that can be generated by assigning one of the m labels to each vertex of $p_{i,k}$, which we denote with $\mathsf{patterns}(i,k,m)$. Table 2 reports the equations to compute the number of labeled patterns with m labels for $k \in \{3,4,5\}$, for each *unlabeled* (i.e., single-labeled) pattern numbered as in Fig. 7. The 'Explanation' column of Table 2 comments on how we obtain the formula on the second column. We use 'h-axis' and 'v-axis' to refer to the *middle* (i.e., center) horizontal and vertical axis of the graphical representation of the pattern. Formulas similar to the ones in Table 2 can be derived for higher values of k, with a *careful* analysis of each unlabeled pattern and its orbits.

By combining these quantities with eq. (10), we get the number of distinct labeled patterns of size k using m labels as

$$\mathsf{patterns}(k,m) = \sum_{i=1}^{u_k} \mathsf{patterns}(i,k,m) \ .$$

We can use the formulas in Table 2 to explicitly obtain the total number of labeled patterns of size k with m labels. For example, for $k=3$, the number of labeled patterns is

$$\mathsf{patterns}(3,m) = \frac{2m^3+3m^2+m}{3} \ .$$

For $k=4$, it is

$$\mathsf{patterns}(4,m) = \frac{19m^4+24m^3+23m^2+6m}{12},$$

and for $k=5$, it is

$$\mathsf{patterns}(5,m) = \frac{91m^5+95m^4+95m^3+25m^2+9m}{15} \ .$$

Studying properties of the labeled pattern search space beyond its size, and any property of the labeled orbit search space, could lead to interesting algorithmic developments, and it is a promising direction for future theoretical research.

4.3 The frequent patterns range spaces

We now define an appropriate set of range spaces and show how to compute bounds to their eVC-dimensions. Given $\tau \in (0,1]$, let $\mathcal{F}_i$ for $i=1,\ldots,k$ be the set of patterns with i vertices that belong to $\mathsf{FP}_V(\tau)$.[10] Let $\mathcal{R}_i$ be the set whose elements are the image sets on V of all the orbits of all the patterns in $\mathcal{F}_i$, $1 \le i \le k$, i.e.,

$$\mathcal{R}_i \doteq \{\mathsf{Z}_V(A) \ : \ A \text{ is an orbit of } P \in \mathcal{F}_i\} \ .$$

Henceforth, we use the range spaces $(V,\mathcal{R}_i)$, $1 \le i \le k$. The relevance of these range spaces is clear when looking at Equation (3). We now show novel results to upper bound the eVC-dimension of $(V,\mathcal{R}_i)$ on any $S \subseteq V$. MaNIACS computes such bounds to derive the approximation guarantees and to prune the search space.

The following two results are presented in the most general form because they hold for *any* range space. We later tailor them for our case, and discuss how to compute the presented bounds efficiently.

Lemma 4.3. *Let $(\mathcal{D},\mathcal{R})$ be a range space, and let $\mathcal{T} \subseteq \mathcal{D}$. Consider the* set $\mathcal{R}_\mathcal{T} \doteq \{\mathcal{T} \cap R \ : \ R \in \mathcal{R}\}$. *Let g^* be the maximum g such that $\mathcal{T}$ contains at least g points each appearing in at least 2^{g-1} sets from $\mathcal{R}_\mathcal{T}$. If, for at least one set B of such g^* points, there exists a set $Z_B \in \mathcal{R}_\mathcal{T}$ such that $Z_B \supseteq B$, then $\mathsf{E}_\mathcal{T}(\mathcal{R})$ is at most g^*, otherwise it is at most g^*-1.*

[10] The sets $\mathcal{F}_i$, $1 \le i \le k$, depend on G and on τ, but the notation does not reflect these dependencies to keep it light.

Proof. For a set A of $|A| = g$ to be shattered, it is necessary that each $a \in A$ belongs to at least 2^{g-1} distinct sets in $\mathcal{R}_{\mathcal{T}}$, as a belongs to these many non-empty subsets of A. Additionally, there must be a set $Z_A \in \mathcal{R}_{\mathcal{T}}$ that contains the whole A, i.e., such that $Z_A \supseteq A$. The quantity g^* is the maximum for which both these two conditions hold. If only the first one holds, then there is no set of size g^* that can be shattered, thus the eVC-dimension is at most $g^* - 1$. □

Lemma 4.4. *Let $(\mathcal{D}, \mathcal{R})$ and $\mathcal{T}$ as in Lemma 4.3. Let $R_1, \ldots, R_{|\mathcal{R}|}$ be a labeling of the ranges in $\mathcal{R}$ such that $|R_w \cap \mathcal{T}| \geq |R_u \cap \mathcal{T}|$ for $1 \leq w < u \leq |\mathcal{R}|$. Let $(a_j)_{j=1}^{\ell}$ be a non-increasing sequence of $\ell \geq |\mathcal{R}|$ naturals such that $a_j \geq \left|R_j \cap \mathcal{T}\right|$ for $1 \leq j < |\mathcal{R}|$ and $a_j \geq \left|R_{|\mathcal{R}|} \cap \mathcal{T}\right|$ for $|\mathcal{R}| \leq j \leq \ell$.*

Let h^* be the maximum natural h such that, for every $0 \leq j < h$, if we let $c_j = \sum_{y=0}^{j} \binom{h}{y}$,[11] it holds $a_{c_j} \geq h - j$. Then, $\mathsf{E}_{\mathcal{T}}(\mathcal{R}) \leq h^*$.

Proof. Let $z = \mathsf{E}_{\mathcal{T}}(\mathcal{R})$. Then there is a set $A \subseteq \mathcal{T}$ with $|A| = z$ that is shattered by $\mathcal{R}$. For a set A with $|A| = z$ to be shattered by $\mathcal{R}$, there must be, for every $0 \leq i < z$, $\binom{z}{i}$ distinct ranges $H_{i,1}, \ldots, H_{i,\binom{z}{i}} \in \mathcal{R}$ such that $\left|H_{i,j} \cap A\right| = z - i$, as A has $\binom{z}{i}$ subsets of size $z - i$. It must then also hold that $\left|H_{i,j} \cap \mathcal{T}\right| \geq z - i$.

If $\ell = |\mathcal{R}|$ and $a_j = \left|R_j \cap \mathcal{T}\right|$ for every $1 \leq j \leq |\mathcal{R}|$, it follows from the definition of h^* that it must be $z \leq h^*$. For a generic sequence $(a_j)_{j=1}^{\ell}$, the thesis follows from the fact that the value h^* computed on this generic sequence cannot be smaller than the value h^* computed on the specific sequence for which $\ell = |\mathcal{R}|$ and $a_j = \left|R_j \cap \mathcal{T}\right|$ for every $1 \leq j \leq |\mathcal{R}|$. □

While the lemma above may seem complex at first, its proof is essentially an application of the pigeonhole principle, and the procedure to compute the bound h^* from the sequence $(a_i)_{i=1}^{\ell}$ is straightforward, as we discuss in Sect. 4.4.

For $\lambda \in L$, let $\mathcal{R}_{i,\lambda}$ be the subset of $\mathcal{R}_i$ containing all and only the image sets of the orbits whose vertices have all label λ. Clearly each $(V, \mathcal{R}_{i,\lambda})$ is a range space. The following result ties the empirical VC-dimension of these range spaces to that of $(V, \mathcal{R}_i)$.

Lemma 4.5. *For any $S \subseteq V$, it holds $\mathsf{E}_S(\mathcal{R}_i) = \max_{\lambda \in L} \mathsf{E}_S(\mathcal{R}_{i,\lambda})$.*

Lemma 4.5 is an immediate corollary of the following result.

Lemma 4.6. *No $S \subseteq V$ containing vertices with different labels can be shattered by $\mathcal{R}_i$.*

Proof. The statement is immediate from the definition of image set (see Equation (2)). For any orbit A, its image set $\mathsf{Z}_V(A)$ on V only contains vertices with the same label, thus there would be no range in $\mathcal{R}_i$ that would contain, for example, the whole S, thus $S \notin \mathsf{P}_{\mathcal{R}_i}(S)$, which implies that S cannot be shattered by $\mathcal{R}_i$. □

Lemma 4.5 says that $\mathsf{E}_S(\mathcal{R}_i)$ is, in some sense, *independent from the number $|L|$ of labels*, which is surprising, from a theoretical point of view. MaNIACS computes upper bounds to $\mathsf{E}_S(\mathcal{R}_{i,\lambda})$, $\lambda \in L$, using Lemmas 4.3 and 4.4, and then leverages Lemma 4.5 to derive an upper bound to $\mathsf{E}_S(\mathcal{R}_i)$.

