Motivo: fast motif counting via succinct color coding and adaptive sampling

Marco Bressan*
Dipartimento di Informatica,
Sapienza Università di Roma
bressan@di.uniroma1.it

Stefano Leucci
Department of Algorithms & Complexity,
Max Planck Institute for Informatics
stefano.leucci@mpi-inf.mpg.de

Alessandro Panconesi*
Dipartimento di Informatica,
Sapienza Università di Roma
ale@di.uniroma1.it

ABSTRACT

The randomized technique of color coding is behind state-of-the-art algorithms for estimating graph motif counts. Those algorithms, however, are not yet capable of scaling well to very large graphs with billions of edges. In this paper we develop novel tools for the "motif counting via color coding" framework. As a result, our new algorithm, MOTIVO, scales to much larger graphs while at the same time providing more accurate motif counts than ever before. This is achieved thanks to two types of improvements. First, we design new succinct data structures for fast color coding operations, and a biased coloring trick that trades accuracy versus resource usage. These optimizations drastically reduce the resource requirements of color coding. Second, we develop an adaptive motif sampling strategy, based on a fractional set cover problem, that breaks the additive approximation barrier of standard sampling. This gives multiplicative approximations for all motifs at once, allowing us to count not only the most frequent motifs but also extremely rare ones.

To give an idea of the improvements, in 40 minutes MOTIVO counts 7-nodes motifs on a graph with 65M nodes and 1.8B edges; this is 30 and 500 times larger than the state of the art, respectively in terms of nodes and edges. On the accuracy side, in one hour MOTIVO produces accurate counts of ≈ 10.000 distinct 8-node motifs on graphs where state-of-the-art algorithms fail even to find the second most frequent motif. Our method requires just a high-end desktop machine. These results show how color coding can bring motif mining to the realm of truly massive graphs using only ordinary hardware.

PVLDB Reference Format:
M. Bressan, S. Leucci, A. Panconesi. Motivo: fast motif counting via succinct color coding and adaptive sampling. *PVLDB*, 12(11): 1651-1663, 2019.
DOI: https://doi.org/10.14778/3342263.3342640

*Supported in part by the ERC Starting Grant DMAP 680153, a Google Focused Research Award, and by the MIUR grant "Dipartimenti di eccellenza 2018-2022" awarded to the Department of Computer Science of the Sapienza University of Rome.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342640

1. INTRODUCTION

Graphlets, also called motifs or patterns, are small induced subgraphs of a graph. Graphlets are often considered the "building blocks" of networks [17, 23, 28, 29], and their analysis has helped understanding network evolution [1], designing better graph classification algorithms [28], and developing cutting-edge clustering techniques [29].

A fundamental problem in graphlet mining and analysis is graphlet counting: estimating as accurately as possible the number of copies of a given graphlet (e.g., a tree, a clique, etc.) in a graph. Graphlet counting has a long and rich history, which began with triangle counting and received intense interest in recent years [2, 6, 7, 10, 12, 15, 17, 20, 21, 25, 26, 27, 30]. Since exact graphlet counting is notoriously hard, one must resort to approximate probabilistic counting to obtain algorithms with an acceptable practical performance. Approximate counting is indeed often sufficient, for example when performing hypothesis testing (deciding if a graph comes from a certain distribution or not) or estimating the clustering coefficient of a graph (the fraction of triangles among 3-node graphlets).

The simplest formulation of approximate graphlet counting, which we adopt in this work, is the following. We are given a simple graph G on n nodes, an integer $k > 2$, and two approximation parameters $\epsilon, \delta \in (0, 1)$. For each graphlet H on k nodes (the clique, the path, the star etc.), we want a very reliable and accurate estimate of the number of induced copies of H in G: with probability at least $1 - \delta$, all estimates should be within a factor $(1 \pm \epsilon)$ of the actual values. Note that we are talking about induced copies; non-induced copies are easier to count and can be derived from the induced ones. Our goal is to develop practical algorithms that solve this problem for sizes of G and H that were out of reach before, i.e., graphs with hundreds of millions of edges and graphlets on more than 5 or 6 nodes. We remark that scaling k is harder than it may seem at a first sight: just the number of distinct graphlets grows from 21 for $k = 5$ to over 10.000 for $k = 8$. Thus, mining larger graphlets is not simply a matter of fixing k – instead, it requires new ideas.

To appreciate the main obstacles to large-scale graphlet counting, let us review the existing techniques and their limitations. All known efficient algorithms for approximating graphlet counts are based on sampling graphlets from G. One popular way of sampling graphlets is to define a random walk over the set of graphlets of G, simulate it until it reaches stationarity, and take a sample [6, 12, 15, 25]. This technique is simple, elegant, and has a small memory footprint. Unfortunately, it can be extremely inefficient: it is

known that, even if G is fast-mixing, the random walk may need $\Omega(n^{k-1})$ steps to reach stationarity and/or to hit on the most frequent graphlet of G [8, 9].

A completely different sampling approach is the CC algorithm of [8, 9], an extension of the well-known color coding technique [4] based on two key observations. The first observation is that color coding can be used to build an abstract "urn" which contains a sub-population of all the k-trees of G that is very close to the true one. The second observation is that the task of sampling k-graphlets can be reduced, with minimal overhead, to sampling k-trees from the urn. One can thus estimate graphlet counts in two steps: the *build-up phase*, where one builds the urn from G, and the *sampling phase*, where one samples k-trees from the urn. Building the urn requires time $O(a^k m)$ and space $O(a^k n)$ for some $a > 0$, where n and m are the number of nodes and edges of G, while sampling takes a variable but typically small amount of time per sample. The resulting algorithm, CC, outperforms random walk-based approaches and is the current state of the art [8, 9].

Although CC has extended the outreach of graphlet counting techniques, it cannot effectively cope with graphs with billions of edges and values of k beyond six. This is due to two main bottlenecks. First, the time and space taken by the build-up phase are significant and prevent CC from scaling to the values of G and k that we are interested in this paper. For example, on a machine with 64GB of main memory, the largest graph for which CC runs successfully has 5.4M nodes for $k = 5, 6$ and just 2M nodes for $k = 7$. Second, taking s samples from the abstract urn gives the usual additive $1/s$-approximation, which means we can accurately count only those graphlets whose occurrences are a fraction at least $1/s$ of the total. Unfortunately, in many graphs, most graphlets have a very low relative frequency, and CC is basically useless to count them.

In this work we overcome the limitations of CC by making two main contributions to the "motif counting via color coding" framework. The first contribution is reducing the running time and space usage of the build-up phase. We do so in three ways. First, we introduce succinct color coding data structures that can represent colored rooted trees on up to 16 nodes with just one machine word, and support frequent operations (e.g. merging trees) in just a few elementary CPU instructions. This is key, as colored trees are the main objects manipulated in the build-up phase. Second, for large graphs we present a simple "biased coloring" trick that we use to trade space and time against the accuracy of the urn (the distance of the urn's distribution from the actual tree distribution of G), whose loss we quantify via concentration bounds. Third, we describe a set of architectural and implementation optimizations. These ingredients make the build-up phase significantly faster and bring us from millions to billions of edges and from $k = 5$ to $k = 8$.

Our second contribution is for the sampling phase and is of a fundamentally different nature. To convey the idea, imagine having an urn with 1000 balls of which 990 red, 9 green, and 1 blue. Sampling from the urn, we will quickly get a good estimate of the fraction of red balls, but we will need many samples to witness even one green or blue ball. Now imagine that, after having seen those red balls, we could remove from the urn 99% of all red balls. We would be left with 10 red balls, 9 green balls, and 1 blue ball. At this point we could quickly get a good estimate of the fraction of green balls. We could then ask the urn to delete almost 99% of the red and green balls, and we could quickly estimate the fraction of blue balls. What we show here is that the urn built in the build-up phase can be used to perform essentially this "deletion" trick, where the object to be removed are treelets. In this way, roughly speaking, we can first estimate the most frequent graphlet, then delete it from the urn and proceed to the second most frequent graphlet, delete it from the urn and so on. This means we can in principle obtain a small *relative* error for all graphlets, independently of their relative abundance in G, thus breaking the $\Theta(1/\epsilon)$ barrier of standard sampling. We name this algorithm AGS (adaptive graphlet sampling). To obtain AGS we actually develop an online greedy algorithm for a fractional set cover problem. We provide formal guarantees on the accuracy and sampling efficiency of AGS via set cover analysis and martingale concentration bounds.

In order to properly assess the impact of the various optimizations, in this paper we have added them incrementally to CC, which acts as a baseline. In this way, it is possible to assess in a quantitative way the improvements due to the various components.

Our final result is an algorithm, MOTIVO[1], that scales well beyond the state of the art in terms of input size and simultaneously ensures tighter guarantees. To give an idea, for $k = 7$ MOTIVO manages graphs with tens of millions of nodes and billions of edges, the largest having 65M nodes and 1.8B edges. This is 30 times and 500 times (respectively in terms of n and m) what CC can manage. For $k = 8$, our largest graph has 5.4M nodes and 50M edges (resp. 18 and 55 times CC). All this is done in 40 minutes on just a high-end commodity machine. For accuracy, the most extreme example is the YELP graph, where for $k = 8$ all but two graphlets have relative frequency below 10^{-7}. With a budget of 1M samples, CC finds only the *first* graphlet and misses all the others. MOTIVO instead outputs accurate counts ($\epsilon \leq 0.5$) of more than 90% of all graphlets, or 10.000 in absolute terms. The least frequent ones of those graphlets have frequency below 10^{-20}, and CC would need $\sim 3 \cdot 10^3$ years to find them even if it took one billion samples per second.

1.1 Related work

Counting induced subgraphs is a classic problem in computer science. The exact version is notoriously hard, and even detecting a k-clique in an n-node graph requires time $n^{\Omega(k)}$ under the Exponential Time Hypothesis [11]. Practical exact counting algorithms exist only for $k \leq 5$; currently, the fastest one is ESCAPE [20], which still takes a week on graphs with a few million nodes. We use ESCAPE for computing some of our ground-truth counts.

For approximate graphlet counting many techniques have been proposed. For $k \leq 5$, one can sample graphlets via path sampling (do a walk on k nodes in G and check the subgraph induced by those nodes) [17, 26, 27]. This technique, however, does not scale to $k > 5$. A popular approach is to sample graphlets via random walks [6, 25, 12, 15]. The idea is to define two graphlets as adjacent in G if they share $k-1$ nodes. This implicitly defines a reversible Markov chain over the graphlets of G which can be simulated efficiently. Once at stationarity, one can take the sample and easily compute an unbiased estimator of the graphlet frequencies.

[1]The C++ source code of MOTIVO is publicly available at `https://bitbucket.org/steven_/motivo`.

Unfortunately, these algorithms cannot estimate counts, but only frequencies. Even then, they may give essentially no guarantee unless one runs the walk for $\Omega(n^{k-1})$ steps, and in practice they are outperformed by CC [8, 9]. Another recent approach is that of edge-streaming algorithms based on reservoir sampling [22], which however are tailored to $k \leq 5$. As of today, the state of the art in terms of G and k is the color-coding based CC algorithm of [8, 9]. CC can manage graphs on $\sim$5M nodes for $k = 5, 6$, on $\sim$2M nodes for $k = 7$, and on less than 0.5M nodes for $k = 8$, in a matter of minutes or hours. As said above, CC does not scale to massive graphs and suffers from the "naive sampling barrier" that allows only for additive approximations. Finally, we shall mention the algorithm of [16] that in a few minutes can estimate clique counts with high accuracy on graphs with tens of millions of edges. We remark that that algorithm works *only* for cliques, while MOTIVO is general purpose and provides counts for *all* graphlets at once.

Preliminaries and notation. We denote the host graph by $G = (V, E)$, and we let $n = |V|$ and $m = |E|$. A *graphlet* is a connected graph $H = (V_H, E_H)$. A *treelet* T is a graphlet that is a tree. We let $k = |V_H|$. We denote by $\mathcal{H}$ the set of all k-node graphlets, i.e., all non-isomorphic connected graphs on k nodes. When needed we denote by H_i the i-th graphlet of $\mathcal{H}$. A colored graphlet has a color $c_u \in [k]$ associated to each one of its nodes u. A graphlet is *colorful* if its nodes have pairwise distinct colors. We denote by $C \subseteq [k]$ a subset of colors. We denote by (T, C) or T_C a colored treelet whose nodes span the set of colors C; we only consider colorful treelets, i.e., the case $|T|{=}|C|$. Often treelets and colored treelets are rooted at a node $r \in T$. Finally, d_v and $u \sim v$ will denote the degree of a node v in G and a neighbor u of v, respectively.

Paper organization. Section 2 reviews color coding and the CC algorithm. Section 3 introduces our data structures and techniques for accelerating color coding. Section 4 describes our adaptive sampling strategy. Section 5 concludes with our experiments.

2. COLOR CODING AND CC

The color coding technique was introduced in [4] to probabilistically detect paths and trees in a graph. The CC algorithm of [8, 9] is an extension of color coding that enables sampling colorful graphlet occurrences from G. It consists of a *build-up phase* and a *sampling phase*.

2.1 The build-up phase

The goal of this phase is to build a *treelet count table* that is the abstract "urn" used for sampling. First, we do a *coloring* of G: for each $v \in G$ independently, we draw uniformly at random a color $c_v \in [k]$. We then look at the treelets of G that are colorful. For each v and every rooted colored treelet T_C on up to k nodes, we want a count $c(T_C, v)$ of the number of copies of T_C in G that are rooted in v (note that we mean *non-induced* copies here). To this end, for each v we initialize $c(T_C, v) = 1$, where T is the trivial treelet on 1 node and $C = \{c_v\}$. For a T_C on $h > 1$ nodes, the count $c(T_C, v)$ is then computed via dynamic programming, as follows. First, T has a unique decomposition into two subtrees T' and T'' rooted respectively at the root r of T and at a child of r. The uniqueness is given by a total order over treelets (see next section). Now, since T' and T'' are smaller than T, their counts have already been computed for all possible colorings and all possible rootings in G. Then $c(T_C, v)$ is given by (see [9]):

$$c(T_C, v) = \frac{1}{\beta_T} \sum_{u \sim v} \sum_{\substack{C' \subset C \\ |C'|=|T'|}} c(T'_{C'}, v) \cdot c(T''_{C''}, u) \qquad (1)$$

where β_T is the number of subtrees of T isomorphic to T'' rooted at a child of r. CC employs (1) in the opposite way: it iterates over all pairs of counts $c(T'_{C'}, v)$ and $c(T''_{C''}, u)$ for all $u \sim v$, and if $T'_{C'}, T''_{C''}$ can be merged in a colorful treelet T_C, then it adds $c(T'_{C'}, v) \cdot c(T''_{C''}, u)$ to the count $c(T_C, v)$. This requires to perform a check-and-merge operation for each count pair, which is quite expensive (see below).

A simple analysis gives the following complexity bounds:

THEOREM 1. *([9], Theorem 5.1) The build-up takes time $O(a^k m)$ and space $O(a^k n)$, for some constant $a > 0$.*

The size of the dynamic programming table is a major bottleneck for CC: already for $k = 6$ and $n = 5$M, it takes 45GB of main memory [9].

2.2 The sampling phase

The goal of this phase is to sample *colorful* graphlet copies u.a.r. from G, using the treelet count table from the build-up phase. The key observation in [8, 9] is that we only need to sample colorful non-induced *treelet* copies; by taking the corresponding induced subgraph in G, we then obtain our induced graphlet copies. Colorful treelets are sampled via a multi-stage sampling, as follows. First, draw a node $v \in G$ with probability proportional to $\eta_v = \sum_{T_C} c(T_C, v)$. Second, draw a colored treelet T_C with probability proportional to $c(T_C, v)/\eta_v$. We want to sample a copy of T_C rooted at v. To this end we decompose T_C into $T'_{C'}$ and $T''_{C''}$, with $T'_{C'}$ rooted at the root r of T and $T''_{C''}$ at a child of r (see above). We then recursively sample a copy of $T'_{C'}$ rooted at v, and a copy of $T''_{C''}$ rooted at node $u \sim v$, where u is chosen with probability $c(T''_{C''}, u)/\sum_{z \sim v} c(T''_{C''}, z)$. Note that computing this probability requires listing all neighbors z of v, which takes time proportional to d_v. Finally, we combine $T'_{C'}$ and $T''_{C''}$ into a copy of T_C. One can see that this gives a colorful copy of T drawn uniformly at random from G.

Consider then a given k-graphlet H_i (e.g., the clique), and let c_i be the number of colorful copies of H_i in G. We can estimate c_i as follows. Let χ_i be the indicator random variable of the event that a graphlet sample x is an occurrence of H_i. It is easy to see that $\mathbb{E}[\chi_i] = c_i\, \sigma_i / t$, where σ_i is the number of spanning trees in H_i and t is the total number of colorful k-treelets of G. Both t and σ_i can be computed quickly, by summing over the treelet count table and via Kirchhoff's theorem (see below). We thus let $\hat{c}_i = t\, \sigma_i^{-1} \chi_i$, and $\mathbb{E}[\hat{c}_i] = c_i$. By standard concentration bounds we can then estimate c_i by repeated sampling. Note that the expected number of samples to find a copy of H_i grows as $1/c_i$. This is the additive error barrier of CC's sampling.

Estimators and errors. Finally, let us see how to estimate the number of total (i.e., *uncolored*) copies g_i of H_i in G, which is our final goal. First, note that the probability that a fixed subset of k nodes in G becomes colorful is $p_k = k!/k^k$. Therefore, if G contains g_i copies of H_i, and c_i is the number those copies that become colorful, then by linearity of expectation $\mathbb{E}[c_i] = p_k g_i$ (seeing c_i as a random variable). Hence, $\hat{g}_i = c_i/p_k$ is an unbiased estimator for g_i.

This is, indeed, the count estimate returned by CC and by MOTIVO.

For what concerns accuracy, the error given by $\hat{g}_i$ can be formally bounded via concentration bounds. An additive error bound is given by Theorem 5.3 of [9], which we slightly rephrase. Let $g = \sum_i g_i$ be the total number of induced k-graphlet copies in G. Then:

THEOREM 2 ([9], THEOREM 5.3). *For all $\epsilon > 0$,*

$$\Pr\left[|\hat{g}_i - g_i| > \frac{2\epsilon g}{1-\epsilon}\right] = \exp(-\Omega(\epsilon^2 g^{1/k})).$$

Since we aim at multiplicative errors, we prove a multiplicative bound, which is also tighter than Theorem 2 if the maximum degree Δ of G is small. We prove (see Appendix A):

THEOREM 3. *For all $\epsilon > 0$,*

$$\Pr\left[|\hat{g}_i - g_i| > \epsilon\, g_i\right] < 2\exp\left(-\frac{2\epsilon^2}{(k-1)!}\frac{p_k\, g_i}{\Delta^{k-2}}\right). \quad (2)$$

In practice, $\hat{g}_i$ appears always concentrated. In other words, the coloring does not introduce a significant distortion. Note that this holds for a single coloring, i.e., for a single execution of CC. If one averages over γ executions with independent colorings, the probabilities decrease exponentially with γ.

3. SPEEDING UP COLOR CODING

This section details step-by-step the data structures and optimizations that are at the heart of MOTIVO's efficiency. First, we implemented a C++ porting of CC (which is in Java), translating all algorithms and data structures carefully to their closest C++ equivalent. This porting is our baseline benchmark. We then incrementally plugged in our data structures and optimizations, ultimately obtaining MOTIVO. Figures 1 and 2 show how the performance improves during the process. Note that MOTIVO uses 128-bit counts, while CC uses 64-bit counts which causes overflows (just the number of 6-stars centered in a node of degree 2^{16} is $\approx 2^{80}$).

Before moving on, we note that a perfectly fair porting of CC is not possible. This is because CC makes heavy use of fast specialized integer hash tables provided by the `fastutil`[2] library, which exists only in Java and seems to be crucial to its performance. Indeed, for the porting we tested three popular libraries – `google::sparse_hash_map` and `google::dense_hash_map` of the sparsehash library[3], and `std::unordered_map` from the C++ containers library. With the first two, the porting is up to 17× slower than CC, and with the latter one it is up to 7× slower. Nonetheless, after all optimizations are in place, MOTIVO is faster than CC and relatively insensitive to the hash table choice, with running times changing by at most 30%. We thus use `std::unordered_map` in the porting for the sake of measuring the impact of optimizations, and use the memory-parsimonious `google::sparse_hash_map` in MOTIVO.

3.1 Succinct data structures

The main objects manipulated by CC and MOTIVO are rooted colored treelets and their associated counts, which are stored in the treelet count table. We first describe their implementation in CC, then introduce the one of MOTIVO.

[2] `http://fastutil.di.unimi.it/`
[3] `https://github.com/sparsehash/sparsehash`

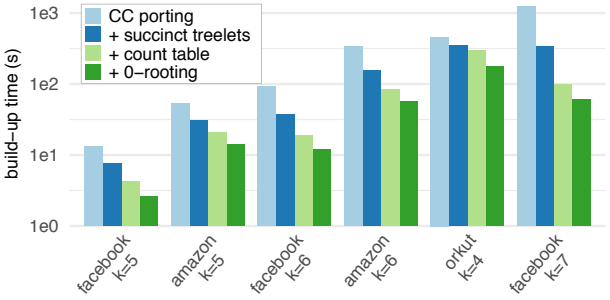


Figure 1: cumulative impact of our optimizations on the running time of the build-up phase.

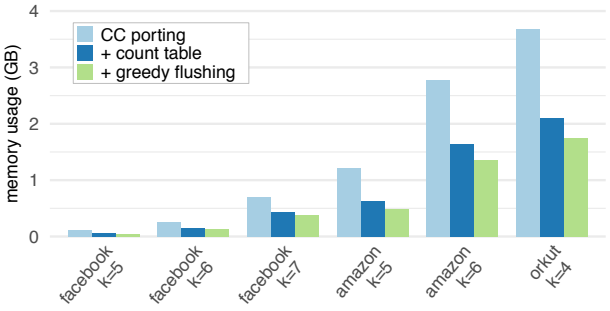


Figure 2: cumulative impact of our optimizations on the memory usage of the build-up phase.

The internals of CC. In CC, each T_C has a unique *representative instance*, that is a classic pointer-based tree data structure equipped with a structure storing the colors. The pointer to this instance acts as unique identifier for T_C. The treelet count table of CC is then implemented as follows: for each $v \in G$, a hash table maps the pointer of each T_C to the count $c(T_C, v)$, provided that $c(T_C, v) > 0$. Thus, each entry uses 128 bits – 64 for the pointer and 64 for the count – plus the overhead of the hash table. For computing $c(T_C, v)$, CC processes every neighbor $u \sim v$ as follows (see also Section 2.1). For every pair of counts $c(T'_{C'}, v)$ and $c(T''_{C''}, u)$ in the hash tables of v and u, check that $C' \cap C'' = \emptyset$, and that $T''_{C''}$ comes before the smallest subtree of $T'_{C'}$ in the total order of the treelets (see below). If these conditions hold, then $T'_{C'}$ and $T''_{C''}$ can be merged into a treelet T_C whose unique decomposition yields precisely $T'_{C'}$ and $T''_{C''}$. Then, the value of $c(T_C, v)$ in the hash table of v is incremented by $c(T'_{C'}, v) \cdot c(T''_{C''}, u)$. The expensive part is the check-and-merge operation, which CC does with a recursive algorithm on the treelet representative instances. This has a huge impact, since on a graph with a billion edges the check-and-merge is easily performed trillions of times. In fact, in our porting, check-and-merge operations consume from 30% to 70% of the build-up time.

Motivo's treelets. Let us now describe MOTIVO's data structures, starting with an uncolored treelet T rooted at $r \in T$. We encode T with the binary string s_T defined as follows. Perform a DFS traversal of T starting from r. Then the i-th bit of s_T is 1 (resp. 0) if the i-th edge is traversed moving away from (resp. towards) r. For all $k \leq 16$, this encoding takes at most 30 bits, which fits nicely in a 4-byte integer type (padded with 0s). The lexicographic ordering over the

s_T's gives a total ordering over the T's that is exactly the one used by CC. This ordering is also a tie-breaking rule for the DFS traversal: the children of a node are visited in the order given by their rooted subtrees. This implies that every T has a well-defined unique encoding s_T. Moreover, merging T' and T'' into T requires just concatenating $1, s_{T''}, s_{T'}$ in this order. This makes check-and-merge operations extremely fast: we measure a speedup ranging from 150× for $k = 5$ to 1000× for $k = 7$.

This succinct encoding supports the following operations:

- `getsize()`: return the number of vertices in T. This is one plus the Hamming weight of s_T, which can be computed in a single machine instruction (e.g., `POPCNT` from the SSE4 instruction set).
- `merge(`T'`,` T''`)`: merge two treelets T', T'' by appending T'' as a child of the root of T'. This requires just to concatenate $1, s_{T''}, s_{T'}$ in this order.
- `decomp(`T`)`: decompose T into T' and T''. This is the inverse of `merge` and is done by suitably splitting s_T.
- `sub(`T`)`: compute the value β_T of (1), i.e., the number of subtrees of T that (i) are isomorphic to the treelet T'' of the decomposition of T, and (ii) are rooted at some child of the root. This is done via bitwise `shift` and `and` operations on s_T.

A colored rooted treelet T_C is encoded as the concatenation s_{T_C} of s_T and of the characteristic vector s_C of C.[4] For all $k \leq 16$, s_{T_C} fits in 46 bits. Set-theoretical operations on C become bitwise operations over s_C (`or` for union, `and` for intersection). Finally, the lexicographical order of the s_{T_C}'s induce a total order over the T_C's, which we use in the count table (see below). An example of a colored rooted treelet and its encoding is given in Figure 3 (each node labelled with its color).

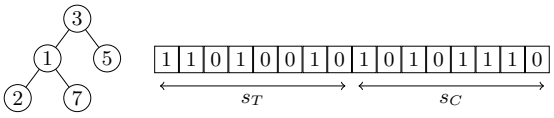


Figure 3: a colored rooted treelet and its encoding, shown for simplicity on just $8 + 8 = 16$ bits.

Motivo's count tables. In CC, treelet counts are stored in n hash tables, one for each node $v \in G$. In each table, the pair $(T_C, c(T_C, v))$ is stored using the pointer to the representative instance of T_C as key. This imposes the overhead of dereferencing a pointer before each check-and-merge operation to retrieve the actual structure of T_C. Instead of using a hash table, MOTIVO maintains the key-value pairs $(T_C, c(T_C, v))$ such that $c(T_C, v) > 0$ in a set of arrays, one for each $v \in G$ and for each treelet size $h \in [k]$. These arrays are sorted lexicographically w.r.t. the order of the keys described above. This makes iterating over the counts extremely fast and, since each key T_C is explicitly stored using its representation s_{T_C}, eliminates the need for dereferencing. The price to pay is that searching for a given T_C in the count table requires a binary search. However, this still takes only $O(k)$ time, since the whole record has length $O(6^k)$.[5] Recall that MOTIVO uses 128-bit counts[6], whereas CC uses 64-bit integers. This increases by 64 bits the space per pair compared to CC; however, MOTIVO saves 16 bits per pair by packing s_{T_C} into 48 bits, using a total of 176 bits per pair. Finally, in place of $c(T_C, v)$, MOTIVO actually stores the cumulative count $\eta(T_C, v) = \sum_{T'_{C'} \leq T_C} c(T'_{C'}, v)$. In this way each $c(T_C, v)$ can be recovered with negligible overhead, and the total count for a single node v (needed for sampling) is just at the end of the record.

MOTIVO's count table supports the following operations:

- `occ(`v`)`: returns the total number of colorful treelet copies rooted at v. Running time: $O(1)$.
- `occ(`T_C, v`)`: returns the total number of copies of T_C rooted at v. Running time: $O(k)$ via binary search.
- `iter(`T, v`)`: returns an iterator to the first pair $(T_C, c(T_C, v))$ stored in the table such that $T_C = (T, C)$. Running time: $O(k)$, plus $O(1)$ per accessed treelet.
- `iter(`T_C, v`)`: If $c(T_C, v) > 0$, returns an iterator to the pair $(T_C, c(T_C, v))$. When $c(T_C, v) = 0$, the returned iterator refers to the pair $T'_{C'}$, where $T'_{C'}$ if the first colored treelet that follows T_C for which $c(T'_{C'}, v) > 0$. Running time: $O(k)$, plus $O(1)$ per accessed treelet.
- `sample(`v`)`: returns a random colored treelet T_C with probability proportional to $c(T_C, v)/\eta_v$. This is used in the sampling phase. Running time $O(k)$: first we get η_v in $O(1)$ time (see above); then in $O(k)$ time we draw R u.a.r. from $\{1, \ldots, \eta_v\}$; finally, we search for the first pair (T_C, η) with $\eta \geq R$, and we return T_C.

0-rooting. Consider a colorful treelet copy in G that is formed by the nodes $v_1, \ldots, v_h$. In the count table, this treelet is counted in each one of the h records of $v_1, \ldots, v_h$, since it is effectively a colorful treelet rooted in each one of those nodes. Therefore, the treelet is counted h times. This is inevitable for $h < k$, since excluding some rooting would invalidate the dynamic programming (Equation 1). However, for $h = k$ we can store only one rooting and the sampling works just as fine. Thus, for $h = k$ we count only the k-treelets rooted at their node of color 0. This cuts the running time by $30\% - 40\%$, while reducing by a factor of k the size of the k-treelets records, and by $\approx 10\%$ the total space usage of MOTIVO.

3.2 Other optimizations

Greedy flushing. To further reduce memory usage, we use a greedy flushing strategy. Suppose we are currently building the table for treelets of size h. While being built, the record of v is actually stored in a hash table, which allows for efficient insertions. However, immediately after completion it is stored on disk in the compact form described above, but still unsorted w.r.t. the other records. The hash table is then emptied and memory released. When all records have been stored on disk, a second I/O pass sorts them w.r.t. their

[4] Given an universe U, the characteristic vector $\langle x_1, x_2, \ldots \rangle$ of a subset $S \subseteq U$ contains one bit x_i for each element $i \in U$, which is 1 if $i \in S$ and 0 otherwise.

[5] By Cayley's formula: there are $O(3^k k^{-3/2})$ rooted treelets on k vertices [19], and 2^k subsets of k colors.

[6] Tests on our machine show that summing 500k unsigned integers is 1.5× slower with 128-bit than with 64-bit integers.

corresponding node. At the end, the treelet count table is stored on disk without having entirely resided in memory. The price to pay is the time for sorting, which was at most 10% of the total time in all our runs.