4.4 MaNIACS, the algorithm

The intuition behind MaNIACS is the following. It creates a sample S by drawing vertices independently and uniformly at random without replacement from V. Then it computes from S a value ε_i such that S is an ε_i-sample for the range space $(V, \mathcal{R}_i)$, for $1 \leq i \leq k$. For such an ε_i, thanks to eq. (7) and eq. (3), it holds, for any $P \in \mathcal{F}_i$, that

$$\mathsf{c}_S(A) \geq \mathsf{c}_V(A) - \varepsilon_i \geq \tau - \varepsilon_i \text{ for any orbit } A \text{ of } P,$$

[11] We define $\binom{q}{0} = 1$ for any q.

which implies that $\mathrm{f}_S(P) \geq \tau - \varepsilon_i$. This lower bound to the possible frequency of $P \in \mathcal{F}_i \subseteq \mathrm{FP}_V(\tau)$ on S allows us to determine which patterns may actually belong to $\mathrm{FP}_V(\tau)$ and which ones cannot.

Unfortunately, the sets $\mathcal{F}_i$, $1 \leq i \leq k$, are *not known a priori*, as if they were, we could use them to exactly obtain $\mathrm{FP}_V(\tau)$. MaNIACS therefore computes a *superset* $\mathcal{H}_i$ of each set. It uses the sizes of the image sets on S of the orbits of the patterns in $\mathcal{H}_i$ to compute an upper bound to the eVC-dimension of $(V, \mathcal{R}_i)$, thanks to Lemmas 4.3 to 4.5. By plugging this upper bound in eq. (7), it gets a value ε_i such that S is (probabilistically) an ε_i-sample for $\mathcal{F}_i$.

Algorithm 1: MaNIACS

Input: Graph $G = (V, E)$, maximum pattern size k, frequency threshold τ, sample size s, failure probability δ
Output: A set $\mathcal{Q}$ with the properties from Thm. 4.7

1 $S \leftarrow$ `drawSample(`V, s`)`
2 $\mathcal{Q} \leftarrow \emptyset$; $i \leftarrow 1$
3 $\mathcal{H}_1 \leftarrow \{P \in \mathcal{P} \ : \ P$ has a single vertex$\}$
4 **while** $i \leq k$ **and** $\mathcal{H}_i \neq \emptyset$ **do**
5 $\mathcal{Z}_i \leftarrow$ `getImageSets(`$\mathcal{H}_i, S, \tau$`)`
6 **do**
7 $b_i^* \leftarrow$ `getEVCBound(`$\mathcal{Z}_i$`)`
8 $\varepsilon_i \leftarrow$ `getEpsilon(`$b_i^*, \delta/k$`)`
9 $\mathcal{H}'_i \leftarrow \mathcal{H}_i$
10 $\mathcal{H}_i \leftarrow \{P \in \mathcal{H}_i \ : \ \mathrm{f}_S(P) \geq \tau - \varepsilon_i\}$
11 **while** $\mathcal{H}'_i \neq \mathcal{H}_i$ **and** $\mathcal{H}_i \neq \emptyset$
12 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(P, \mathrm{f}_S(P), \varepsilon_i) \ : \ P \in \mathcal{H}_i\}$
13 **if** $i < k$ **then** $\mathcal{H}_{i+1} \leftarrow$ `createChildren(`$\mathcal{H}_i, \mathcal{Z}_i$`)`
14 $i \leftarrow i + 1$
15 **return** $\mathcal{Q}$

We first present a simplified version of MaNIACS (pseudocode in Alg. 1), and discuss more details in Sect. 4.4.2. MaNIACS takes as input a graph $G = (V, E)$, a maximum pattern size k, a minimum frequency threshold τ, a sample size s, and an acceptable failure probability δ. It outputs a set $\mathcal{Q}$ with the following properties.

Theorem 4.7. *With probability at least $1 - \delta$ over the choice of S, the output $\mathcal{Q}$ of MaNIACS contains a triplet $(P, \mathrm{f}_S(P), \varepsilon_P)$ for every $P \in \mathrm{FP}_V(\tau)$ such that $|\mathrm{f}_S(P) - \mathrm{f}_V(P)| \leq \varepsilon_P$.*

Proof. For $1 \leq i \leq k$, let η_i be the value η computed as in Thm. 3.1 for $\phi = \delta/k$, $(\mathcal{D}, \mathcal{R}) = (V, \mathcal{R}_i)$, $\mathcal{T}$ chosen as S on line 1, and d being the eVC-dimension of $(V, \mathcal{R}_i)$ on S. It follows from Thm. 3.1 and an application of the union bound over the k sets (hence the use of δ/k), that, with probability at least $1 - \delta$, it holds that S is, simultaneously, an η_i-sample for $(V, \mathcal{R}_i)$ for every $1 \leq i \leq k$. Assume for the rest of the proof that that is the case.

We show inductively that, at the end of every iteration of the "main" loop of MaNIACS (lines 4–14), it holds that

(1) $\mathcal{Q}$ contains a triplet $(P, \mathrm{f}_S(P), \varepsilon_i)$ for each $P \in \mathcal{F}_i$, and the triplet is such that

$$|\mathrm{f}_V(P) - \mathrm{f}_S(P)| \leq \varepsilon_i \ ;$$

(2) $\mathcal{F}_i \subseteq \mathcal{H}_i$, for $i \leq k$.

At the beginning of the first iteration, i.e., for $i = 1$, it obviously holds $\mathcal{F}_1 \subseteq \mathcal{H}_1$ from the definition of $\mathcal{H}_1$ (line 3). Thus, at the first iteration of the do-while loop on lines 6–11, the value ε_1 computed on line 8 using Thm. 3.1 is not smaller than η_1, because b_1^* is an upper bound to the eVC-dimension of $(V, \mathcal{R}_1)$ on S, thanks to Lemmas 4.3 to 4.5, and the value η on the l.h.s. of eq. (7) is monotonically increasing with the value d used on the r.h.s. of the same equation. It then follows, from this fact and from Corol. 4.1, that no pattern $P \in \mathsf{FP}_\tau(V)$ may have $\mathsf{f}_S(P) < \tau - \varepsilon_1$, therefore the refinement of $\mathcal{H}_1$ on line 10 is such that it still holds $\mathcal{F}_1 \subseteq \mathcal{H}_1$ at the end of the first iteration of the do-while loop. Following the same reasoning, one can show that both this condition and the fact that $\varepsilon_1 \geq \eta_1$ hold throughout every iteration of the do-while loop.

The solution set $\mathcal{Q}$, updated on line 12, therefore contains, among others, a triplet for every pattern $P \in \mathsf{FP}_\tau(V)$. The properties from the thesis hold because of this fact, and because $\varepsilon_1 \geq \eta_1$, thus completing the base case for point (1) in the list above. Point (2), i.e., that $\mathcal{F}_2 \subseteq \mathcal{H}_2$, then follows from the anti-monotone property of the MNI-frequency (Fact 2).

Assume now that points (1) and (2) hold at every iteration of the while loop from $i = 1, \ldots, i^* < k$. The proof that they hold at the end of iteration $i^* + 1$ follows the same reasoning as above. □

The algorithm starts by initializing the empty set $\mathcal{Q}$, which will contain the output (line 2) and by creating the sample $S = \{v_1, \ldots, v_s\}$ of s vertices by drawing them independently and uniformly at random from V (line 1).

Algorithm 2: getImageSets

```
   Input: Set of patterns H_i, sample S, frequency threshold τ
   Output: The image sets Z_i of the patterns in H_i
1  Z_i ← ∅
2  foreach P ∈ H_i do
3      foreach orbit A of P do
4          n ← a vertex of P in A
5          S_A ← vertices in S with the label of A
6          Z_S(A) ← ∅, remain ← |S_A|
7          foreach v ∈ S_A do
8              M ← ∅; M[n] ← v
9              if existsIsomorphism(P, M) then
10                 Z_S(A) ← Z_S(A) ∪ {v}
11             remain ← remain − 1
12             if (remain + |Z_S(A)|)/|S| < τ − ε_i then
13                 prune P and go to next pattern
14         Z_i ← Z_i ∪ {Z_S(A)}
15 return Z_i
```

MaNIACS keeps, for every $1 \leq i \leq k$, a superset $\mathcal{H}_i$ of the set $\mathcal{F}_i$. The first such superset $\mathcal{H}_1$ is initialized to contain every pattern of a single vertex (line 3). The algorithm then enters a loop (lines 4–14) which is repeated until i is greater than k or until $\mathcal{H}_i$ is empty. At every iteration, MaNIACS first calls `getImageSets` to obtain the collection $\mathcal{Z}_i$ of the image sets $\mathsf{Z}_S(A)$ on S of every orbit A of every pattern $P \in \mathcal{H}_i$ (pseudocode in Alg. 2). Each set $\mathsf{Z}_S(A)$ is obtained by running an existence query (pseudocode in Alg. 4 in App. A.2) for each vertex v in the sub-sample S_A, to determine whether v belongs to at least one subgraph isomorphic to P. The existence query is a recursive function that incrementally builds a dictionary M, by inserting, at each iteration, a new

candidate match from a pattern vertex to a graph vertex. If a match can be found for each vertex of the pattern, the query returns true, and v is inserted into $\mathsf{Z}_S(A)$. A match z for a pattern vertex u is added to M only if it is consistent with the matches already in M, i.e., if the pattern vertices already matched and u are connected in the same way as the graph vertices mapped to them. If we wish instead to find the edge-induced subgraph isomorphic to P, we just need to modify this consistency check.