Neighbor buffering. Our final optimization concerns sampling. In most graphs, MOTIVO natively achieves sampling rates of 10k samples per second or higher. But on some graphs, such as BERKSTAN or ORKUT, we get only 100 or 1000 samples per second. The reason is the following. Those graphs contain a node v with a degree Δ much larger than any other node. Inevitably then, a large fraction of the treelets of G are rooted in v. This has two combined effects on the sampling phase (see Subsection 2.2). First, v will be frequently chosen as root. Second, upon choosing v will spend time $\Theta(\Delta)$ to sweep over its neighbors. The net effect is that the time to take one sample grows *superlinearly* with Δ, slowing down the sampling dramatically. To overcome this, we perform buffered sampling. If $d_v \geq 10^4$, then with a single sweep over v's neighbors, MOTIVO samples 100 neighbors at random instead of just one. It then returns the first, and caches the remaining 99 for the future. In this way, on large-degree nodes MOTIVO sweeps only once every 100 samples. As Figure 4 shows, this increases the sampling speed by $\approx 20\times$ on ORKUT and $\approx 40\times$ on BERKSTAN.

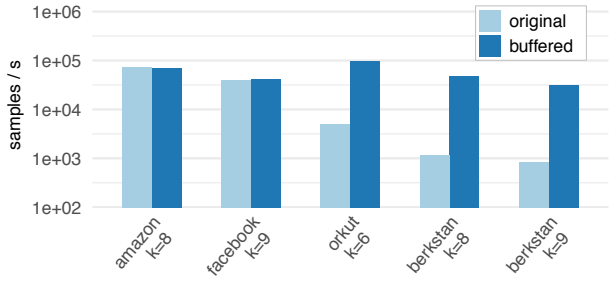


Figure 4: impact of neighbor buffering on sampling.

3.3 Implementation details

We describe some other implementation details of MOTIVO that, although not necessarily being "optimizations", are necessary for completeness and reproducibility.

Input graph. The graph G is represented by adjacency lists. Each list is a sorted static array of the vertex's neighbors; arrays of consecutive vertices are contiguous in memory. This allows for fast iterations over the set of outgoing edges of a vertex, and for $O(\log n)$-time edge-membership queries[7], that we need in the sampling phase to obtain the induced graphlet from the sampled treelet.

Multi-threading. Similarly to CC, MOTIVO makes heavy use of thread-level parallelism in both the build-up and sampling phases. For the build-up phase, for any given v the counts $c(\cdot, v)$ can be computed independently from each other, which we do using a pool of threads. As long as the number of remaining vertices is sufficiently large, each thread is assigned a (yet unprocessed) vertex v and will compute all the counts $c(T_C, v)$ for all pairs T_C. While this requires minimal synchronization, when the number of unprocessed vertices decreases below the amount of available threads, the above strategy is no longer advantageous as it would cause some of the threads to become idle. This can increase the time needed by the build-up phase if G exhibits skewed degree and/or treelet distributions. To overcome this problem, the last remaining vertices are handled differently: we allow multiple threads to concurrently compute different summands of the outermost sum of (1) for the same vertex v, i.e., those corresponding to the edges $(v, u) \in E$. Once all the incident edges of v have been processed, the partial sums are then combined together to obtain all the counts $c(\cdot, v)$. This reduces the running time by a few percentage points. For the sampling phase, samples are by definition independent and are taken by different threads.

Memory-mapped reads. In the build-up phase, to compute the count table for treelets of size h we must access the count tables of size j, for all $j < h$. For large instances, loading all those tables simultaneously in memory is infeasible. One option would be to carefully orchestrate I/O and computation, hoping to guarantee a small number of load/store operations on disk. We adopt a simpler solution: memory-mapped I/O. This delegates the I/O to the operating system in a manner that is transparent to MOTIVO, which sees all tables as if they resided in main memory. When enough memory is available this solution gives ideally no overhead. Otherwise, the operating system will reclaim memory by unloading part of the tables, and future requests to those parts will incur a page fault and prompt a reload from the disk. The overhead of this approach can be indeed measured via the number of page faults. This reveals that the total I/O volume due to page faults is less than 100MB, except for $k = 8$ on LIVEJOURNAL (34GB) and YELP (8GB) and for $k = 6$ on FRIENDSTER (15GB). However, in those cases additional I/O is inevitable, as the total size of the tables (respectively 99GB, 90GB, and 61GB) is close to or even larger than the total memory available (about 60GB).

Alias method sampling. Recall that, to sample a colorful graphlet from G, we first sample a node v with probability proportional to the number of colorful k-treelets rooted at v (Subsection 2.2). We do this in time $O(1)$ by using the alias method [24], which requires building an auxiliary lookup table in time and space linear in the support of the distribution. In our case this means time and space $O(n)$; the table is built during the second stage of the build-up process. In practice, building the table takes negligible amounts of time (a fraction of a second out of several minutes).

Graphlets. In MOTIVO, each graphlet H is encoded as an adjacency matrix packed in a 128-bit integer. Since a graphlet is a simple graph, the $k \times k$ adjacency matrix is symmetric with diagonal 0 and can be packed in a $(k-1)\times\frac{k}{2}$ matrix if k is even and in a $k \times \frac{k-1}{2}$ matrix if k is odd (see e.g. [5]). The resulting triangular matrix can then be reshaped into a $1 \times \frac{k^2-k}{2}$ vector, which fits into 120 bits for all $k \leq 16$. In fact, one can easily compute a bijection between the pair of vertices of the graphlet and the indices $\{1, \ldots, 120\}$. Before encoding a graphlet, MOTIVO replaces it with a canonical representative from its isomorphism class, computed using the Nauty library [18].

Spanning trees. By default, MOTIVO computes the number of spanning trees σ_i of H_i in time $O(k^3)$ via Kirchhoff's matrix-tree theorem which relates σ_i to the determinant of a $(k-1)\times(k-1)$ submatrix of the Laplacian matrix of H_i. To compute the number σ_{ij} of occurrences of T_i in H_j (needed

[7] This is actually $O(\log d_u)$ where (u, v) is the edge being tested. In practice, it is often the case that $d_u \ll n$.

for our sampling algorithm AGS, see Section 4), we use an in-memory implementation of the build-up phase. The time taken is negligible for $k < 7$, but is significant for $k \geq 7$. For this reason, MOTIVO caches the σ_{ij} and stores them to disk for later reuse. This accelerates sampling by up to 10×.

3.4 Biased coloring

Finally, we describe an optimization, that we call "biased coloring", that can be used to manage graphs that would otherwise be too large. Suppose for simplicity that, for each treelet T on j nodes, each $v \in G$ appears in a relatively small number of copies of T, say $k^j/j!$. Then, given a set C of j colors, a copy of T is colored with C with probability $j!/k^j$. This implies that we will have an expected $\Theta(1)$ copies of T colored with C containing v, in which case the total table size (and the total running time) will approach the worst-case space bounds.

Suppose now we bias the distribution of colors. In particular, we give probability $\lambda \ll \frac{1}{k}$ to each color in $\{1, \ldots, k-1\}$. The probability that a given j-treelet copy is colored with C is then:

$$p_{k,j}(C) = \begin{cases} j!\lambda^j & \text{if } k \notin C \\ \sim j!\lambda^{j-1} & \text{if } k \in C \end{cases} \quad (3)$$

If λ is sufficiently small, then, for most T we will have a zero count at v; and most nonzero counts will be for a restricted set of colorings – those containing k. This reduces the number of pairs stored in the treelet count table, and consequently the running time of the algorithm. The price to pay is a loss in accuracy, since a lower p_k increases the variance of the number c_i of colorful copies of H_i. However, if n is large enough and most nodes v belong to even a small number of copies of H_i, then the *total* number of copies g_i of H_i is large enough to ensure concentration. In particular, by Theorem 3 the accuracy loss remains negligible as long as $\lambda^{k-1} n/\Delta^{k-2}$ is large (ignoring factors depending only on k). We can thus trade a $\Theta(1)$ factor in the exponent of the bound for a $\Theta(1)$ factor in both time and space, especially on large graphs where saving resources is precious. To find a good value for λ, one can start with $\lambda \ll 1/kn$ and grow it until a small but non-negligible fraction of counts are positive. At this point by Theorem 3 we have achieved concentration, and we can safely proceed to the sampling phase.

Impact. With $\lambda = 0.001$, the build-up time on FRIENDSTER (65M nodes, 1.8B edges) shrinks from 17 to 10 minutes (1.7×) for $k = 5$, and from 1.5 hours to 13 minutes (7×) for $k = 6$. In both cases, the main memory usage and the disk space usage decrease by at least 2×. The relative graphlet count error increases correspondingly, as shown in Figure 5 (see Section 5 for the error definition). For $k = 7$, the build takes 20 minutes – in this case we have no comparison term, as without biased coloring MOTIVO did not terminate a run within 2 hours. Note that FRIENDSTER has 30 (500) times the nodes (edges) of the largest graph managed by CC for the same values of k [9]. In our experiments (Section 5) biased coloring is disabled since mostly unnecessary.

4. ADAPTIVE GRAPHLET SAMPLING

This section describes AGS, our adaptive graphlet sampling algorithm for color coding. Recall that the main idea of CC is to build a sort of "urn" supporting a primitive sample() that returns a colorful k-treelet occurrence u.a.r. from G. The first step of AGS is to "refine" this interface with one urn for each possible k-treelet shape T. More precisely, for every k-treelet shape T our urn should support the following primitive:

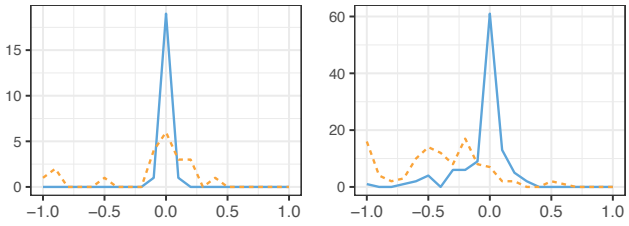


Figure 5: graphlet count error distribution of uniform and biased coloring (dashed), for k=5 and k=6.

- sample(T): return a colorful copy of T u.a.r. from G

With sample(T) one can selectively sample treelets of different shapes, and this can be used to virtually "delete" undesired graphlets from the urn. Let us try to convey the idea with a simple example. Imagine G contains just two types of colorful graphlets, H_1 and H_2, of which H_2 represents a tiny fraction p (say 0.01%). Using our original primitive, sample(), we will need $\Theta(1/p)$ calls before finding H_2. Suppose however H_1 and H_2 are spanned by treelets of different shape, say T_1 and T_2. We could then start by using sample(T_1), until we estimate accurately H_1. At this point we switch to sample(T_2), which completely ignores H_1 (since it is not spanned T_2), until we estimate accurately H_2 as well. In this way we can estimate accurately both graphlets with essentially $O(1)$ samples. Clearly, in general we have more than just two graphlets, and distinct graphlets may have the same spanning trees. Still, this adaptive sampling strategy strikingly outperforms naive sampling in the presence of rare graphlets. Moreover, AGS yields multiplicative guarantees on all graphlets, while taking only $O(k^2)$ times the minimum number of samples any algorithm must take (see below).

We now describe AGS in more detail. Recall from Section 2 that for every $v \in G$ we know $\texttt{occ}(T, v)$, the number of colorful copies of T rooted at v. Using this fact, one can easily restrict the sampling (Subsection 2.2) to a specific treelet T, thus drawing u.a.r. a colorful copy of T in G. This gives our primitive sample(T). We then start invoking sample(T), using the k-treelet T with the largest number of colorful occurrences. Eventually, some graphlet H_i spanned by T will appear enough times, say $\Theta(\frac{1}{\epsilon^2}\ln(\frac{1}{\delta}))$. We then say H_i is *covered*. Now, since we do not need more samples of H_i, we would like to continue with sample(T') for some T' that does *not* span H_i, as if we were asking to "delete" H_i from the urn. More precisely, we seek T' that minimizes the probability that by calling sample(T') we observe H_i.

The crux of AGS is that we can find T' as follows. First, we estimate the number g_i of colorful copies of H_i in G, which we can do since we have enough samples of H_i. Then, for each k-treelet T_j we estimate the number of copies of T_j that span a copy of H_i in G as $g_i\sigma_{ij}$, where σ_{ij} is the number of spanning trees of H_i isomorphic to T_j. We then divide this estimate by the number t_j of colorful copies of T_j in G, obtained summing $\texttt{occ}(T_j, v)$ over all $v \in G$. The result is an estimate of the probability that sample(T_j) spans a copy of H_i, and we choose the treelet T_{j^*} that minimizes this

probability. More in general, we need the probability that sample(T_j) spans a copy of *some* graphlet among the ones covered so far, and to estimate g_i we must take into account that we have used different treelets along the sampling.

The pseudocode of AGS is listed below. A graphlet is marked as covered when it has appeared in at least $\bar{c}$ samples. For a union bound over all k-graphlets one would set $\bar{c} = O(\frac{1}{\epsilon^2}\ln(\frac{s}{\delta}))$ where $s = s_k$ is the number of distinct k-graphlets. In our experiments we set $\bar{c} = 1000$, which gives good accuracy on most graphlets. We denote by $H_1, \ldots, H_s$ the distinct k-node graphlets and by $T_1, \ldots, T_\varsigma$ the distinct k-node treelets.

Algorithm AGS(ϵ, δ)

1: $(c_1, \ldots, c_s) \leftarrow (0, \ldots, 0)$ ▷ graphlet counts
2: $(w_1, \ldots, w_s) \leftarrow (0, \ldots, 0)$ ▷ graphlet weights
3: $\bar{c} \leftarrow \lceil \frac{4}{\epsilon^2}\ln(\frac{2s}{\delta}) \rceil$ ▷ covering threshold
4: $C \leftarrow \emptyset$ ▷ graphlets covered
5: $T_j \leftarrow$ an arbitrary treelet type
6: **while** $|C| < s$ **do**
7: **for** each i' in $1, \ldots, s$ **do**
8: $w_{i'} \leftarrow w_{i'} + \sigma_{i'j}/t_j$
9: $T_G \leftarrow$ an occurrence of T_j drawn u.a.r. in G
10: $H_i \leftarrow$ the graphlet type spanned by T_G
11: $c_i \leftarrow c_i + 1$
12: **if** $c_i \geq \bar{c}$ **then** ▷ switch to a new treelet T_j
13: $C \leftarrow C \cup \{i\}$
14: $j^* \leftarrow \arg\min_{j'=1,\ldots,\varsigma} \frac{1}{t_{j'}} \sum_{i' \in C} \sigma_{i'j'} c_{i'}/w_{i'}$
15: $T_j \leftarrow T_{j^*}$
16: **return** $(\frac{c_1}{w_1}, \ldots, \frac{c_s}{w_s})$

4.1 Approximation guarantees

We prove that, if AGS chooses the "right" treelet T_{j^*}, then we obtain multiplicative error guarantees. Formally:

THEOREM 4. *If the tree T_{j^*} chosen by AGS at line 14 minimizes* Pr[sample(T_j) *spans a copy of some $H_i \in C$] then, with probability $(1-\delta)$, when AGS stops c_i/w_i is a multiplicative $(1\pm\epsilon)$-approximation of g_i for all $i = 1, \ldots, s$.*

The proof requires a martingale analysis and is deferred to Appendix B. We stress that the guarantees hold for all graphlets, irrespective of their relative frequency. In practice, AGS gives accurate counts for many or almost all graphlets at once, depending on the graph (see Section 5).

4.2 Sampling efficiency

Let us turn to the sampling efficiency of AGS. We first observe that, in some cases, AGS does no better than naive sampling. However, this is not a limitation of AGS itself: it holds for *any* algorithm based solely on sampling treelets via the primitive sample(T). This class of algorithms is quite natural, and indeed includes the graphlet sampling algorithms of [17, 26, 27]. Formally, we prove:

THEOREM 5. *For any constant $k \geq 2$, there are graphs G in which some graphlet H represents a fraction $p_H = 1/\operatorname{poly}(n) = \Omega(n^{1-k})$ of all graphlet copies, and any algorithm needs $\Omega(1/p_H)$ calls to* sample(T) *in expectation to just find one copy of H.*

PROOF. Let T and H be the path on k nodes. Let G be the $(n-k+2, k-2)$ lollipop graph; so G is formed by a clique on $n-k+2$ nodes and a dangling path on $k-2$ nodes, connected by an arc. G contains $\Theta(n^k)$ non-induced occurrences of T in G, but only $\Theta(n)$ induced occurrences of H (all those formed by the $k-2$ nodes of the dangling path, the adjacent node of the clique, and any other node in the clique). Since there are at most $\Theta(n^k)$ graphlets in G, then H forms a fraction $p_H = \Theta(n^{1-k})$ of these. Obviously T is the only spanning tree of H; however, an invocation of sample(G, T) returns H with probability $\Theta(n^{1-k})$ and thus we need $\Theta(n^{k-1}) = \Theta(1/p_H)$ samples in expectation before obtaining H. One can make p_H larger by considering the $(n', n-n')$ lollipop graph for larger values of n'. □

Theorem 5 rules out good *absolute* bounds on the number of samples used by AGS. However, we can show that AGS is close to the best possible, clairvoyant algorithm based on sample(T). By clairvoyant we mean that the algorithm knows in advance how many sample(T_j) calls to make for every treelet T_j in order to get the desired bounds with the minimum total number of calls. Formally, we prove:

THEOREM 6. *If the tree T_{j^*} chosen by AGS at line 14 minimizes* Pr[sample(T_j) *spans a copy of some $H_i \in C$], then AGS makes a number of calls to* sample() *that is at most $O(\ln(s)) = O(k^2)$ times the minimum needed to ensure that every graphlet H_i appears in $\bar{c}$ samples in expectation.*

The proof of the theorem relies on a fractional set cover and can be found in Appendix C.

5. EXPERIMENTAL RESULTS

In this section we compare the performance of MOTIVO to CC [9] which, as said, is the current state of the art. For readability, we give plots for a subset of datasets that are representative of the entire set of results.

Set-up. We ran all our experiments on a commodity machine equipped with 64GB of main memory and 48 Intel Xeon E5-2650v4 cores at 2.5GHz with 30MB of L3 cache. We allocated 880GB of secondary storage on a Samsung SSD850 solid-state drive, dedicated to the treelet count tables of MOTIVO. Table 1 shows the graphs on which we tested MOTIVO, and the largest tested value of k. All graphs were made undirected and converted to the MOTIVO binary format. For each graph we ran MOTIVO for all $k = 5, 6, 7, 8, 9$, or until the build time did not exceed 1.5 hours; except for TWITTER and LIVEJOURNAL, where we did $k = 5, 6$ regardless of time.

Table 1: our graphs (* = with biased coloring)

graph	M nodes	M edges	source	k
FACEBOOK	0.1	0.8	MPI-SWS	9
BERKSTAN	0.7	6.6	SNAP	9
AMAZON	0.7	3.5	SNAP	9
DBLP	0.9	3.4	SNAP	9
ORKUT	3.1	117.2	MPI-SWS	7
LIVEJOURNAL	5.4	49.5	LAW	8
YELP	7.2	26.1	YLP	8
TWITTER	41.7	1202.5	LAW	6 (7*)
FRIENDSTER	65.6	1806.1	SNAP	6 (7*)

Ground truth. We computed exact 5-graphlet counts for FACEBOOK, DBLP, AMAZON, LIVEJOURNAL and ORKUT by running the ESCAPE algorithm [20]. On the remaining graphs ESCAPE died by memory exhaustion or did not return within 24 hours. For $k > 5$ and/or larger graphs, we

averaged the counts given by MOTIVO over 20 runs, 10 using naive sampling and 10 using AGS.

5.1 Computational performance

Build-up time. The table below shows the speedup of MOTIVO's build-up phase over the original Java implementation of CC; dashes mean that CC failed by memory exhaustion or 64-bit integer overflow. We do not report TWITTER and FRIENDSTER, as CC failed even for $k = 5$. MOTIVO is 2×-5× faster than CC on 5 out of 7 graphs, and never slower on the other ones.

Table 2: MOTIVO vs. CC build-up phase speedup

graph	k=5	k=6	k=7	k=8	k=9
FACEBOOK	4.2	3.4	3.9	5.5	10.0
BERKSTAN	2.2	-	-	-	-
AMAZON	2.1	1.7	1.5	1.6	2.2
DBLP	2.1	1.4	1.7	2.2	3.9
ORKUT	5.6	-	-		
LIVEJOURNAL	3.1	3.1	-	-	
YELP	2.4	-	-	-	

Count tables size. The table below shows the ratio between the main memory footprint of CC and the total external memory usage of MOTIVO; both are indicators of the total count table size. The footprint of CC is computed as the smallest JVM heap size that allowed it to run. In almost all cases MOTIVO saves a factor of 2, in half of the cases a factor of 5, and on YELP, the largest graph managed by CC, a factor of 8. For $k = 7$, CC failed on 6 over 9 graphs, while MOTIVO processed all of them with a space footprint of less than 12GB (see below).

Table 3: MOTIVO vs. CC table size shrinkage factor

graph	k=5	k=6	k=7	k=8	k=9
FACEBOOK	108.7	87.5	54.5	17.3	7.1
BERKSTAN	36.5	-	-	-	-
AMAZON	6.6	3.3	2.1	1.1	1.0
DBLP	6.2	4.0	2.4	1.1	1.0
ORKUT	8.5	-	-		
LIVEJOURNAL	11.4	3.8	-	-	
YELP	8.0	-	-	-	

Sampling speed. The table below shows the sampling speedup of MOTIVO's naive sampling with respect to CC. MOTIVO is always 10× faster, and even 160× faster on YELP. This means MOTIVO gives more accurate estimates for a fixed time budget, even though in the sampling phase it must access the count tables from disk.

Table 4: MOTIVO vs. CC sampling rate speedup

graph	k=5	k=6	k=7	k=8	k=9
FACEBOOK	14.9	13.2	9.4	50.0	115.9
BERKSTAN	29.3	-	-	-	-
AMAZON	60.7	12.6	13.2	13.2	16.5
DBLP	17.7	11.4	10.1	44.8	88.6
ORKUT	29.2	-	-		
LIVEJOURNAL	31.8	28.5	-	-	
YELP	159.6	-	-	-	

Additional remarks. MOTIVO runs in minutes on graphs that CC could not even process, and in less than one hour for all but the largest instance. The contrast with ESCAPE [20] and [6, 12, 15, 25] is even starker, as those algorithms take entire days even for graphs 10–100 times smaller. We also point out that, unlike all these algorithms, MOTIVO's behavior is very predictable, as shown in Figure 6.

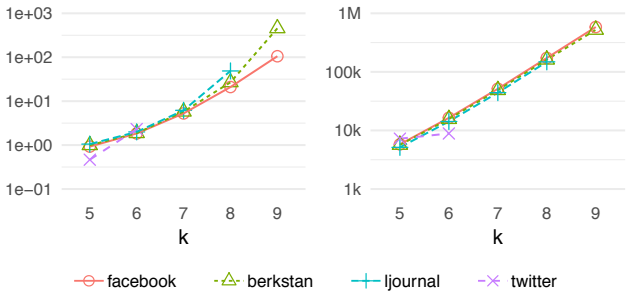


Figure 6: MOTIVO's build-up time (seconds per million edge) and space usage (bits per input node).

5.2 Accuracy

We assess the accuracy of MOTIVO's count estimates, and in particular of AGS against the naive sampling strategy. All plots below report the average over 10 runs, with whiskers for the 10% and 90% percentiles. Naive sampling is shown by the left bars, and AGS by the right bars. Note that MOTIVO's naive sampling strategy and CC's sampling algorithm are exactly the same algorithm, so for a given number of samples they give the same output. However, MOTIVO is much faster than CC and so takes many more samples per unit of time. Hence, the accuracy of CC is dominated by the accuracy of MOTIVO.

A necessary remark. The accuracy of estimates obviously depends on the number of samples taken. One option would be to fix an absolute budget, say 1M samples. Since however for $k = 5$ there are only 21 distinct graphlets and for $k = 8$ there are over 10k, we would certainly have much higher accuracy in the first case. As a compromise, we tell MOTIVO to spend in sampling the same amount of time taken by the build-up phase. This is also what an "optimal" time allocation strategy would do – statistically speaking, if we have a budget of 100 seconds and the build-up takes 5 seconds, we would perform 10 runs with 5 seconds of sampling each and average over the estimates.

Error in ℓ_1 norm. First, we evaluate how accurately MOTIVO reconstructs the global k-graphlet distribution. If $\mathbf{f} = (f_1, \ldots, f_s)$ are the ground-truth graphlet frequencies, and $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_s)$ their estimates, then the ℓ_1 error is $\ell_1(\mathbf{f}, \hat{\mathbf{f}}) = \sum_{i=1}^{s} |\hat{f}_i - f_i|$. In our experiments, the ℓ_1 error was below 5% in all cases, and below 2.5% for all $k \leq 7$.

Single-graphlet count error. The count error of H is:

$$\mathrm{err}_H = \frac{\hat{c}_H - c_H}{c_H} \tag{4}$$

where c_H is the ground-truth count of H and $\hat{c}_H$ its estimate. Thus $\mathrm{err}_H = 0$ means a perfect estimate, and $\mathrm{err}_H = -1$ means the graphlet is missed. Figure 7 shows the distribution of err_H for one run, for naive sampling (top) and AGS (bottom), as k increases from left to right. AGS is much more accurate, especially for larger values of k, and CC's naive sampling misses many graphlets. (We will give a very

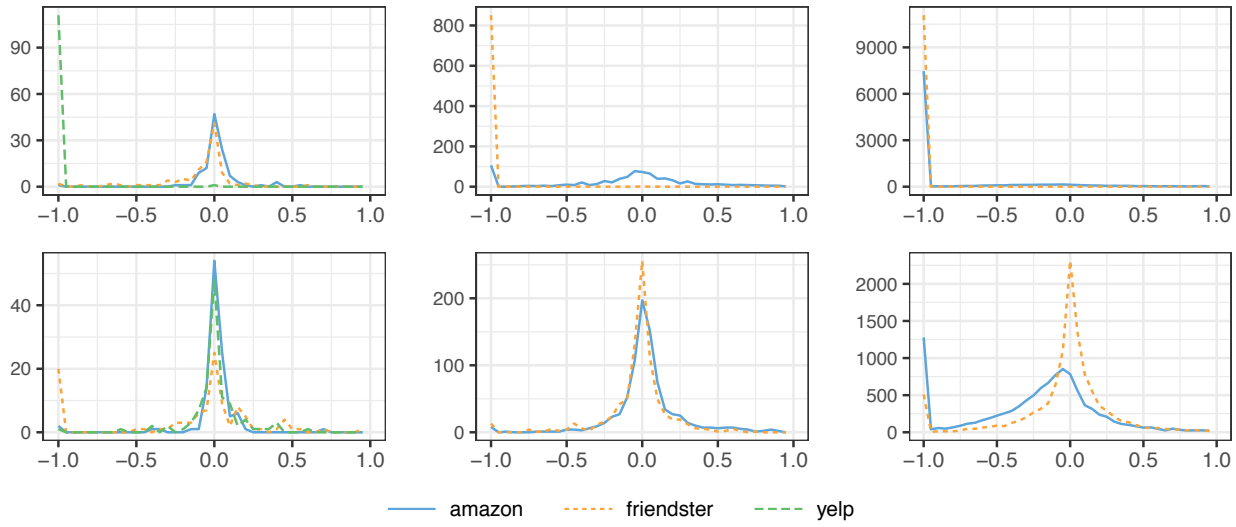


Figure 7: distribution of graphlet count error for $k = 6, 7, 8$. Top: naive sampling. Bottom: AGS.

Consider now AGS(ϵ, δ). Recall that we are looking at a *fixed* graphlet H_i (which here does *not* denote the graphlet sampled at line 10). Note that $\sum_{\tau=1}^{t} X_\tau$ is exactly the value of c_i after t executions of the main cycle (see line 11). Similarly, note that $\sum_{\tau=1}^{t} P_\tau$ is the value of $g_i \cdot w_i$ after t executions of the main cycle: indeed, if $Y_j^{t-1} = 1$, then at step τ we add to w_i the value $\frac{\sigma_{ij}}{t_j}$ (line 8), while the probability that a sample of T_j yields H_i is exactly $\frac{g_i \sigma_{ij}}{t_j}$. Therefore, after the main cycle has been executed t times, $Z_t = \sum_{\tau=1}^{t}(X_t - P_t)$ is the value of $c_i - g_i w_i$.

Now to the bounds. Suppose that, when AGS(ϵ, δ) returns, $\frac{c_i}{w_i} \geq g_i(1+\epsilon)$, i.e., $c_i(1 - \frac{\epsilon}{1+\epsilon}) \geq g_i w_i$. On the one hand this implies that $c_i - g_i w_i \geq c_i \frac{\epsilon}{1+\epsilon}$, i.e., $Z_t \geq c_i \frac{\epsilon}{1+\epsilon}$; and since upon termination $c_i = \bar{c}$, this means $Z_t \geq \bar{c}\frac{\epsilon}{1+\epsilon}$. On the other hand it implies $g_i w_i \leq c_i(1 - \frac{\epsilon}{1+\epsilon})$, i.e., $\sum_{\tau=1}^{t} P_\tau \leq c_i(1 - \frac{\epsilon}{1+\epsilon})$; again since upon termination $c_i = \bar{c}$, this means $\sum_{\tau=1}^{t} P_\tau \leq \bar{c}(1 - \frac{\epsilon}{1+\epsilon})$. We can then invoke Lemma 1 with $z = \bar{c}\frac{\epsilon}{1+\epsilon}$ and $v = \bar{c}(1 - \frac{\epsilon}{1+\epsilon})$, and since $v + z = \bar{c}$ we get:

$$\Pr\Big[\frac{c_i}{w_i} \geq g_i(1+\epsilon)\Big] \leq \exp\Big[-\frac{(\bar{c}\frac{\epsilon}{1+\epsilon})^2}{2\bar{c}}\Big] \tag{7}$$

$$= \exp\Big[-\frac{\epsilon^2 \bar{c}}{2(1+\epsilon)^2}\Big] \tag{8}$$

but $\frac{\epsilon^2 \bar{c}}{2(1+\epsilon)^2} \geq \frac{\epsilon^2}{2(1+\epsilon)^2}\frac{4}{\epsilon^2}\ln\big(\frac{2s}{\delta}\big) \geq \ln\big(\frac{2s}{\delta}\big)$ and thus the probability above is bounded by $\frac{\delta}{2s}$.