MaNIACS then enters a do-while loop (lines 6–11), whose dual purpose is to compute an ε_i such that S is a ε_i-sample for $(V, \mathcal{R}_i)$, and to iteratively refine $\mathcal{H}_i$ as a superset of $\mathcal{F}_i$. To compute ε_i, we first need an upper bound to the eVC-dimension of $(V, \mathcal{R}_i)$ on S (line 7). This bound is computed by the `getEVCBound` function (pseudocode in Alg. 3). For each label $\lambda \in L$, let D_λ be the subset of $\mathcal{Z}_i$ containing the distinct sets associated to orbits of vertices with label λ (line 2 of Alg. 3). First, it computes g^*_λ from Lemma 4.3 (lines 3–7), by taking into account the number of image sets in D_λ in which each vertex $v \in S$ appears. Then, it computes the value h^*_λ from Lemma 4.4 (lines 8–14). The value b^*_i returned by `getEVCBound` is the *maximum*, over $\lambda \in L$, of $\min\{h^*_\lambda, g^*_\lambda\}$.

Algorithm 3: `getEVCBound`

Input: Bag $\mathcal{Z}_i$ of image sets $\mathsf{Z}_A(S)$, $\forall$ orbit A of each pattern in $\mathcal{H}_i$
Output: A value $b^*_i \geq \mathsf{E}_S(\mathcal{R}_i)$

1 **foreach** $\lambda \in L$ **do**
2 $\quad D_\lambda \leftarrow$ set of image sets in $\mathcal{Z}_i$ of orbits of vertices with label λ
3 $\quad M \leftarrow |S|$-vector with element $(v, |\{Z \in D_\lambda : v \in Z\}|), \forall v \in S$
4 $\quad$ sort M in decreasing order of the 2^{nd} component
$\quad$ `// Denote with` (v_i, q_i) `the` i`-th element of` M
5 $\quad g^*_\lambda \leftarrow \max\{g \ : \ v_g \geq 2^{g-1}\}$
6 $\quad \gamma \leftarrow \max\{i \ : \ v_i > 2^{g^*_\lambda - 1}\}$
7 $\quad$ **if** $\nexists Q \subseteq \{v_1, \ldots, v_\gamma\}, |Q| = g^*_\lambda$, *s.t.* $\exists Z \in D_\lambda$ *s.t.* $Q \subseteq Z$ **then** $g^*_\lambda \leftarrow g^*_\lambda - 1$
8 $\quad N \leftarrow |D_\lambda|$-vector with element $|Z|, \forall Z \in D_\lambda$
9 $\quad$ sort N in decreasing order
$\quad$ `// Denote with` a_i `the` i`-th element of` N
10 $\quad h^*_\lambda \leftarrow \min\{a_1, \lfloor \log_2(|D_\lambda| + 1) \rfloor\}$
11 $\quad$ **while** $h^*_\lambda > 1$ **do**
12 $\quad\quad$ **foreach** $j \in \{0, \ldots, h^*_\lambda - 1\}$ **do** $c_j \leftarrow \sum_{z=0}^{j} \binom{h^*_\lambda}{z}$
13 $\quad\quad$ **if** $\nexists j \in \{0, \ldots, h^*_\lambda - 1\}$ *s.t.* $a_{c_j} < h^*_\lambda - j$ **then break**
14 $\quad\quad$ **else** $h^*_\lambda \leftarrow h^*_\lambda - 1$
15 **return** $\max_{\lambda \in L} \min\{g^*_\lambda, h^*_\lambda\}$

MaNIACS uses b^*_i in eq. (7) together with $\eta = \delta/k$ to obtain ε_i (line 8 of Alg. 1). The value ε_i is used to refine $\mathcal{H}_i$ by removing from it any pattern whose frequency in S is lower than $\tau - \varepsilon_i$ (line 10), as they cannot belong to $\mathsf{FP}_V(\tau)$ (see proof of Thm. 4.7). The frequencies can be obtained from $\mathcal{Z}_i$. This refinement process is iterated until no more patterns can be pruned, i.e., $\mathcal{H}'_i = \mathcal{H}_i$, or $\mathcal{H}_i$ becomes empty (line 11). At this point, the patterns still in $\mathcal{H}_i$ are added to the output set $\mathcal{Q}$, together with their frequencies on S and ε_i (line 12). If $i < k$, MaNIACS creates the set $\mathcal{H}_{i+1}$ to contain the patterns on $i + 1$ vertices whose parents are *all* in $\mathcal{H}_i$, by calling the function `createChildren` (line 13). Thanks to Fact 2, this requirement ensures that $\mathcal{H}_i$ is the smallest superset of $\mathcal{F}_i$ that can be obtained on the basis of the currently available information. At this point, the current iteration

of the while loop is completed. When the loop condition (line 4) is no longer satisfied, the algorithm returns the set Q (line 15).

4.4.1 Generating the next set of patterns. MaNIACS takes an apriori-like, level-wise approach that explores a subset of the "level" i of the pattern search space containing the patterns on i vertices, after having explored and pruned the level $i-1$. This subset is generated by the `createChildren` function on the basis of the non-pruned patterns at level $i-1$. In particular, this function extends each non-pruned pattern in the level $i-1$, by adding an edge in every possible position. As this procedure may generate the same pattern multiple times (a pattern can have multiple parents), we identify the canonical form of each pattern generated [30] and prune duplicate patterns. For each distinct extension, we need to compute its orbits, in order to compute their image sets (`getImageSets` function from Alg. 1). The generation of the orbits and patterns in MaNIACS follows steps similar to the procedure by Melckenbeeck et al. [37], adapted to take into consideration the fact that we are working with labeled graphs.

4.4.2 Additional pruning. An efficient pattern mining algorithm must take any chance for pruning the search space. This requirement is particularly important when dealing with subgraphs, because computing the collection $\mathcal{Z}_i$ (line 5) of image sets of the orbits of a pattern $P \in \mathcal{H}_i$ is particularly expensive. We now describe how MaNIACS can prune as much as possible, as early as possible, without any effect on its quality guarantees.

Before delving into pruning, we comment on the computation of the set $\mathrm{Z}_S(A)$ for an orbit A of a pattern $P \in \mathcal{H}_i$. Computing $\mathrm{Z}_S(A)$ does not require to explicitly verify whether $v \in \mathrm{Z}_S(A)$ for every $v \in S$. Rather, the algorithm can create, when initializing $\mathcal{H}_i$, a subset $S_A \subseteq S$ for every orbit as above such that it holds

$$\mathrm{Z}_S(A) \subseteq S_A \ . \tag{11}$$

For $i = 1$, this set contains all and only the vertices in S whose label is the same as the label of the single vertex of the patterns. For $1 < i \leq k$, we can use Fact 3: when creating $\mathcal{H}_i$ on line 13, the algorithm can associate to each orbit A of a pattern in the set $\mathcal{H}_i$ returned by `createChildren`, a set S_A obtained by taking the intersections of the image sets $\mathrm{Z}_S(B)$ on S of every parent B of the orbit A, which are available from $\mathcal{Z}_i$, i.e.,

$$S_A \doteq \bigcap_{B \text{ parent of } A} \mathrm{Z}_S(B) \ .$$

The computation of these sets can be done in the call to the `createChildren` function on line 13 of Alg. 1, for $1 < i \leq k$, and just before the starting of the loop on line 4 for $i = 1$. The properties of the orbit child-parent relation (Fact 3) therefore enable a faster computation of the collection $\mathcal{Z}_i$ because $\mathrm{Z}_S(A) = \mathrm{Z}_{S_A}(A)$, and we only need to check for subgraph isomorphisms involving S_A, which may be much smaller than S. We remark that, thanks to eq. (2), we need to find only one subgraph isomorphism for each vertex in S_A, rather than enumerating all of them.