Suppose instead that, when AGS(ϵ, δ) returns, $\frac{c_i}{w_i} \leq g_i(1-\epsilon)$, i.e., $c_i(1 + \frac{\epsilon}{1-\epsilon}) \leq g_i w_i$. On the one hand this implies that $c_i - g_i w_i \geq \frac{\epsilon}{1-\epsilon} c_i$, that is, upon termination we have $-Z_t \geq \frac{\epsilon}{1-\epsilon}\bar{c}$. Obviously $(-Z_t)_{t\geq 0}$ is a martingale too with respect to the filter $(\mathcal{F}_t)_{t\geq 0}$, and therefore Lemma 1 still holds if we replace Z_t with $-Z_t$. Let then $t_0 \leq t$ be the first step where $-Z_{t_0} \geq \frac{\epsilon}{1-\epsilon}\bar{c}$; since $|Z_t - Z_{t-1}| \leq 1$, it must be $-Z_{t_0} < \frac{\epsilon}{1-\epsilon}\bar{c} + 1$. Moreover $\sum_{\tau=1}^{t} X_\tau$ is nondecreasing in t, so $\sum_{\tau=1}^{t_0} X_\tau \leq \bar{c}$. It follows that $\sum_{\tau=1}^{t_0} P_\tau = -Z_{t_0} + \sum_{\tau=1}^{t_0} X_\tau < \frac{\epsilon}{1-\epsilon}\bar{c} + 1 + \bar{c} = \frac{1}{1-\epsilon}\bar{c} + 1$. Invoking again Lemma 1 with $z = \frac{\epsilon}{1-\epsilon}\bar{c}$ and $v = \frac{1}{1-\epsilon}\bar{c} + 1$, we obtain:

$$\Pr\Big[\frac{c_i}{w_i} \leq g_i(1-\epsilon)\Big] \leq \exp\Big[-\frac{(\bar{c}\frac{\epsilon}{1-\epsilon})^2}{2(\frac{1+\epsilon}{1-\epsilon}\bar{c}+1)}\Big] \tag{9}$$

$$\leq \exp\Big[-\frac{\epsilon^2 \bar{c}^2}{2(1+\bar{c})}\Big] \tag{10}$$

but since $\bar{c} \geq 4$ then $\frac{\bar{c}}{1+\bar{c}} \geq \frac{4}{5}$ and so $\frac{\epsilon^2\bar{c}^2}{2(1+\bar{c})} \geq \frac{2\epsilon^2\bar{c}}{5}$. By replacing $\bar{c}$ we get $\frac{2\epsilon^2\bar{c}}{5} \geq \frac{2\epsilon^2}{5}\frac{4}{\epsilon^2}\ln\big(\frac{2s}{\delta}\big) > \ln\big(\frac{2s}{\delta}\big)$ and thus once again the probability of deviation is bounded by $\frac{\delta}{2s}$.

By a simple union bound, the probability that $\frac{c_i}{w_i}$ is not within a factor $(1\pm\epsilon)$ of g_i is at most $\frac{\delta}{s}$. The theorem follows by a union bound on all $i \in [s]$.

C. PROOF OF THEOREM 6

For each $i \in [s]$ and each $j \in [\varsigma]$ let a_{ji} be the probability that sample(T_j) returns a copy of H_i. Note that $a_{ji} = g_i \sigma_{ij}/t_j$, the fraction of colorful copies of T_j that span a copy of H_i. Our goal is to allocate, for each T_j, the number x_j of calls to sample(T_j), so that (1) the total number of calls $\sum_j x_j$ is minimised and (2) each H_i appears at least $\bar{c}$ times in expectation. Formally, let $\mathbf{A} = (a_{ji})^\intercal$, so that columns correspond to treelets T_j and rows to graphlets H_i, and let $\mathbf{x} = (x_1, \ldots, x_\varsigma) \in \mathbb{N}^\varsigma$. We obtain the following integer program:

$$\begin{cases} \min \mathbf{1}^\intercal \mathbf{x} \\ \text{s.t. } \mathbf{A}\mathbf{x} \geq \bar{c}\,\mathbf{1} \\ \quad \mathbf{x} \in \mathbb{N}^\varsigma \end{cases}$$

We now describe the natural greedy algorithm for this problem; it turns out that this is precisely AGS. The algorithm proceeds in discrete time steps. Let $\mathbf{x}^0 = \mathbf{0}$, and for all $t \geq 1$ denote by $\mathbf{x}^t$ the partial solution after t steps. The vector $\mathbf{A}\mathbf{x}^t$ is an s-entry column whose i-th entry is the expected number of occurrences of H_i drawn using the sample allocation given by $\mathbf{x}^t$. We define the vector of residuals at time t as $\mathbf{c}^t = \max(\mathbf{0}, \mathbf{c} - \mathbf{A}\mathbf{x}^t)$, and for compactness we let $c^t = \mathbf{1}^\intercal \mathbf{c}^t$. Note that $\mathbf{c}^0 = \bar{c}\,\mathbf{1}$ and $c^0 = s\bar{c}$. Finally, we let

vivid account of this difference below). Inevitably, the error spreads out with k; recall that the total number of distinct 8-graphlets is over 10^4.

Number of accurate graphlets. For a complementary view, we consider the number of graphlets whose estimate is within $\pm 50\%$ of the ground-truth value (Figure 8). This number easily often reaches the thousands, and for $k = 9$ even hundreds of thousands (note that the plot is in log-scale). We remind the reader that all this is carried out in minutes or, in the worst case, in two hours. Alternatively, we can look at these numbers in relative terms, that is, as a fraction of the total number of distinct graphlets in the ground truth (Figure 8). On all graphs except BERKSTAN, this ratio is over 90% of graphlets for $k = 6$, over 75% of graphlets for $k = 7$, and over 50% of graphlets for $k = 8$, for either naive sampling or AGS. The choice of 50% is just to deliver the picture, but we remark that such an error is achieved simultaneously for thousand of distinct graphlets whose counts differ by many orders of magnitude.

5.3 Breaking the sampling barrier with AGS

Finally, we show how AGS breaks the additive error barrier of naive sampling. The best example is the YELP graph. Look again at Figure 8. For $k = 8$, over 99.9996% of the k-graphlets are stars. Consequently, in our experiments naive sampling finds only the star graphlet. This means that naive sampling gives accurate estimates for only 1 graphlet, that is, 0.01% of the total. Put in other terms, it misses 9999 out of 10000 graphlets. Instead, AGS returns fair estimates for 9645 graphlets, that is, 87% of the total.

The contrast is even sharper if we look at the frequency of the graphlets found by the two algorithms. These are shown in Figure 9 (to filter out noise, we consider only graphlets appearing in at least 10 samples). Naive sampling finds only graphlets with frequency at least 99.9996% (that is, again, the star). AGS finds graphlets with frequency below 10^{-21}, that is, seven orders of magnitude smaller. To convey the idea, to find those graphlets naive sampling would need $\approx 3 \cdot 10^3$ years even if running at 10^9 samples per second.

Let us make a final remark. On some graphs, AGS is slightly worse than naive sampling. This is expected: AGS is designed for skewed graphlet distributions, and loses ground on flatter ones. As a sanity check, we computed the ℓ_2 norm of the graphlet distributions. The three graphs where AGS beats naive sampling by a largest margin, BERKSTAN, YELP and TWITTER, have for all k the highest ℓ_2 norms ($> .99$). Symmetrically, FACEBOOK, DBLP and FRIENDSTER, have for all k the three lowest ℓ_2 norms, and there AGS performs slightly worse than naive sampling.

6. CONCLUSIONS

We have shown how the color coding technique, despite its simplicity (or perhaps thanks to it), can be harnessed to scale motif counting to truly massive graphs. Thanks to color coding we can mine motifs in graphs with billions of edges, with approximation guarantees previously out of reach, using just ordinary hardware. These results can be seen as the first steps in the direction of *unexpensive large-scale motif mining*. We believe it may be possible to scale even further, by devising appropriate optimizations and algorithmic solutions; and especially by reducing the space usage, which is still a bottleneck of this approach.

APPENDIX

A. PROOF OF THEOREM 3

We use a concentration bound for dependent random variables from [13]. Let $\mathcal{V}_i$ be the set of copies of H_i in G. For any $h \in \mathcal{V}_i$ let X_h be the indicator random variable of the event that h becomes colorful. Let $c_i = \sum_{h \in \mathcal{V}_i} X_h$; clearly $\mathbb{E}[c_i] = p_k |\mathcal{V}_i| = p_k n_i$. Note that for any $h_1, h_2 \in \mathcal{V}_i$, X_{h_1}, X_{h_2} are independent if and only if $|V(h_1) \cap V(h_2)| \leq 1$, i.e., if h_1, h_2 share at most one node. For any $u, v \in G$ let then $g(u,v) = |\{h \in \mathcal{V}_i : u, v \in h\}|$, and define $\chi_k = 1 + \max_{u,v \in G} g(u,v)$. By standard counting argument one can see that $\max_{u,v \in G} g(u,v) \leq (k-1)!\Delta^{k-2} - 1$ and thus $\chi_k \leq (k-1)!\Delta^{k-2}$. The bound then follows immediately from Theorem 3.2 of [13] by setting $t = \epsilon c_i$, $(b_\alpha - a_\alpha) = 1$ for all $\alpha = h \in \mathcal{V}_i$, and $\chi^*(\Gamma) \leq \chi_k \leq (k-1)!\Delta^{k-2}$.

B. PROOF OF THEOREM 4

The proof requires a martingale analysis, since the distribution from which we draw the graphlets changes over time. We use a martingale tail inequality originally from [14] and stated (and proved) in the following form in [3, p. 1476]:

THEOREM 7. ([3], Theorem 2.2). *Let $(Z_0, Z_1, \ldots)$ be a martingale with respect to the filter $(\mathcal{F}_\tau)_{t \geq 0}$. Suppose that $Z_{\tau+1} - Z_\tau \leq M$ for all τ, and write $V_t = \sum_{\tau=1}^{t} \mathrm{Var}[Z_\tau | \mathcal{F}_{\tau-1}]$. Then for any $z, v > 0$ we have:*

$$\Pr\left[\exists\, t : Z_t \geq Z_0 + z, V_t \leq v\right] \leq \exp\left[-\frac{z^2}{2(v + Mz)}\right] \quad (5)$$

We now plug into the formula of Theorem 7 the appropriate quantities. In what follows we *fix* a graphlet H_i and analyse the concentration of its estimate. Unless necessary, we drop the index i from the notation.

A. For $t \geq 1$ let X_t be the indicator random variable of the event that H_i is the graphlet sampled at step t (line 10 of AGS).

B. For $t \geq 0$ let Y_j^t be the indicator random variable of the event, at the end of step t, the treelet to be sampled at the next step is T_j.

C. For $t \geq 0$ let $\mathcal{F}_t$ be the event space generated by the random variables $Y_j^\tau : j \in [\varsigma]$, $\tau = 0, \ldots, t$. For any random variable Z, then, $\mathbb{E}[Z \,|\, \mathcal{F}_t] = \mathbb{E}[Z \,|\, Y_j^\tau : j \in [\varsigma], \tau = 0, \ldots, t]$, and $\mathrm{Var}[Z \,|\, \mathcal{F}_t]$ is defined analogously.

D. For $t \geq 1$ let $P_t = \mathbb{E}[X_t | \mathcal{F}_{t-1}]$ be the probability that the graphlet sampled at the t-th invocation of line 10 is H_i, as a function of the events up to time $t-1$. It is immediate to see that $P_t = \sum_{j=1}^{\varsigma} Y_j^{t-1} a_{ji}$.

E. Let $Z_0 = 0$, and for $t \geq 1$ let $Z_t = \sum_{\tau=1}^{t} (X_t - P_t)$. Now, $(Z_t)_{t \geq 0}$ is a martingale with respect to the filter $(\mathcal{F}_t)_{t \geq 0}$, since Z_t is obtained from Z_{t-1} by adding X_t and subtracting P_t which is precisely the expectation of X_t w.r.t. $\mathcal{F}_{t-1}$.

F. Let $M = 1$, since $|Z_{t+1} - Z_t| = |X_{t+1} - P_t| \leq 1$ for all t.

Finally, notice that $\mathrm{Var}[Z_t | \mathcal{F}_{t-1}] = \mathrm{Var}[X_t | \mathcal{F}_{t-1}]$, since again $Z_t = Z_{t-1} + X_t - P_t$, and both Z_{t-1} and P_t are a function of $\mathcal{F}_{t-1}$, so their variance w.r.t. $\mathcal{F}_{t-1}$ is 0. Now, $\mathrm{Var}[X_t | \mathcal{F}_{t-1}] = P_t(1 - P_t) \leq P_t$; and therefore we have $V_t = \sum_{\tau=1}^{t} \mathrm{Var}[Z_\tau \,|\, \mathcal{F}_{\tau-1}] \leq \sum_{\tau=1}^{t} P_\tau$. Then by Theorem 7:

LEMMA 1. *For all $z, v > 0$ we have*

$$\Pr\left[\exists\, t : Z_t \geq z, \sum_{\tau=1}^{t} P_\tau \leq v\right] \leq \exp\left[-\frac{z^2}{2(v+z)}\right] \quad (6)$$

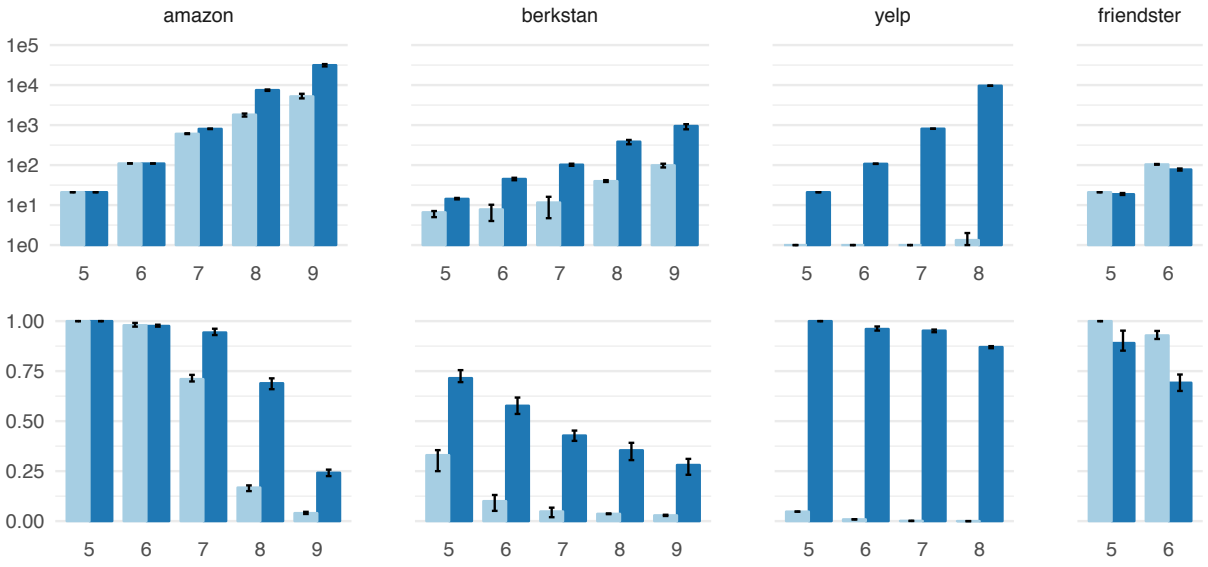


Figure 8: graphlet counts with error within ±50% (dark bars = AGS). Top: absolute number. Bottom: as a fraction.

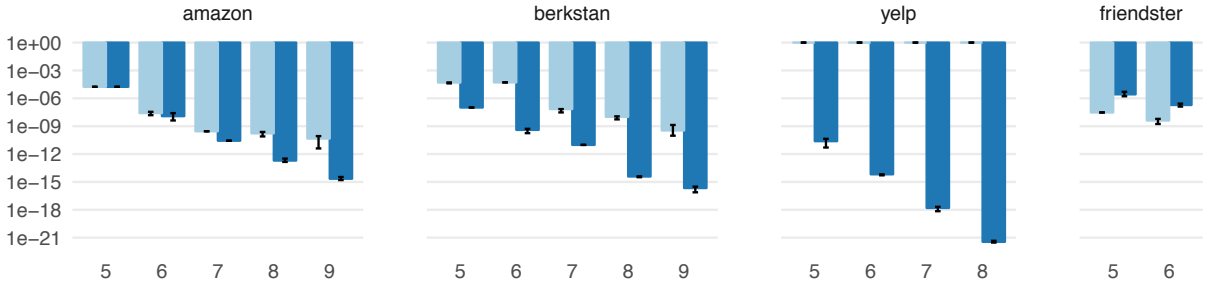


Figure 9: frequency of the rarest graphlet appearing in 10 or more samples (dark bars = AGS).

$U^t = \{i : c_i^t > 0\}$; this is the set of graphlets not yet covered at time t, and clearly $U^0 = [s]$.

At the t-th step the algorithm chooses the T_{j^*} such that sample(T_{j^*}) spans an uncovered graphlet with the highest probability, by computing:

$$j^* := \arg\max_{j=1,\dots,\varsigma} \sum_{i \in U_t} a_{ji} \tag{11}$$

It then lets $\mathbf{x}^{t+1} = \mathbf{x} + \mathbf{e}_{j^*}$, where $\mathbf{e}_{j^*}$ is the indicator vector of j^*, and updates $\mathbf{c}^{t+1}$ accordingly. The algorithm stops when $U^t = \emptyset$, since then $\mathbf{x}^t$ is a feasible solution. We prove:

LEMMA 2. *Let z be the cost of the optimal solution. Then the greedy algorithm returns a solution of cost $O(z \ln(s))$.*

PROOF. Let $w_j^t = \sum_{i \in U_t} a_{ji}$ (note that this is a *treelet* weight). For any $j \in [\varsigma]$ denote by $\Delta_j^t = c^t - c^{t+1}$ the decrease in overall residual weight we would obtain if $j^* = j$. Note that $\Delta_j^t \le w_j^t$. We consider two cases.

Case 1: $\Delta_{j^*}^t < w_{j^*}^t$. This means for some $i \in U_t$ we have $c_i^{t+1} = 0$, implying $i \notin U_{t+1}$. In other terms, H_i becomes covered at time $t+1$. Since the algorithm stops when $U_t = \emptyset$, this case occurs at most $|U^0| = s$ times.

Case 2: $\Delta_{j^*}^t = w_{j^*}^t$. Suppose then that the original problem admits a solution with cost z. Obviously, the "residual" problem where $\mathbf{c}$ is replaced by $\mathbf{c}^t$ admits a solution of cost z, too. This implies the existence of $j \in [\varsigma]$ with $\Delta_j^t \ge \frac{1}{z} c^t$, for otherwise any solution for the residual problem would have cost $> z$. But by the choice of j^* it holds $\Delta_{j^*} = w_{j^*}^t \ge w_j^t \ge \Delta_j^t$ for any j, hence $\Delta_{j^*}^t \ge \frac{1}{z} c^t$. Thus by choosing j^* we get $c^{t+1} \le (1 - \frac{1}{z}) c^t$. After running into this case ℓ times, the residual cost is then at most $c^0 (1 - \frac{1}{z})^\ell$.

Note that $\ell + s \ge c^0 = s \cdot \bar{c}$ since at any step the overall residual weight can decrease by at most 1. Therefore the algorithm performs $\ell + s = O(\ell)$ steps overall. Furthermore, after $\ell + s$ steps we have $c^{\ell+s} \le s\bar{c} e^{-\frac{\ell}{z}}$, and by picking $\ell = z \ln(2s)$ we obtain $c^{\ell+s} \le \frac{\bar{c}}{s}$, and therefore each one of the s graphlets receives weight at least $\frac{\bar{c}}{2}$. Now, if we replace $\bar{c}\,\mathbf{1}$ with $2\bar{c}\,\mathbf{1}$ in the original problem, the cost of the optimal solution is at most $2z$, and in $O(z \ln(s))$ steps the algorithm finds a cover where each graphlet has weight at least $\bar{c}$. □

Now, note that the treelet index j^* given by Equation 11 remains unchanged as long as U_t remains unchanged. Therefore we need to recompute j^* only when some new graphlet exits U_t, i.e., becomes covered. In addition, we do not need each value a_{ji}, but only their sum $\sum_{i \in U_t} a_{ji}$. This is precisely the quantity that AGS estimates at line 14. Theorem 6 follows immediately as a corollary.

D. REFERENCES

[1] A. F. Abdelzaher, A. F. Al-Musawi, P. Ghosh, M. L. Mayo, and E. J. Perkins. Transcriptional network growing models using motif-based preferential attachment. *Frontiers in Bioengineering and Biotechnology*, 3:157, 2015.

[2] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *Proc. of ICDM*, pages 1–10, 2015.

[3] N. Alon, O. Gurel-Gurevich, and E. Lubetzky. Choice-memory tradeoff in allocations. *The Annals of Applied Probability*, 20(4):1470–1511, 2010.

[4] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[5] T. Baroudi, R. Seghir, and V. Loechner. Optimization of triangular and banded matrix operations using 2d-packed layouts. *ACM TACO*, 14(4):55:1–55:19, 2017.

[6] M. A. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *Proc. of ICDM*, pages 91–100, 2012.

[7] M. Bressan. Counting subgraphs via DAG tree decompositions. In *Proc. of IPEC*, 2019. (To appear).

[8] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Counting graphlets: Space vs time. In *Proc. of ACM WSDM*, pages 557–566, 2017.

[9] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Motif counting beyond five nodes. *ACM TKDD*, 12(4), 2018.

[10] V. T. Chakaravarthy, M. Kapralov, P. Murali, F. Petrini, X. Que, Y. Sabharwal, and B. Schieber. Subgraph counting: Color coding beyond trees. In *Proc. of IEEE IPDPS*, pages 2–11, 2016.

[11] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006.

[12] X. Chen, Y. Li, P. Wang, and J. C. S. Lui. A general framework for estimating graphlet statistics via random walk. *PVLDB*, 10(3):253–264, 2016.

[13] D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[14] D. A. Freedman. On tail probabilities for martingales. *The Annals of Probability*, 3(1):100–118, 1975.

[15] G. Han and H. Sethu. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. *Proc. of ICDM*, pages 181–190, 2016.

[16] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using Turán's theorem. In *Proc. of WWW*, pages 441–449, 2017.

[17] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proc. of WWW*, pages 495–505, 2015.

[18] B. D. McKay and A. Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94–112, 2014.

[19] R. Otter. The number of trees. *Annals of Mathematics*, pages 583–599, 1948.

[20] A. Pinar, C. Seshadhri, and V. Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *Proc. of WWW*, pages 1431–1440, 2017.

[21] G. M. Slota and K. Madduri. Fast approximate subgraph counting and enumeration. In *Proc. of ICPP*, pages 210–219, 2013.

[22] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal. TriÈst: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Trans. Knowl. Discov. Data*, 11(4):43:1–43:50, June 2017.

[23] N. H. Tran, K. P. Choi, and L. Zhang. Counting motifs in the human interactome. *Nature Communications*, 4(2241), 2013.

[24] M. D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Software Eng.*, 17(9):972–975, 1991.

[25] P. Wang, J. C. S. Lui, B. Ribeiro, D. Towsley, J. Zhao, and X. Guan. Efficiently estimating motif statistics of large networks. *ACM TKDD*, 9(2):8:1–8:27, 2014.

[26] P. Wang, J. Tao, J. Zhao, and X. Guan. Moss: A scalable tool for efficiently sampling and counting 4- and 5-node graphlets. *CoRR*, abs/1509.08089, 2015.

[27] P. Wang, X. Zhang, Z. Li, J. Cheng, J. C. S. Lui, D. Towsley, J. Zhao, J. Tao, and X. Guan. A fast sampling method of exploring graphlet degrees of large directed and undirected graphs. *CoRR*, abs/1604.08691, 2016.

[28] Ö. N. Yaveroğlu, N. Malod-Dognin, D. Davis, Z. Levnajic, V. Janjic, R. Karapandza, A. Stojmirovic, and N. Pržulj. Revealing the hidden language of complex networks. *Scientific Reports*, 4:4547 EP –, 04 2014.

[29] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich. Local higher-order graph clustering. In *Proc. of ACM KDD*, pages 555–564, New York, NY, USA, 2017. ACM.

[30] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *Proc. of ICPP*, pages 594–603, 2010.

Data Mining and Knowledge Discovery (2022) 36:414–447
https://doi.org/10.1007/s10618-021-00802-3

Sequential stratified regeneration: *MCMC* for large state spaces with an application to subgraph count estimation

Carlos H. C. Teixeira[1] · Mayank Kakodkar[2] · Vinícius Dias[1,3] · Wagner Meira Jr.[1] · Bruno Ribeiro[2]

Received: 23 November 2020 / Accepted: 29 September 2021 / Published online: 4 January 2022
© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2021

Abstract

This work considers the general task of estimating the sum of a bounded function over the edges of a graph, given neighborhood query access and where access to the entire network is prohibitively expensive. To estimate this sum, prior work proposes Markov chain Monte Carlo (*MCMC*) methods that use random walks started at some seed vertex and whose equilibrium distribution is the uniform distribution over all edges, eliminating the need to iterate over all edges. Unfortunately, these existing estimators are not scalable to massive real-world graphs. In this paper, we introduce Ripple, an *MCMC*-based estimator that achieves unprecedented scalability by stratifying the Markov chain state space into ordered strata with a new technique that we denote *sequential stratified regenerations*. We show that the Ripple estimator is consistent, highly parallelizable, and scales well. We empirically evaluate our method by applying Ripple to the task of estimating connected, induced subgraph counts given some input graph. Therein, we demonstrate that Ripple is accurate and can estimate counts of up to 12-node subgraphs, which is a task at a scale that has been considered unreachable, not only by prior *MCMC*-based methods but also by other sampling approaches. For instance, in this target application, we present results in which the Markov chain state space is as large as 10^{43}, for which Ripple computes estimates in less than 4 h, on average.

Responsible editor: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautman.

Carlos H. C. Teixeira and Mayank Kakodkar have contributed to this work equally.

✉ Carlos H. C. Teixeira
carlos@dcc.ufmg.br

1 Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

2 Purdue University, West Lafayette, USA

3 Universidade Federal de Ouro Preto, Ouro Preto, Brazil

Keywords Markov Chain Monte Carlo · Random walk · Regenerative sampling · Motif analysis · Subgraph counting · Graph mining

1 Introduction

This work considers the following general task: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple graph, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, and $\mathcal{E}$ contains at most a single edge between any pair of vertices and no self-loops. Our goal is to efficiently estimate the sum of a bounded function over all the edges of $\mathcal{G}$,

$$\mu(\mathcal{E}) = \sum_{(u,v)\in\mathcal{E}} f(u, v) \tag{1}$$

where $f : \mathcal{E} \to \mathbb{R}$, $f(\cdot) < B$ is a bounded function for some constant $B \in \mathbb{R}$ under the following query model from Avrachenkov et al. (2016).

Assumption 1 (*Query Model*) Assume we are given arbitrary seed vertices and can query the neighborhood $\mathbf{N}(u) \triangleq \{v \in \mathcal{V} : (u, v) \in \mathcal{E}\}$ for any vertex $u \in \mathcal{V}$ such that accessing the entire graph $\mathcal{G}$ is prohibitively expensive.

This setting arises in the subgraph counting problem (Sect. 4), where $|\mathcal{V}| \gg 10^{10}$ (Table 1) and we assume O(1) seeds. Simple Monte Carlo procedures are not useful because random vertex and edge queries are not directly available, and reservoir sampling would require iteration over all edges. Standard Markov chain Monte Carlo (*MCMC*) methods cannot estimate the quantity in Eq. (1) and are limited to estimate $\mu(\mathcal{E})/|\mathcal{E}|$, because $|\mathcal{E}|$ in our task is unknown (Ribeiro and Towsley 2012). Generally, under Assumption 1, Eq. (1) is estimated using specialized *MCMC* estimators that use a random-walk-like Markov chain that has a uniform distribution over the edges $\mathcal{E}$ as its equilibrium distribution. However, these estimators (Avrachenkov et al. 2016) are impractical in large graphs because their running time is $O(|\mathcal{E}|)$.

Traditional *MCMC* methods are limited by their reliance on the Markov chain on $\mathcal{G}$ reaching equilibrium or *burning in*. Because the rate of convergence to equilibrium depends on the spectral gap (Aldous and Fill 2002), a significant number of Markov chain steps is needed to *burn in* in order to produce accurate estimates of Eq. (1), particularly in large graphs. Parallel approaches that divide the state space into disjoint "chunks", which are to be processed in parallel (Wilkinson 2006; Neiswanger et al. 2014), offer no respite because we cannot access the entire graph. In fact, $\mathcal{G}$ may not even have disconnected components (i.e., disjoint chunks) that can be parallelized. Therefore, traditional *MCMC* on $\mathcal{G}$ offers no meaningful parallelization opportunities and running times may be arbitrarily long.

Contributions. This work introduces *sequential stratified regeneration* (Ripple), a novel parallel *MCMC* technique that expands the application frontier of *MCMC* to large state-space graphs $\mathcal{G}$. Ripple stratifies the underlying Markov chain state space into ordered strata that need not be disjoint chunks, rather, they need to be connected. Markov chain regeneration (Nummelin 1978) is then used to compute estimates in each stratum sequentially, using a recursive method, which improves regeneration

frequencies and reduces variance. Ripple offers an unprecedented level of efficiency and parallelism for *MCMC* sampling on large state-space graphs while retaining the benefits of *MCMC*-based algorithms, such as low memory demand (polynomial w.r.t. output).

Surprisingly, the parallelism of Ripple comes from the regeneration rather than the stratification: the strata's job is to keep regeneration times short. We demonstrate that the estimates obtained by Ripple are consistent, among other theoretical guarantees. In addition, we empirically show the power of Ripple in a real-world application by specializing Eq. (1) to subgraph counting in multi-million-node attributed graphs–to the best of our knowledge, a task at a scale that has been thought unreachable by any other *MCMC* method. Our specific contributions to the subgraph counting problem include streaming-based optimizations coupled with a parallel reservoir sampling algorithm, novel efficiency improvements to the random walk on the $-HON$ (Wang et al. 2014) and a theoretical analysis of scalability in terms of running time and memory w.r.t. the subgraph size, verified empirically on large datasets.

2 Background and prior work

The *MCMC* random-walk-like Markov chain over the graph $\mathcal{G}$ is defined as:

Definition 1 (*Random Walk on* $\mathcal{G}$) Given a simple graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a simple random walk is a time-homogenous Markov chain $\boldsymbol{\Phi}$ with state space $\mathcal{V}$ and transition probability $p_{\boldsymbol{\Phi}}(u, v) = 1/\mathbf{d}(u)$, when $(u, v) \in \mathcal{E}$ and 0 otherwise, where $\mathbf{d}(u) = |\mathbf{N}(u)|$ is the degree of u in $\mathcal{G}$ and $\mathbf{N}(u) = \{v : (u, v) \in \mathcal{E}\}$ is the neighborhood of u.