Maintaining the sets S_A for every orbit A of a pattern $P \in \mathcal{H}_i$ allows for pruning $\mathcal{H}_i$ *before* even computing the collection $\mathcal{Z}_i$ of the image sets. The idea is that the sets S_A can be used in place of the exact image set $\mathrm{Z}_S(A)$ to compute an upper bound to the eVC-dimension of $(V, \mathcal{R}_i)$ on S. It holds by definition that $S_A \supseteq \mathrm{Z}_S(A)$ for every orbit A, so a call to `getEVCBound` (with the minor tweak of not getting rid of duplicated sets on line 2 of Alg. 3) that uses the collection of these supersets would return a valid upper bound $\hat{b}_i^*$ to the eVC-dimension of $(V, \mathcal{R}_i)$ on S. Thus, a call to `getEpsilon` with parameters $\hat{b}_i^*$ and δ/k, would return a value $\hat{\varepsilon}_i$ that is not smaller than the value ε_i that would be returned if we used b_i^*. We can then further improve MaNIACS by adding a do-while loop as the first step of every iteration of the loop on lines 4–14. This inner loop is

exactly the same as the do-while loop on lines 6–11, but with $\hat{b}_i^*$ being used in place of b_i^*. At each iteration of this loop, some orbits and therefore some patterns may be pruned from $\mathcal{H}_i$ because not frequent enough, resulting potentially in a lower bound to the eVC-dimension, thus in a lower $\hat{\varepsilon}_i$, thus creating a positive feedback loop. The improved algorithm has exactly the same properties as the vanilla MaNIACS, i.e., Thm. 4.7 holds.

The pruning strategies above can be incorporated in the call to the `createChildren` function. The call to `getImageSets` on line 5 also offers opportunities for pruning. MaNIACS computes one image set $\mathrm{Z}_S(A) = \mathrm{Z}_{S_A}(A)$ at a time, and it evaluates whether $v \in S_A$ belongs to the image set, one vertex v at a time. With the goal of maximizing pruning, we can first sort the orbits of a pattern by increasing size of their sets, and then compute the image sets of the orbits according to the obtained order. We can stop *early* the identification of the image set of an orbit A, if the sum between the number of vertices in S_A that are left to be examined, and the number of vertices in S_A that we found to belong to $\mathrm{Z}_{S_A}(A)$, divided by the size of S, is less than $\tau - \varepsilon_i$. Thus, we can also skip computing the image sets of the remaining orbits of the same pattern, and we can remove the pattern from $\mathcal{H}_i$.

Pruning is extremely important for MaNIACS, not only for *computational* efficiency reasons, but also for *statistical* efficiency reasons, as aggressive pruning leads to better bounds to the eVC-dimension, and therefore to a smaller bound to the maximum estimation error, i.e., to a better approximation quality guarantee.

4.5 Varying the sample size

Aggressive pruning *at the early levels* (i.e., when i is small) is particularly important to obtain an efficient approximation algorithm for FSPM, for the reasons mentioned at the end of the previous section. The value ε_i effectively controls how aggressive the pruning is: the smaller this value, the more patterns we *may* be able to prune, because they have an MNI-frequency smaller than $\tau - \varepsilon_i$. There are three "tunable knobs" to obtain a lower ε_i, given how this quantity is computed (see Thm. 3.1): obtaining a better (i.e., smaller) bound to the eVC-dimension, acting on the "allocation" of δ to the levels, or using a larger sample size s. The first is certainly a great direction for future work, although not an easy one: in our preliminary investigations we have found it extremely hard to derive smaller bounds to the eVC-dimension that are also efficient to compute (inefficient-to-compute bounds are easy to obtain, but useless for practical purposes). Tuning the allocation of δ to the different levels, e.g., allocating a larger fraction of δ to the first level, and decreasingly smaller to lower levels (higher i) would be straightforward to do, but has very limited impact, due to the fact that the failure probability appears only as the argument of a logarithm in the computation of ε_i (see eq. (7), where ϕ is the failure probability).

Thus, the only available direction of attack is to use *different sample sizes* $(s_i)_{i=1}^k$, *at different levels*, and specifically *larger* sample sizes for *lower* values of i, i.e., $s_1 > s_2 > \cdots > s_k$. The *sample schedule* $(s_i)_{i=1}^k$ becomes an input parameter of this variant of the algorithm. In our experimental evaluation we used a *geometric decaying factor* α so that $s_i = \alpha^{i-1} s_1$, but in general one can use any sample schedule. Having multiple sample sizes does *not* mean that at each level the algorithm must create a *new* sample S_i *from scratch*. Doing so would actually be extremely inefficient, as it would require to essentially re-mine each new sample S_i from level 1 to level i, and thus being unable to use the optimizations described in the previous section. Since we draw vertices *uniformly at random* from V to create the sample S, we can set $S_1 = S$, and create S_i, $1 < i \leq k$, by *subsampling* S_{i-1}, i.e., by drawing s_i vertices from it. This approach guarantees that S_i is a uniform random sample of V of size s_i, because "a uniform random sample of a uniform random sample is a uniform random sample". We remark that the vertices included in S_i are chosen *without using any information*

obtained from $S_1, \ldots, S_{i-1}$. This way, we avoid introducing dependencies between the samples, which would make it harder to prove the correctness of this variant of our algorithm. Thus, by creating "concentric" samples $S_1 \supset S_2 \supset \cdots \supset S_k$, we maintain the efficiency of the algorithm and avoid having to modify the proof of correctness.

The modifications to the implementation of MaNIACS required by this variant are minimal. We need to call `getImageSets` with S_i as argument on line 5 of Alg. 1; we must use $\mathrm{f}_{S_i}(P)$ on line 10 and line 12; and we must create the sample S_{i+1} by subsampling S_i (keeping s_{i+1} vertices at random) after computing the children on line 13.

This variant of the algorithm offers exactly the same guarantees as MaNIACS, i.e., Thm. 4.7 holds for this variant as well. The proof follows the same line as the proof for Thm. 4.7, with the only difference that the union bound is now applied to the events "S_i is a η_i-sample for $(V, \mathcal{R}_i)$".

In addition to being essentially the only available option, using larger sample sizes at the early levels (lower i) is also ingenious because it *opportunistically leverages* the computational properties of the task at hand: the patterns considered at the early levels (e.g., edges, wedges, triangles) are "simple", in the sense that it is very easy to check, given a vertex, whether it partakes in a subgraph isomorphism of the pattern. Thus, even if we have to perform such a check for more vertices (due to the larger sample size), the additional cost is minimal, and actually could be "time well spent" if it leads (as it does, see Sect. 5) to additional pruning at the lower levels thanks to the smaller value ε_i obtained via the larger sample size. We believe this insight to be of key importance, and to be an example of the kind of study of the task at hand that needs to be conducted in order to develop efficient algorithms for data analysis.

5 EXPERIMENTAL EVALUATION

In this section, we report the results of our experimental evaluation of MaNIACS on real datasets, comparing it to exact algorithms. The goals of the evaluation are to

- study the *trade-off between the sample size and the Maximum Absolute Error (MaxAE)* in the estimation of the frequencies of the patterns, and compare the observed MaxAE with the upper bounds ε_i to it that are output by the algorithm;
- evaluate the accuracy of MaNIACS in terms of the quality of the collection of patterns returned in output, compared to the exact collection;
- assess the running time of MaNIACS, and compare it with that of exact algorithms;
- understand the impact of the α parameter on the behavior of the algorithm.

Evaluation Metrics. We evaluate the output quality in terms of:

- *Maximum Absolute Error (MaxAE)* in the frequency estimations of the patterns (i.e., what MaNIACS guarantees (Thm. 4.7));
- *Precision*, i.e., the fraction of returned patterns that are actually frequent; and
- *Kendall's rank correlation coefficient*, i.e., the correlation between the ordered vectors of frequency estimates and actual frequencies. Values close to 1 indicate strong agreement between the two rankings, while values close to -1 indicate disagreement. When $\mathrm{FP}_V(\tau) = \emptyset$, we set it to 0.