It is easy to check that the above random walk can be sampled under Assumption 1 and that on a connected graph, this walk samples edges uniformly at random in a steady state (check "Appendix B.1" for details). Our notation is summarized in "Appendix A".

2.1 Regenerations in discrete Markov Chains

The rate of convergence to stationarity of the random walk $\boldsymbol{\Phi}$ from Definition 1 depends on the spectral gap[1] (Aldous and Fill 2002). As such, practitioners are encouraged to run a single, long sample path, which prevents them from splitting the task among multiple cores. Usually, because the spectral gap is unknown or loosely bounded, practitioners use various diagnostics to *eyeball* if the chain has mixed (Rosenthal 1995). The variance of an estimate computed from a stationary chain (Ribeiro and Towsley 2012) also depends on the spectral gap.

A solution to the above problems is to *split* (Nummelin 1978) the Markov chain using regenerations. Discrete Markov chains regenerate every time they enter a fixed state, which is referred to as a regeneration point. This naturally yields the definition of a *random walk tour* (*RWT*).

[1] The spectral gap is defined as $\delta = 1 - \max\{|\lambda_2|, |\lambda_{|\mathcal{V}|}|\}$, where λ_i denotes the i-th eigenvalue of the transition probability matrix of $\boldsymbol{\Phi}$.

Definition 2 (*RWTover* $\boldsymbol{\Phi}$) Given a time-homogenous Markov chain $\boldsymbol{\Phi}$ over finite state space $\mathcal{V}$ and a fixed point $x_0 \in \mathcal{V}$, an *RWT* $\mathbf{X} = (X_i)_{i=1}^{\xi}$ is a sequence of states visited by $\boldsymbol{\Phi}$ between two consecutive visits to x_0, that is, $X_1 = x_0$ and $\xi = \min\{i > 1 : X_{i+1} = x_0\}$ is the first return time to x_0.

Because of the strong Markov property (Bremaud 2001, Chap-2,Thm-7.1), *RWT*s started at x_0 are i.i.d. and can be used to estimate $\mu(\mathcal{E})$ from Eq. (1) when $|\mathcal{E}|$ is unknown (Avrachenkov et al. 2016, 2018; Teixeira et al. 2018; Savarese et al. 2018; Cooper et al. 2016; Massoulié et al. 2006).

Lemma 1 (*RWT* Estimate) *Given the graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *and the random walk* $\boldsymbol{\Phi}$ *from Definition 1, consider* $f : \mathcal{E} \to \mathbb{R}$ *bounded by* B*, and* $\mathcal{T}$*, a set of* m RWT*s started at* $x_0 \in \mathcal{V}$ *(Definition 2) sampled in a parallel* z *core environment assuming each core samples an equal number of tours. Then,*

$$\hat{\mu}_*(\mathcal{T};\, f, \mathcal{G}) = \frac{\mathbf{d}(x_0)}{2m} \sum_{\mathbf{X} \in \mathcal{T}} \sum_{j=1}^{|\mathbf{X}|} f(X_j, X_{j+1})\,, \tag{2}$$

is an unbiased and consistent estimator of $\mu(\mathcal{E}) = \sum_{(u,v)\in\mathcal{E}} f(u, v)$ *if* $\mathcal{G}$ *is connected, where each* X_j *refers to the* j*th state in the* RWT $\mathbf{X} \in \mathcal{T}$.

The expected running time for sampling m *tours is* $\mathrm{O}\left(m/z\frac{2|\mathcal{E}|}{\mathbf{d}(x_0)}\right)$*, and when* $\mathcal{G}$ *is non-bipartite, the variance of the estimate is bounded as*

$$\mathrm{Var}\left(\hat{\mu}_*(\mathcal{T})\right) \leq \frac{3B^2}{m}\frac{|\mathcal{E}|^2}{\delta(\boldsymbol{\Phi})}\,, \tag{3}$$

where $\delta(\boldsymbol{\Phi})$ *is the spectral gap as defined in the beginning of this section.*

The *RWT* Estimate can be considered a Las Vegas transformation of *MCMC*, which takes random time but yields unbiased estimates of objectives, such as Eq. (1). The parallelism in the expected running time in Lemma 1 is directly due to the independence of *RWT*s. Moreover, confidence intervals for the *RWT* Estimate can be computed, because $\sqrt{m}\frac{\hat{\mu}_*(\mathcal{T})-\mu(\mathcal{E})}{\hat{\sigma}(\mathcal{T})}$ approaches the standard normal distribution for sufficiently large m, where $\hat{\sigma}(\mathcal{T})^2$ is the empirical variance of $\hat{\mu}_*(\mathbf{X})$, the *RWT* Estimate computed using an individual tour $\mathbf{X} \in \mathcal{T}$.

2.2 Improving the regeneration frequency

From Lemma 1, it is clear that increasing the degree of the regeneration point $\mathbf{d}(x_0)$ and spectral gap $\delta(\boldsymbol{\Phi})$ and decreasing $|\mathcal{E}|$ reduces the variance as well as the running time of the *RWT* Estimate. Avrachenkov et al. (2016) showed that using the *supernode* in a contracted graph as a regeneration point achieves the above reductions.

Definition 3 (*Contracted Graph* Avrachenkov et al. 2016) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from Definition 1 and a set of vertices $I \subset \mathcal{V}$, a contracted graph is a multigraph $\mathcal{G}_I$ formed by collapsing I into a single node ζ_I. The vertex set of $\mathcal{G}_I$ is then given by

$\mathcal{V}\backslash I \cup \{\zeta_I\}$, and its edge multiset is obtained by conditionally replacing each endpoint of each edge with ζ_I if it is a member of I and removing self-loops on ζ_I. We refer to the set I and the vertex ζ as the supernode.

Contractions benefit *RWT*s because the supernode degree $\mathbf{d}_{\mathcal{G}_I}(\zeta_I)$ in $\mathcal{G}_I$ and the spectral gap $\delta(\boldsymbol{\Phi}_I)$ of the random walk on the contracted graph increase monotonically with $|I|$ (Avrachenkov et al. 2016). Moreover, *RWT*s can be sampled on $\mathcal{G}_I$ without explicit construction, as we see next.

Remark 1 Let the multi-set $\mathbf{N}(\zeta_I) \triangleq \uplus_{u\in I}\mathbf{N}_{\mathcal{G}}(u)\backslash I$ be the neighborhood of the supernode in $\mathcal{G}_I$ from Definition 3. Let $\boldsymbol{\Phi}_I$ be the simple random walk on $\mathcal{G}_I$. An *RWT* $(X_i)_{i=1}^{\xi}$ on $\boldsymbol{\Phi}_I$ from ζ_I is sampled by setting $X_1 = \zeta_I$, sampling X_2 u.a.r. from $\mathbf{N}(\zeta_I)$ and subsequently sampling transitions from $\boldsymbol{\Phi}$ until the chain enters I, i.e., $\xi = \min\{i > 1 \colon X_{\xi+1} \in I\}$.

This construction naturally stratifies $\mathcal{E}$ and decomposes $\mu(\mathcal{E})$ as $\mu(\mathcal{E}_*)+\mu(\mathcal{E}\backslash\mathcal{E}_*)$, where we can exactly compute the $\mu(\mathcal{E}_*)$ and compute an *RWT* Estimate of $\mu(\mathcal{E}\backslash\mathcal{E}_*)$ on the contracted graph. However, to compute the supernode degree, $\mathbf{d}_{\mathcal{G}_I}(\zeta_I)$; furthermore, to sample from $\mathbf{N}(\zeta_I)$, we need to enumerate the set of the edges incident on I in $\mathcal{G}$ given by $\mathcal{E}_* \subset \mathcal{E}$. As such, a massive supernode I (which is crucial when $|\mathcal{E}|$ is large) makes enumerating $\mathcal{E}_*$ prohibitively expensive. We overcome these issues and gain additional control over regenerations by further stratifying $|\mathcal{E}|$.

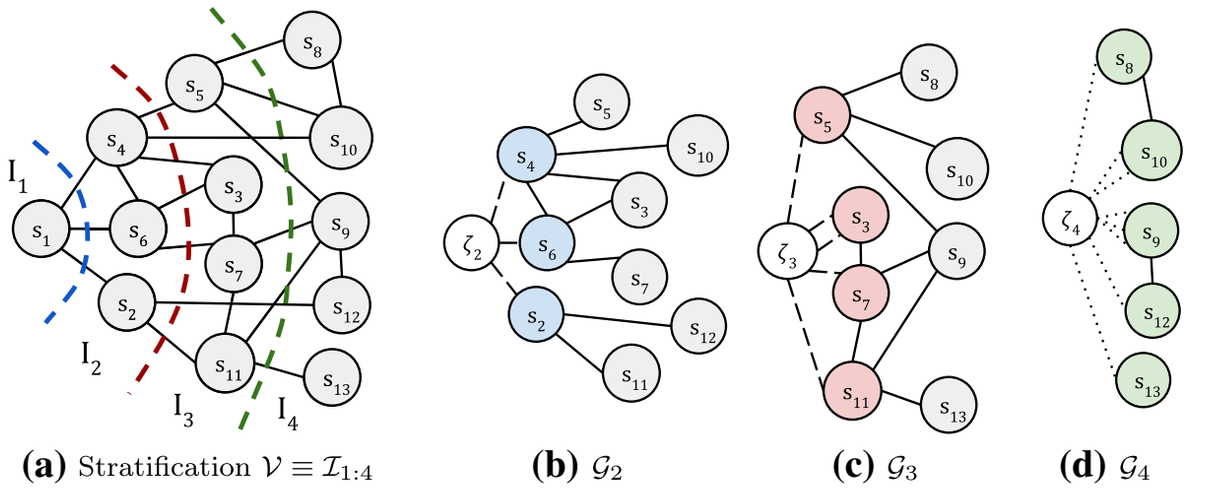
3 Sequential stratified regenerations

Ripple controls regeneration times through a *sequential stratification* of the vertices and edges of $\mathcal{G}$ into ordered strata as illustrated in Fig. 1, which allows us to control the regeneration frequency and the *RWT* Estimate variance. For each stratum, we then construct a graph in which the supernode is created by collapsing all prior strata, from which *RWT*s can be sampled. We use the *RWT*s from the previous strata to estimate the degree of and sample transitions from the supernode. The core idea is described in two steps: Sect. 3.1 details the stratification and conditions that it needs to satisfy and Sect. 3.2 describes the recursion. Finally, we show that the estimator bias converges to zero asymptotically in the number of tours. Particularly for subgraph counting, we show that Ripple's time complexity is independent of the (higher-order) graph size ($|\mathcal{E}|$) and only depends polynomially on the diameter and maximum degree of the *input* graph and the subgraph size (Sect. 4).

3.1 Sequential stratification

Consider the following vertex and edge stratification procedure.

Definition 4 *(Sequential Stratification)* Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ from Definition 1, a function $\rho \colon \mathcal{V} \to \{1, \ldots, R\}$ induces the stratification $(\mathcal{I}_r, \mathcal{J}_r)_{r=1}^{R}$ if $s \in \mathcal{I}_{\rho(s)}$, for each $s \in \mathcal{V}$, and $(u, v) \in \mathcal{J}_{\min(\rho(u),\rho(v))}$, for each $(u, v) \in \mathcal{E}$.



(a) Stratification $\mathcal{V} \equiv \mathcal{I}_{1:4}$ **(b)** $\mathcal{G}_2$ **(c)** $\mathcal{G}_3$ **(d)** $\mathcal{G}_4$

Fig. 1 **a** A simple graph $\mathcal{G}$ that is stratified into four strata $\{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_4\}$. **b**–**d** The second, third and fourth graph strata constructed by Definition 5. In the multi-graph $\mathcal{G}_2$ (Fig. 1b), vertices in $\mathcal{I}_1$ are collapsed into ζ_2 and only edges incident on $\mathcal{I}_2$ are preserved. The edge set therefore contains $\mathcal{J}_2$ and the edges between ζ_2 and $\mathcal{I}_2$. Consequently, self-loops on ζ_2 and edges between $\mathcal{I}_{3:4}$ are absent. **c**, **d** follow suit. In each stratum $\mathcal{G}_r$, *RWT*s from ζ_r are started by sampling u.a.r. from the dotted edges and estimates are computed over the solid edges

Note that these strata are pairwise disjoint and their union is the set of vertices and edges of the graph. Next, we describe the contracted graph over which *RWT*s are to be sampled in each stratum.

Definition 5 *(r-th Graph Stratum)* Let $\mathcal{A}_{i:j} \triangleq \cup_{x=i}^{j} \mathcal{A}_x$ be defined for any ordered tuple of sets. Let $(\mathcal{I}_r, \mathcal{J}_r)_{r=1}^{R}$ be the stratification induced by ρ from Definition 4 on $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The r-th graph stratum $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$, $r > 1$, is obtained by removing all edges not incident on $\mathcal{I}_r$ and vertices that do not neighbor vertices in $\mathcal{I}_r$ and subsequently contracting $\mathcal{I}_{1:r-1}$ into ζ_r according to Definition 3. Further, let $\boldsymbol{\Phi}_r$ denote the simple random walk on $\mathcal{G}_r$.

It can be shown that the vertex set $\mathcal{V}_r$ contains the r-th stratum $\mathcal{I}_r$, the r-th supernode ζ_r, obtained by collapsing $\mathcal{I}_{1:r-1}$, and vertices from subsequent strata neighboring $\mathcal{I}_r$, $\cup_{u \in \mathcal{I}_r} \mathbf{N}(u) \cap \mathcal{I}_{r+1:R}$. The edge multiset $\mathcal{E}_r$ is the union of $\mathcal{J}_r$ and edges that connect ζ_r to vertices in $\mathcal{I}_r$ resulting from the graph contraction. A detailed example is shown in Fig. 1. Note that when $R = 2$, Ripple reduces to the estimator from Avrachenkov et al. (2016).

Ergodicity-Preserving Stratification. Because the *RWT* Estimate is consistent only if the underlying graph is connected according to Lemma 1, we have the following definition:

Definition 6 (Ergodicity-Preserving Stratification (*EPS*)) The stratification due to ρ from Definition 4 is an Ergodicity-Preserving Stratification if each graph stratum from Definition 5 is connected, i.e., $\boldsymbol{\Phi}_r$, $r > 1$, is irreducible.

We propose necessary and sufficient conditions on ρ that yield an *EPS*.

Proposition 1 *ρ yields an* EPS *if the following three conditions are satisfied:*

(a) for at least one vertex in each connected component of $\mathcal{G}$, ρ evaluates to 1*;*

(b) for each u: $\rho(u) = r$, *there exists* $v \in \mathbf{N}(u)$ *such that* $\rho(v) \leq r$; *and*
(c) there exists $(u_0, v_0) \in \mathcal{E}$ *such that* $\rho(u_0) = r$ *and* $\rho(v_0) < r$.

Although the optimal stratification would depend on $\mathcal{G}$ and the quantity being estimated, an ideal stratification would yield graph strata wherein the supernode degree and connectivity are maximized (Lemma 1) while minimizing the number of strata (because of the bias propagation described in Theorem 2). ρ needs to be efficient as well because we will see that it is evaluated at each step of the random walk and the Ripple estimators from Definitions 8 and 9 heavily depend on it. In Proposition 10 we show that return times to the supernode are inversely proportional to the fraction of vertices in $\mathcal{I}_r$ connected to $\mathcal{I}_{1:r-1}$.