We do not report the *recall*, i.e., the fraction of real frequent patterns returned by MaNIACS, because Thm. 4.7 (probabilistically) guarantees a *perfect recall*, and that was indeed the case in *all* the runs of our experiments. Thus, an important result of our experimental evaluation is that, in terms of recall, *MaNIACS behaves in practice even better than what is guaranteed by the theory.*

Baselines. We compare the performance of MaNIACS to that of a *naïve* exact algorithm that searches for the frequent patterns in the whole graph (i.e., as if using a sample containing all the vertices in the graph), without the need to compute any ε_i. The naïve exact algorithm returns the

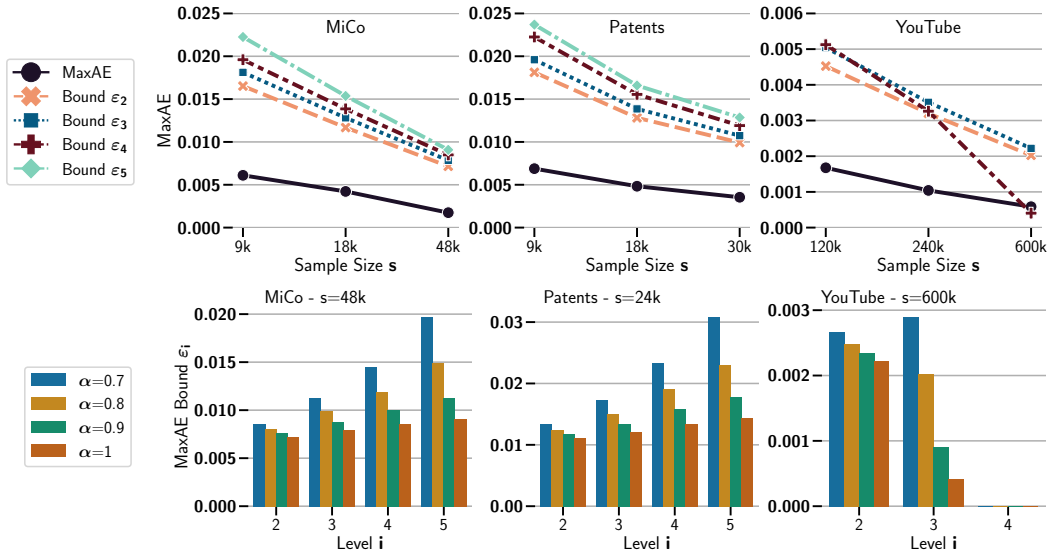


Fig. 4. Empirical Maximum Absolute Error (MaxAE) and error bounds ε_i for each level i, at fixed minimum frequency threshold τ, for MiCo (left, $\tau = 0.09$), Patents (middle, $\tau = 0.17$), and YouTube (right, $\tau = 0.08$). Upper plots: varying sample sizes, fixed $\alpha = 1$; lower plots: fixed (initial) sample size, varying α.

complete collection of frequent patterns, together with their frequencies, and allows to compute the error in the estimations of the frequencies made by MaNIACS, for any pattern. We also compare MaNIACS with a recent *exact* pattern mining approach, Peregrine [29], which is a *parallel algorithm written in C++*. Peregrine is a pattern-aware graph mining system able to explore the subgraphs of interest directly, hence bypassing unnecessary isomorphism and canonicality checks. Peregrine generates an exploration plan by means of two abstractions, the *anti-edge* and the *anti-vertex*, which express structural constraints on patterns. Frequent labeled patterns are found by considering unlabeled (or partially labeled) patterns, and returning labeled matches. Frequent labeled patterns are extended with unlabeled vertices to discover larger frequent labeled patterns. The difference in the languages used for the implementation makes direct comparison of absolute running time not entirely informative, but we use this information to discuss the *scalability* of the algorithms. We do not compare with other exact Java implementations such as GraMi [19] because they search for edge-induced patterns, and do not compute the pattern exact frequencies. We also do not compare MaNIACS with any of the other approximate methods described in Sect. 2 because they either suffer from similar issues or they offer no guarantees on the quality of the approximation.

Implementation and Environment. We implemented MaNIACS and the naïve exact algorithm in Java 1.8, while we used the original C++ implementation of Peregrine. The code of the algorithms is publicly available at https://github.com/lady-bluecopper/MaNIACS, complete with instructions on how to run all the experiments, and two Jupyter Notebooks with the full results of our experimental evaluation, which are qualitatively similar to the ones we report here. The first notebook, *results.ipynb*, includes the analysis on the impact of the sample size on the performance of MaNIACS, while the second notebook, *results_tist.ipynb*, includes the results on the concentric-sample-based version of MaNIACS and its performance level by level. We run our experiments on a 32-Core (2.54 GHz) AMD EPYC 7571 Amazon AWS instance, with 250GB of RAM, and running Amazon Linux 2.

Parameter Configuration. We run MaNIACS using several sample sizes, frequency thresholds, and values for α, while the parameters k, c,and δ were always set to 5, 0.5, and 0.1, respectively. The last two have minimal impact on the performance (see eq. (7)). Given that the sample extracted from the graph highly affects the quality of the results, we perform each test five times and report the averages. The variances among runs are always small, except in a few cases that we report. The exact baselines only use k and the minimum frequency threshold τ.

Datasets. We consider three real world networks, whose characteristics are summarized in Table 4 in App. A.2. All datasets are publicly available. MiCo [19] is a co-authorship network: nodes represent authors and edges collaborations between them. Node labels indicate each author's research field. Patents[33] is a network of all the citations in utility patents granted between 1975 and 1999. Each node label indicates the year the patent was granted. YouTube [14] is a network with nodes representing videos. Two videos are connected if they are related. The vertex label is a combination of a video's rating and length.

MaxAE: empirical values and error bounds. Figure 4 (upper) shows the empirical MaxAE in the output collection of MaNIACS,[12] together with its theoretical upper bounds ε_i computed by MaNIACS, at various sample sizes, with fixed min. frequency threshold and $\alpha = 1$. As expected, the upper bounds ε_i and the observed MaxAE decrease very fast as the sample size increases. The empirical MaxAE is at least 2.5 times lower than the upper bounds, and in some cases almost five times lower: MaNIACS works even better in practice than what is guaranteed by the theory. This fact is not surprising as some of the bounds used in the analysis are known to be loose because they consider the worst case for the sample and for the distribution of frequencies, and improving them is an important direction for future work.

Figure 4 (lower) presents the MaxAE bounds ε_i as α varies, for a fixed sample size s (48k for Mico, 24k for Patents, 600k for YouTube). We recall that the parameter α controls the sample schedule, i.e., how the sample size changes when the algorithm explores the various levels the of the orbit search space (see Sect. 4.5). Lower values of α lead to smaller sample sizes at the deeper levels (higher i), which then result in larger values of ε_i, but has an almost mirrored beneficial impact on the running time (discussed below).

Quality of the output collection. We evaluate the quality of the collection of patterns that MaNIACS outputs by measuring the precision w.r.t. the exact collection and the Kendall rank correlation. Table 3 shows the results. We remark once again that recall was always one in all runs of the algorithm: the collection output by MaNIACS is always a superset of the exact collection of frequent patterns, as probabilistically guaranteed by the analysis. It is also important to realize that MaNIACS does not and cannot offer guarantees on the precision (in some sense, one has to choose between guarantees on the recall or guarantees on the precision), because the precision depends on the distribution of the pattern frequencies around the threshold τ, which is unknown to the algorithm: if there are many patterns with an exact frequency slightly below τ (i.e., "almost-frequent"), MaNIACS is "forced" to include them in the output because it cannot distinguish between them and the actually frequent ones. This inclusion lowers the precision, which is the price to pay in order to obtain an approximate collection with a recall of 1 much faster than by performing the exact computation.

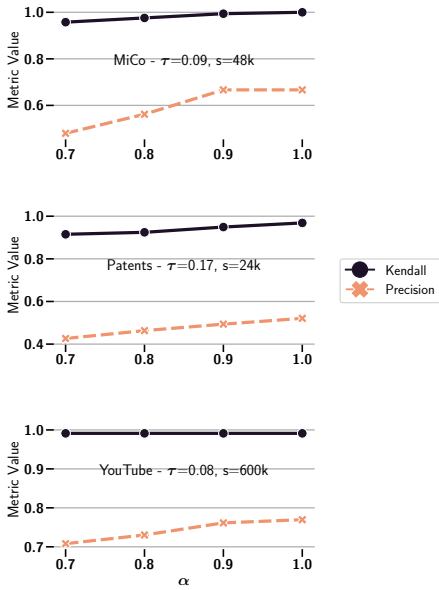
Despite this lack of guarantees on the precision, MaNIACS obtains very good results given the small sample sizes that it uses (see Table 3a): on Patents with $\tau = 0.23$, MaNIACS achieves a precision of 0.52 with a sample size of 9k, which is roughly 0.3% of the graph size, i.e., a very

[12] As the baselines are *exact*, we do not report this information for them, because it would be zero. The same holds for the later analysis of precision and Kendall's rank correlation.

Table 3. Quality of the output collection: precision and Kendall's rank correlation.

Dataset	τ	s	Precision	Kendall
MiCo	0.14	9k	0.690	1.00000
		18k	0.920	1.00000
		48k	1.000	1.00000
	0.09	9k	0.518	0.93802
		18k	0.612	0.97333
		48k	0.667	1.00000
Patents	0.23	9k	0.521	0.34947
		18k	0.589	0.34947
		30k	0.517	0.73094
	0.17	9k	0.439	0.69916
		18k	0.488	0.72166
		30k	0.517	0.73094
YouTube	0.10	120k	0.900	0.97778
		240k	0.900	0.96667
		600k	0.900	1.00000
	0.08	120k	0.653	0.97524
		240k	0.750	0.98667
		600k	0.769	0.99121

(a) Varying sample size, fixed $\alpha = 1$



(b) Fixed (initial) sample size, varying α

small sample. For YouTube, MaNIACS gets very high precision and Kendall's rank correlation by sampling less than 2% of the vertices.

As expected, the precision and the Kendall correlation increase with the sample size, with the Kendall reaching very high values in most cases. The reason for this phenomenon is that, the upper bounds ε_i decrease as the sample size grows (see eq. (7)), and so do the frequency estimation errors (as shown and discussed for the MaxAE previously). With larger samples, MaNIACS obtains lower upper bounds, thus it can perform a better pruning of superfluous patterns, leading to better precision and Kendall correlation, as shown in Table 3. Figure 8 (in App. A.3) shows how the size of the output collection changes as the sample size grows.

The plots in Table 3b show the impact of α on precision and Kendall's correlation, and the advantages of having this parameter: for $\alpha = .9$ the precision and Kendall's correlation are essentially the same as for $\alpha = 1$, while, as we discuss in the next paragraph, leading to a much smaller runtime. For lower values of α, we see that there is a drop in precision (due to the smaller sample sizes, thus larger MaxAE error bounds ε_i), but a very minor drop in correlation: even if more false positives are included in the output, the low empirical error in the estimation of the frequencies of all patterns included in the output results in a very accurate ranking of the patterns.

Running time. The motivation behind the development of sampling-based approximation algorithms is that they allow to obtain *approximate* results of *(probabilistically) guaranteed quality* much faster than exact algorithms.

Figure 5 (upper) shows the running time of MaNIACS at different sample sizes (x-axis), fixed τ and $\alpha = 1$, compared to the running times of the naïve exact algorithm and Peregrine (which, we recall, is implemented in C++, while the other algorithms are implemented in Java). The shaded

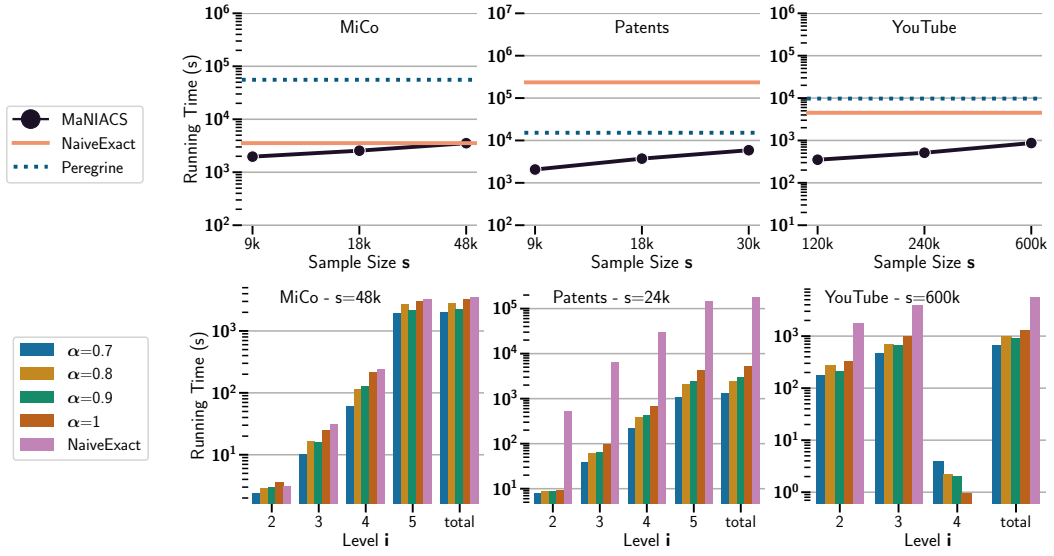


Fig. 5. Running times of MaNIACS and the exact algorithm, at fixed minimum frequency threshold τ, for MiCo (left, $\tau = 0.09$), Patents (middle, $\tau = 0.17$), and YouTube (right, $\tau = 0.08$). Upper plots: varying sample size, fixed $\alpha = 1$; lower plots: fixed (initial) sample size, varying α.

area indicates the standard deviation. This area is evident only in MiCo, as the running times for the other datasets across the runs are very similar.

MaNIACS is much faster than both exact algorithms (the y-axis has logarithmic scale), sometimes by orders of magnitude. It is not surprising that on smaller graphs, such MiCo, at larger sample sizes, there is little or no advantage in using an approximate method such as MaNIACS: the speedup that could be obtained by examining a smaller number of vertices is counterbalanced by the less effective pruning of the pattern search space, due to the large values of ε_i (and thus small values $\tau - \varepsilon_i$) at small sample sizes (see Fig. 9 in App. A.3). The lack of pruning forces MaNIACS to compute the frequencies of a larger number of candidate patterns, which obviously takes a significant amount of time. But sampling-based approximation algorithms, like MaNIACS, shine on larger graphs, such as Patents and YouTube, which are really the interesting case in practice, where our approximate algorithm is *orders of magnitude* faster than the exact ones.

Figure 5 (lower) reports the running times for different values of α and fixed initial sample size s. Lower running times are in general achieved with lower value of α, as a smaller sample is used at deeper levels. However, smaller sample sizes may lead to more superfluous patterns retained and extended. In some cases, the number of such *superfluous* patterns leads to a larger running time than if a larger value of α was used, or even, *for that specific level*, than the naïve exact algorithm (see, for example, level $i = 4$ for YouTube, or $\alpha = 0.9$ vs. $\alpha = 0.8$ for $i = 5$ on MiCo). The relationship between running time and α is complex due to the distribution of frequencies around $\tau - \varepsilon_i$, which determines the number of superfluous patterns.

Figure 6 presents the running times of MaNIACS, Peregrine, and the naïve exact algorithm at fixed sample sizes and $\alpha = 1$, while changing τ. We recall that Peregrine is implemented in C++, while the other algorithms are implemented in Java. This makes the comparison of absolute running times not entirely appropriate, but we can still draw some conclusions.

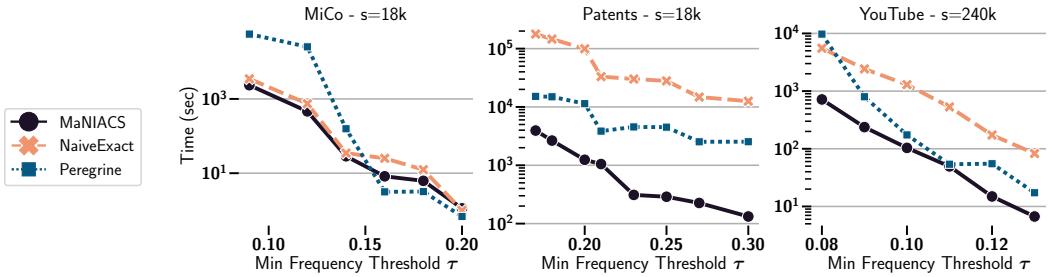


Fig. 6. Running time of MaNIACS, its exact version, and Peregrine, varying min frequency threshold τ, for MiCo (left, $s = 18$k), Patents (middle, $s = 18$k), and YouTube (right, $s = 240$k).

In particular, it seems that Peregrine does not scale as well as the other algorithms at τ decreases (which is the interesting case), to the point of becoming even slower (in absolute runtimes terms) than the naïve exact algorithm in some cases. For larger values of τ, and on small graphs (like MiCO for $\tau = 0.2$), the task of frequent pattern mining is so "easy" that it does not really matter which algorithm one uses. When the task gets harder, due to the larger number of potential patterns at lower values of τ, the approach taken by MaNIACS shows its advantages.

6 CONCLUSIONS

We presented MaNIACS, a sampling-based algorithm that outputs high-quality approximations of the collection of frequent subgraph patterns in a large graph according to the MNI frequency. To compute the quality of the approximation, MaNIACS relies on the empirical VC-dimension, a concept from statistical learning theory that ties the maximum frequency estimation error to *sample-dependent* properties. We showed how to compute an upper bound on the eVC-dimension and how to use the resulting bound on the estimation error to prune the pattern search space to avoid expensive-but-worthless computations. The results of our experimental evaluation showed that MaNIACS achieves high-precision results while being up to two orders of magnitude faster than exact algorithms, and that concentric samples reduce the running time significantly, with only a little reduction in accuracy. Interesting directions for future work include: the design of sampling-based algorithms for measures other than MNI which use the concept of overlap graph, the derivation of better upper bounds to the empirical VC-dimension of the range spaces, the derivation of formulas for the number of labeled patterns with m labels for $k > 5$, finding an optimal sample schedule for the concentric-samples variant of our algorithm, and an extension of our approach that uses parallelization to speed up the algorithm.

ACKNOWLEDGMENTS

We thank Cigdem Aslay and Muhammad Anis Uddin Nasir for their help in the preliminary phase of this work. Part of this work is supported by the National Science Foundation grant 2006765 (https://www.nsf.gov/awardsearch/showAward?AWD_ID=2006765).

REFERENCES

[1] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour. 2016. Scalemine: Scalable Parallel Frequent Subgraph Mining in a Single Large Graph. In *SC*.

[2] I. Alobaidi, J. Leopold, and A. Allami. 2019. The Use of Frequent Subgraph Mining to Develop a Recommender System for Playing Real-Time Strategy Games. In *ICDM*. 146–160.

[3] Ç. Aslay, M.A.U. Nasir, G. De Francisci Morales, and A. Gionis. 2018. Mining Frequent Patterns in Evolving Graphs. In *CIKM*. 923–932.

[4] S.K. Bera and C. Seshadhri. 2020. How to Count Triangles, without Seeing the Whole Graph. In *KDD*. 306–316.
[5] V. Bhatia and R. Rani. 2018. Ap-FSM: A parallel algorithm for approximate frequent subgraph mining using Pregel. *Exp. Sys. Appl.* 106 (2018), 217–232.
[6] M.A. Bhuiyan, M. Rahman, and M. Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *ICDM*. 91–100.
[7] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. 2017. Counting Graphlets: Space vs Time. In *WSDM*. 557–566.
[8] M. Bressan, F. Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *TKDD* 12, 4 (2018).
[9] M. Bressan, S. Leucci, and A. Panconesi. 2019. Motivo: Fast Motif Counting via Succinct Color Coding and Adaptive Sampling. *PVLDB* 12, 11 (2019), 1651–1663.
[10] B. Bringmann and S. Nijssen. 2008. What is frequent in a single graph?. In *PAKDD*. 858–863.
[11] T. Calders, J. Ramon, and D. Van Dyck. 2015. Single-Graph Support Measures. In *Quantitative graph theory: Mathematical Foundations and Applications*. Chapter 10, 303–325.
[12] M.H. Chehreghani, T. Abdessalem, A. Bifet, and M. Bouzbila. 2020. Sampling informative patterns from large single networks. *FGCS* 106 (2020), 653–658.
[13] X. Chen, J .and Qian. 2020. DwarvesGraph: A High-Performance Graph Mining System with Pattern Decomposition. arXiv:2008.09682 [cs.DC]
[14] X. Cheng, C. Dale, and J. Liu. 2008. Statistics and social network of YouTube videos. In *IWQoS*. 229–238.
[15] F. Chierichetti, A. Dasgupta, R. Kumar, S. Lattanzi, and T. Sarlós. 2016. On sampling nodes in a network. In *WWW*. 471–481.
[16] F. Chierichetti and S. Haddadan. 2018. On the Complexity of Sampling Vertices Uniformly from a Graph. In *ICALP*.
[17] G. Das. 2009. Sampling Methods in Approximate Query Answering Systems. In *Encyclopedia of Data Warehousing and Mining*. 1702–1707.
[18] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. 2005. Frequent substructure-based approaches for classifying chemical compounds. *TKDE* 17, 8 (2005), 1036–1050.
[19] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. 2014. Grami: Frequent subgraph and pattern mining in a single large graph. *PVLDB* 7, 7 (2014), 517–528.
[20] W. Fan, X. Wang, Y. Wu, and J. Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.
[21] W Feller. 1968. An Introduction to Probability Theory and Its Applications. *New York: Wiley* 1 (1968).
[22] M. Fiedler and C. Borgelt. 2007. Subgraph support in a single large graph. In *ICDMW*. 399–404.
[23] S. Ghazizadeh and S.S. Chawathe. 2002. SEuS: Structure extraction using summaries. In *DS*. 71–85.
[24] V. Guralnik and G. Karypis. 2001. A scalable algorithm for clustering sequential data. In *ICDM*. 179–186.
[25] G. Han and H. Sethu. 2016. Waddling random walk: Fast and accurate sampling of motif statistics in large graphs. In *ICDM*. 181–190.
[26] T.A.D. Henderson. 2017. *Frequent Subgraph Analysis and its Software Engineering Applications*. Ph. D. Dissertation. Case Western Reserve University.
[27] V. Ingalalli, D. Ienco, and P. Poncelet. 2018. Mining frequent subgraphs in multigraphs. *Information Sciences* 451 (2018), 50–66.
[28] A.P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica. 2018. ASAP: Fast, Approximate Graph Pattern Mining at Scale. In *OSDI*. 745–761.
[29] K. Jamshidi, R. Mahadasa, and K. Vora. 2020. Peregrine: A Pattern-Aware Graph Mining System. In *EuroSys*.
[30] T. Junttila and P. Kaski. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In *ALENEX*. 135–149.
[31] M. Kuramochi and G. Karypis. 2004. Grew-a scalable frequent subgraph discovery algorithm. In *ICDM*.
[32] M. Kuramochi and G. Karypis. 2005. Finding frequent patterns in a large sparse graph. *DMKD* 11, 3 (2005), 243–271.
[33] J. Leskovec, J. Kleinberg, and C. Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*. 177–187.
[34] Y. Li, P.M. Long, and A. Srinivasan. 2001. Improved Bounds on the Sample Complexity of Learning. *J. Comput. System Sci.* 62, 3 (2001), 516–527.
[35] M. Löffler and J.M. Phillips. 2009. Shape Fitting on Point Sets with Probability Distributions. In *ESA*. 313–324.
[36] I. Melckenbeeck, P. Audenaert, D. Colle, and M. Pickavet. 2018. Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. *Bioinformatics* 34, 8 (2018), 1372–1380.
[37] I. Melckenbeeck, P. Audenaert, T. Van Parys, Y. Van De Peer, D. Colle, and M. Pickavet. 2019. Optimising orbit counting of arbitrary order by equation selection. *BMC bioinformatics* 20, 1 (2019), 1–13.
[38] J. Meng, N. Pitaksirianan, and Y. Tu. 2019. Generalizing Design of Support Measures for Counting Frequent Patterns in Graphs. In *BigData*. 533–542.

[39] J. Meng, N. Pitaksirianan, and Y.-C. Tu. 2020. Counting frequent patterns in large labeled graphs: a hypergraph-based approach. *DMKD* (2020), 1–42.
[40] A. Mrzic, P. Meysman, W. Bittremieux, P. Moris, B. Cule, B. Goethals, and K. Laukens. 2018. Grasping frequent subgraph mining for bioinformatics applications. *BioData Mining* 11, 20 (2018).
[41] M.A.U. Nasir, Ç. Aslay, G. De Francisci Morales, and M. Riondato. 2021. TipTap: Approximate Mining of Frequent k-Subgraph Patterns in Evolving Graphs. *TKDD* 15, 3 (2021), 1–35.
[42] K. Paramonov, D. Shemetov, and J. Sharpnack. 2019. Estimating Graphlet Statistics via Lifting. In *KDD*. 587–595.
[43] N. Pashanasangi and C. Seshadhri. 2020. Efficiently Counting Vertex Orbits of All 5-Vertex Subgraphs, by EVOKE. In *WSDM*. 447–455.
[44] L. Pellegrina, C. Cousins, F. Vandin, and M. Riondato. 2020. MCRapper: Monte-Carlo Rademacher Averages for Poset Families and Approximate Pattern Mining. In *KDD*. 2165–2174.
[45] A. Pinar, C. Seshadhri, and V. Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *WWW*. 1431–1440.
[46] G. Pólya. 1937. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica* 68, 1 (1937), 145–254.
[47] G. Preti, G. De Francisci Morales, and M. Riondato. 2021. MaNIACS: Approximate Mining of Frequent Subgraph Patterns through Sampling. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '21)*. ACM.
[48] N. Pržulj, D.G. Corneil, and I. Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.
[49] S. Purohit, S. Choudhury, and L. B. Holder. 2017. Application-specific graph sampling for frequent subgraph mining and community detection. In *Big Data*.
[50] P. Ribeiro, P. Paredes, M. E. P. Silva, D. Aparicio, and F. Silva. 2021. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Comput. Surv.* 54, 2, Article 28 (March 2021), 36 pages.
[51] P. Ribeiro and F. Silva. 2014. Discovering colored network motifs. In *Complex Networks V*. Springer, 107–118.
[52] M. Riondato, J.A. DeBrabant, R. Fonseca, and E. Upfal. 2012. PARMA: A Parallel Randomized Algorithm for Association Rules Mining in MapReduce. In *CIKM*.
[53] M. Riondato and E. Upfal. 2014. Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. *TKDD* 8, 4 (2014), 20.
[54] M. Riondato and E. Upfal. 2018. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *TKDD* 12, 5 (2018).
[55] M. Riondato and F. Vandin. 2014. Finding the True Frequent Itemsets. In *SDM*.
[56] R. A. Rossi, N. K. Ahmed, A. Carranza, D. Arbour, A. Rao, S. Kim, and E. Koh. 2020. Heterogeneous Graphlets. *TKDD* 15, 9 (2020).
[57] T. K. Saha, A. Katebi, W. Dhifli, and M. Al Hasan. 2019. Discovery of Functional Motifs from the Interface Region of Oligomeric Proteins Using Frequent Subgraph Mining. *TCBB* 16, 5 (2019), 1537–1549.
[58] C. Seshadhri and S. Tirthapura. 2019. Scalable Subgraph Counting: The Methods Behind The Madness. In *WWW*.
[59] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
[60] N. Talukder and M.J. Zaki. 2016. A distributed approach for graph mining in massive networks. *DMKD* 30, 5 (2016), 1024–1052.
[61] C.H.C. Teixeira, A.J. Fonseca, M. Serafini, G. Siganos, M.J. Zaki, and A. Aboulnaga. 2015. Arabesque: A System for Distributed Graph Mining. In *SOSP*. 425–440.
[62] N. Vanetik, E. Gudes, and S. E. Shimony. 2002. Computing frequent graph patterns from semistructured data. In *ICDM*. 458–465.
[63] N. Vanetik, S.E. Shimony, and E. Gudes. 2006. Support measures for graph data. *DMKD* 13, 2 (2006), 243–260.
[64] Vladimir N. Vapnik. 1998. *Statistical learning theory*. Wiley.
[65] J. Wang, Y. Wang, W. Jiang, Y. Li, and K. Tan. 2020. Efficient Sampling Algorithms for Approximate Temporal Motif Counting. In *CIKM*. 1505–1514.
[66] P. Wang, J. Lui, B. Ribeiro, D. Towsley, J. Zhao, and X. Guan. 2014. Efficiently estimating motif statistics of large networks. *TKDD* 9, 2 (2014), 8.
[67] P. Wang, J. Lui, D. Towsley, and J. Zhao. 2016. Minfer: A method of inferring motif statistics from sampled edges. In *ICDE*. 1050–1061.
[68] X. Zhao, Y. Chen, C. Xiao, Y. Ishikawa, and J. Tang. 2016. Frequent subgraph mining based on Pregel. *Comput. J.* 59, 8 (2016), 1113–1128.
[69] R. Zou and L. B. Holder. 2010. Frequent subgraph mining on a single large graph using sampling techniques. In *MLG '10: Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. 171–178.

A SUPPLEMENTARY MATERIAL

A.1 Orbits of unlabeled patterns

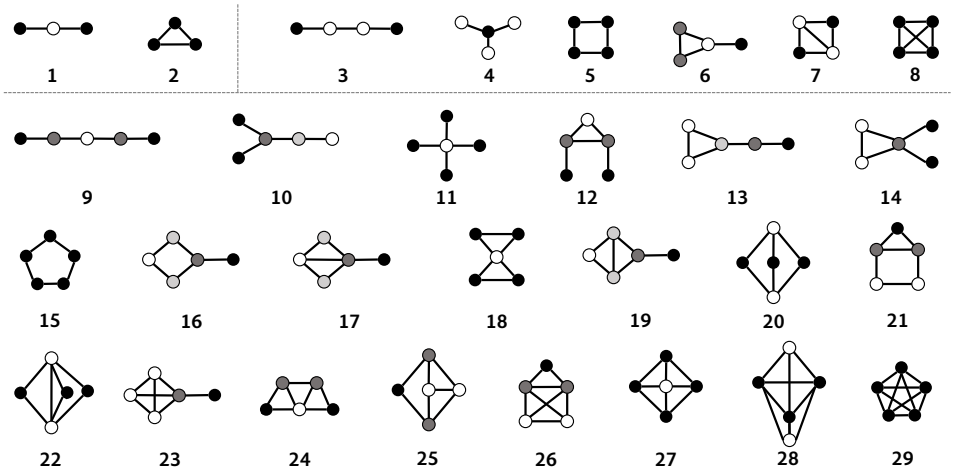


Fig. 7. Unlabeled patterns for $k \in \{3, 4, 5\}$ and their orbits indicated with different shades of grey (vertices with the same shade of grey belong to the same orbit).

A.2 Reproducibility

Alg. 4 looks for a vertex-induced occurrence of P, given the partial vertex assignment M, and returns true or false depending on whether such an occurence exists.

Algorithm 4: existsIsomorphism

Input: Pattern P, Partial match M
Output: **true** iff M contains a match for each vertex of P

```
1  if |M| = |V_P| then return true
2  u ← vertex of P not already in M
3  cands ← ⋂_{w∈u.Γ∩M} M[w].Γ
4  foreach z ∈ cands with the same label as u do
5      isMatch ← true
6      foreach w ∈ M do
7          if w ∉ u.Γ and M[w] ∈ z.Γ then
8              isMatch ← false; break
9      if isMatch then
10         M[u] ← z
11         if existsIsomorphism(P, M) then return true
12 return false
```

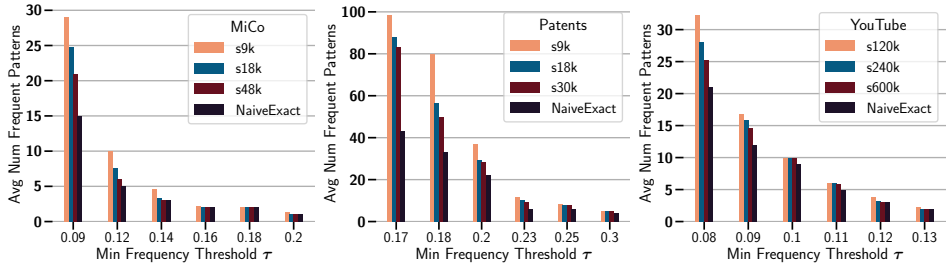
Table 4. Characteristics of the datasets.

	Phy-Cit	MiCo	Patents	YouTube
Vertices	30k	100k	2.7M	4.5M
Edges	347k	1M	13M	43M
Labels	6	29	4	12
Density	7.46×10^{-4}	2.16×10^{-4}	3.67×10^{-6}	4.17×10^{-6}
Avg Label Freq	5k	3.4k	689k	382k
Med Label Freq	5.9k	2.1k	672k	472k
Avg Edge Freq	16k	2.4k	1.5M	563k
Med Edge Freq	12k	1.0k	1.2M	535k

Table 4 reports the main characteristics of the datasets considered in the experimental evaluation.

A.3 Additional results

Figure 8 displays the average number of frequent patterns (i.e., the size of the output collection) at different values of τ, as the sample size increases. Since the recall of the output collection was always one, then the difference from the exact values, for fixed τ, corresponds to the number of false positives. The number of false positives decreases as the sample size increases, thanks to the better pruning, but as discussed in the main body, a larger sample requires more time to be analyzed, thus one must accept a trade-off between precision and runtime.



(a) MiCo (b) Patents (c) YouTube

Fig. 8. Average number of patterns found by MaNIACS, together with the exact number of frequent patterns, varying minimum frequency threshold τ.

Figure 9 shows the number of candidate frequent patterns examined at each level of the search space. The impact of α is more noticeable at deeper levels, because for each pattern retained at level i, MaNIACS generates a number of extensions of size $i + 1$ proportional to m.

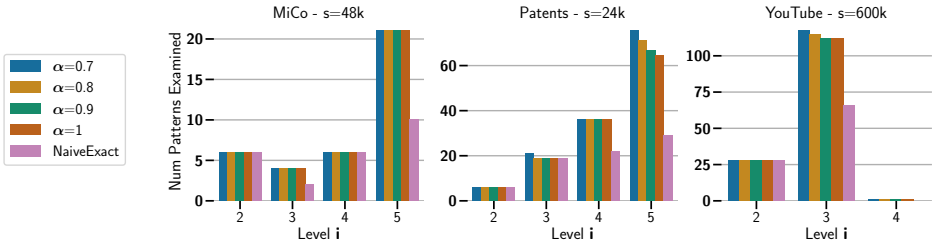


Fig. 9. Number of candidate frequent patterns at each layer for various α, at fixed minimum frequency threshold τ, for MiCo (left, $\tau = 0.09$), Patents (middle, $\tau = 0.17$), and YouTube (right, $\tau = 0.08$).