3.2 Recursive regenerations

Assume for the moment that in each stratum, $r = 2, \ldots, R$, we know the degree of the supernode $\mathbf{d}(\zeta_r)$ and can sample directly from $p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$, which is the transition probability out of ζ_r in the graph stratum $\mathcal{G}_r$. We could then sample *RWT*s $\mathcal{T}_r$ and compute stratumwise *RWT* Estimates, which when combined as $\mu(\mathcal{J}_1) + \sum_{r=2}^{R} \hat{\mu}_r(\mathcal{T}_r)$ provide an unbiased estimate of $\mu(\mathcal{E})$ as a direct consequence of Lemma 1 and the linearity of expectations. Unfortunately, the impracticality of this assumption, especially under Assumption 1 (when $R > 2$), necessitates the following relaxation.

Definition 7 *(Supernode Estimates,* $\widehat{\mathbf{d}}(\zeta_r)$ *and* $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$*)* Given an *EPS* of $\mathcal{G}$ (Definition 6), the supernode estimates in the r-th graph stratum $\mathcal{G}_r$ consist of the estimate of the degree $\widehat{\mathbf{d}}(\zeta_r)$ and a sample from some approximate transition probability out of the supernode $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$. Let $\widehat{\boldsymbol{\Phi}}_r$ be the random walk on $\mathcal{G}_r$, where transitions are sampled according to $\boldsymbol{\Phi}_r$ everywhere except ζ_r, where they are sampled from $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$.

Although $\widehat{\boldsymbol{\Phi}}_r$ may not be reversible, *RWT*s on $\widehat{\boldsymbol{\Phi}}_r$ retain pairwise independence and the benefits stated after Lemma 1. We leverage this fact in the following recursive solution that computes *supernode estimates* in the current stratum using supernode estimates and tours sampled in the previous strata.

Definition 8 (Ripple'*s Recurrence Relation*) Given a graph $\mathcal{G}$ stratified according to ρ (Definition 6) and some stratum r, $1 < r \leq R$, assume access to the result of previous recursive steps, i.e., the set of m_q *RWT*s $(\mathcal{T}_q^{\dagger})$, supernode degree estimates $\widehat{\mathbf{d}}(\zeta_q)$ and estimated transition probabilities out of the supernode $\widehat{p}_{\boldsymbol{\Phi}_q}(\zeta_q, \cdot)$ (Definition 7) for all $2 \leq q < r$. The estimate of the number of edges between $\mathcal{I}_q$ and $\mathcal{I}_r$ is given by

$$\widehat{\beta}_{q,r} = \frac{\widehat{\mathbf{d}}(\zeta_q)}{|\mathcal{T}_q^{\dagger}|} \sum_{\mathbf{X} \in \mathcal{T}_q^{\dagger}} \sum_{j=2}^{|\mathbf{X}|} \mathbf{1}\{\rho(X_j) = r\}, \tag{4}$$

where X_j is the j-th state visited in tour $\mathbf{X}$, and by convention, $\widehat{\beta}_{1,r} = |\mathcal{E} \cap \mathcal{I}_1 \times \mathcal{I}_r|$ is exactly computed. The r-th supernode degree is then estimated as

$$\widehat{\mathbf{d}}(\zeta_r) = \sum_{q=1}^{r-1} \widehat{\beta}_{q,r} \,. \tag{5}$$

Transitions from $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$ are sampled by sampling $q \in \{1, \ldots, r-1\}$ with probability $\widehat{\beta}_{q,r}$ and then sampling u.a.r. from $\widehat{\mathbf{U}}_{q,r}$, which is defined as

$$\widehat{\mathbf{U}}_{q,r} = \uplus_{\mathbf{X} \in \mathcal{T}_q^\dagger} \uplus_{j=2}^{|\mathbf{X}|} \left\{ X_j : \rho(X_j) = r \right\} \,, \text{ when } q > 1 \,, \tag{6}$$

and as $\uplus_{u \in \mathcal{I}_1} \mathbf{N}(u) \cap \mathcal{I}_r$ by convention when $q = 1$, where $\uplus$ is the multi-set union. $\widehat{\mathbf{U}}_{q,r}$, $q > 1$, is thus the multi-set of all states in $\mathcal{I}_r$ visited by *RWT*s on $\widehat{\boldsymbol{\Phi}}_q$. An *RWT* so started stops when it reaches some state X', where $\rho(X') = r$.

Proposition 7 ("Appendix C") contains additional details for sampling *RWT*s on $\boldsymbol{\Phi}_r$. The above recursion therefore allows us to estimate supernode degrees and sample *RWT*s to compute an estimate of $\mu(\mathcal{E})$ from Eq. (1) as follows:

Definition 9 (Ripple*'s* μ *Estimator*) Given the supernode degree estimates $\widehat{\mathbf{d}}(\zeta_r)$ and *RWT*s $\mathcal{T}_r^\dagger$ sampled in each graph stratum from Definition 8 and the edge strata $\mathcal{J}_r$, $2 \leq r \leq R$ based on an *EPS* of $\mathcal{G}$ from Definition 6, the Ripple estimate is defined as

$$\hat{\mu}_{Ripple} = \mu(\mathcal{J}_1) + \sum_{r=2}^{R} \hat{\mu}\left(\mathcal{T}_{2:r}^\dagger; f\right) \,, \tag{7}$$

$$\text{where, } \hat{\mu}\left(\mathcal{T}_{2:r}^\dagger; f\right) = \frac{\widehat{\mathbf{d}}(\zeta_r)}{2|\mathcal{T}_r^\dagger|} \sum_{\mathbf{X} \in \mathcal{T}_r^\dagger} \sum_{j=2}^{|\mathbf{X}|-1} f(X_j, X_{j+1}) \,, \tag{8}$$

and X_j is the jth state visited by the *RWT* $\mathbf{X} \in \mathcal{T}_r^\dagger$. The dependence of $\mathcal{T}_r^\dagger$ and $\widehat{\mathbf{d}}(\zeta_r)$ on $\mathcal{T}_{2:r-1}^\dagger$ is suppressed for brevity.

This estimate of $\mu(\mathcal{E})$ is unbiased when the number of tours is infinite.

Theorem 1 *The Ripple estimate from Definition 9 is a consistent estimator of* $\mu(\mathcal{E})$ *(asymptotically unbiased in the number of tours), that is,*

$$\lim_{|\mathcal{T}_2^\dagger| \to \infty} \ldots \lim_{|\mathcal{T}_R^\dagger| \to \infty} \mu(\mathcal{J}_1) + \sum_{r=2}^{R} \hat{\mu}\left(\mathcal{T}_{2:r}^\dagger; f\right) \overset{a.s}{=} \mu(\mathcal{E}) \,.$$

In the finite regime, however, there exists a bias in each stratum that depends on the estimation bias in the previous strata, which we quantify as follows:

Theorem 2 *Given the random walk* $\boldsymbol{\Phi}_r$ *on the* EPS*-stratum* $\mathcal{G}_r$ *from Definitions 5 and 6, the estimates of the degree and transition probability at the supernode* $\widehat{\mathbf{d}}(\zeta_r)$ *and* $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$ *from Definition 7, and assuming aperiodic* $\boldsymbol{\Phi}_r$*, the bias of the Ripple*

estimate in the rth stratum from Eq. (8) *is given by*

$$\left|\mathbb{E}\left[\hat{\mu}\left(\mathcal{T}^{\dagger}_{2:r};\, f\right)\left|\mathcal{T}^{\dagger}_{2:r-1}\right.\right] - \mu(\mathcal{J}_r)\right| \leq (\lambda_r \nu_r + |1-\lambda_r|)\,\frac{\sqrt{3}B|\mathcal{E}_r|}{\sqrt{\delta_r}}\,,$$

where δ_r *is the spectral gap of* $\boldsymbol{\Phi}_r$, B *is the upper bound of* f, $\nu_r = \|\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot) - p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)\|_2$ *is the* L^2 *distance between transition probabilities out of* ζ_r (Aldous and Fill 2002) (Definition 13) *and* $\lambda_r = \widehat{\mathbf{d}}(\zeta_r)/\mathbf{d}(\zeta_r)$.

Therefore, the bias in each stratum affects the bias in subsequent strata. Consequently, we control the empirical variance in each stratum by increasing the number of tours sampled (we detail this for subgraph counting in Sect. 4).

4 Applying Ripple to count subgraphs

We now focus on a concrete implementation of Ripple to count subgraphs on a given simple input graph $G = (V, E, L)$ with vertices V, edges E, and attribute function L, which is assumed to be finite and undirected. In general, a subgraph induced by any $V' \subset V$ on G is given by $G\left(V'\right) = (V', E \cap (V' \times V'), L)$. However, in this work, we are interested in subgraphs $G\left(V'\right)$ that are connected and where $|V'| = k$, referred to as a connected, induced subgraph ($-CIS$) of size k or $k-CIS$. As such, the task is defined as

Definition 10 *(Subgraph Count)* Let $\mathcal{V}^k$ be the set of all $k-CIS$s of graph G, let $\sim$ denote the graph isomorphism equivalence relation (or any equivalence relation), and let $\mathcal{H}$ be an arbitrary set of pairwise nonequivalent $k-CIS$s. The subgraph count is defined as the $|\mathcal{H}|$-dimensional vector $\mathcal{C}^k = (\mathcal{C}^k{}_H)_{H\in\mathcal{H}}$, where $\mathcal{C}^k{}_H = \sum_{s\in\mathcal{V}^k} \mathbf{1}\{s \sim H\}$, and $\mathbf{1}\{\cdot\}$ is the indicator function.

Therefore, $\mathcal{C}^k$ contains the count of subgraphs in $\mathcal{V}^k$ equivalent to each subgraph in $\mathcal{H}$. We suppress the dependence of $\mathcal{C}^k$ on $\mathcal{H}$ for simplicity.

Subgraph counting is challenging when $k > 3$ in real-world input graphs because $\mathcal{V}^k$ is not tractably enumerable and naively sampling k vertices to obtain $-CIS$s is challenging because ${}^{|\mathcal{V}^k|}/_{|V|^k} \to 0$ (as evidenced by Table 1). Next, we address this issue by reducing the subgraph counting problem to an edge sum (Eq. (1)) over a higher-order graph that only provides neighborhood query access for large-real-world input graphs. We also propose a stratification strategy compatible with the access model and introduce novel solutions to improve speed and memory requirements. We defer the straightforward aspects to "Appendix E", wherein we summarize the entire algorithm (Algorithm 2).

4.1 *MCMC* on the subgraph space

Wang et al. (2014) proposed a network over subgraphs called the $-HON$, which exposes neighborhood query access from Assumption 1 and is therefore amenable to *MCMC* solutions (which we optimize in Algorithm 1).

Definition 11 (it Higher-Order Network $k-HON$ Wang et al. 2014) The higher-order network or $-HON\ \mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$ is a graph whose vertices are the set of all $k-CIS$ contained in the input graph G, and (u, v) form an edge in $\mathcal{E}^k$ if they share all but $k-1$ vertices, that is, $|V(u) \cap V(v)| = k-1$.

In the $k-1-HON$, the subgraph induced by an edge $(u, v) \in \mathcal{E}^{k-1}$, i.e., $G(V(u) \cup V(v))$, is a $k-CIS$. Thus, the subgraph counts from Definition 10 can then be expressed as an edge sum over $\mathcal{G} \equiv \mathcal{G}^{k-1}$ as

$$\mathcal{C}^k = \mu(\mathcal{E}^{k-1}) = \sum_{(u,v)\in\mathcal{E}^{k-1}} \left(\frac{\mathbf{1}\{G(V(u) \cup V(v)) \sim H\}}{\gamma(u, v)} \right)_{H\in\mathcal{H}}, \tag{9}$$

where $\gamma(u, v) = |\{(\ddot{u}, \ddot{v}) \in \mathcal{E} : V(u) \cup V(v) \equiv V(\ddot{u}) \cup V(\ddot{v})\}|$ is the number of edges that represent the same subgraph as (u, v). The set of edges sampled by a random walk on $\mathcal{G}^{k-1}$ is called the *pairwise subgraph random walk* (PSRW). Having reduced the subgraph counting task to Eq. (1), we proceed with implementing Ripple.

4.2 Ergodicity-Preserving Stratification for subgraph counting

Toward using Ripple, we propose an Ergodicity-Preserving Stratification of $\mathcal{G}$ via the stratification function ρ.

Proposition 2 (EPSfor subgraphs) *Consider the set of* n_1 seed *subgraphs* $\mathcal{I}_1$ *whose vertex sets in* G *are pairwise non-intersecting. Let* $V(\mathcal{I}_1) \triangleq \cup_{\ddot{s}\in\mathcal{I}_1} V(\ddot{s})$ *be the set of all vertices in* G *forming subgraphs in* $\mathcal{I}_1$. *Let* $\textsc{dist}(u)$ *be the shortest path distance from* $u \in V$ *to any vertex in* $V(\mathcal{I}_1)$. *Define* ρ *as*

$$\rho(s) = 1 + \sum_{u\in V(s)} \left(\textsc{dist}(u) + \mathbf{1}\{u \in V(\mathcal{I}_1) \backslash V^*) \} \right),$$

where V^* *is the largest connected subset of* $V(s)$ *such that* $V^* \subseteq V(\ddot{s})$ *for some seed vertex* $\ddot{s} \in \mathcal{I}_1$ *with ties broken arbitrarily. If* $\mathcal{I}_1$ *contains a subgraph from each connected component of* G, *the stratification from Definition 4 generated using* ρ *is an Ergodicity-Preserving Stratification (Definition 6).*

$\textsc{dist}$ can be precomputed for all $u \in V$ using a single BFS in $O(|V| + |E|)$, and ρ can be computed in $O(k)$. Although R is unknown a priori, it is upper bounded as $(k-1) \cdot D_G$, where D_G is the diameter of G and the Ripple estimator simply ignores *empty* strata, i.e., strata in which the estimated degree of the supernode $\mathbf{d}(\zeta_r) = 0$. To control bias, we aim to reduce $\max_{u\in V} \textsc{dist}(u)$ by recruiting seed subgraphs in $\mathcal{I}_1$, which are far apart in G.

4.3 Miscellaneous optimizations

Controlling Memory through Streaming. In each pair of strata $r < t$, Definition 8 uses tours $\mathcal{T}_r^\dagger$ to compute $\hat{\mu}(\mathcal{T}_{2:r}^\dagger; f)$, $\widehat{\beta}_{r,t}$ and $\widehat{\mathbf{U}}_{r,t}$, which are, respectively, the estimates

of $\mu(\mathcal{J}_r)$ and the size of and sample from the set of vertices in $\mathcal{I}_t$ connected to $\mathcal{I}_r$. Although $\hat{\mu}(\mathcal{T}^{\dagger}_{2:r}; f)$ and $\widehat{\beta}_{r,t}$ can be computed as running sums, storing $\widehat{\mathbf{U}}_{r,t}$ requires memory on the order of the sum of all tour lengths, which is random. Our solution is to use *Algorithm R* (Vitter 1985), to sample a fixed-size (M) sample without replacement from all the tours in $\mathcal{T}^{\dagger}_r$ (See "Appendix E.1"). We note that although the hyperparameter M controls memory, it may introduce bias when the number of tours $|\mathcal{T}^{\dagger}_r| >$ M due to (possible) oversampling (Teixeira et al. 2021, Appendix F).

Speeding up Subgraph Random Walks. To sample a random walk in the $-HON$, naively sampling u.a.r. from the neighborhood of a $k-1-CIS$ requires $\mathrm{O}(k^4\Delta_G)$ operations, where Δ_G is the maximum degree in the input graph (see Appendix E.2). In Algorithm 1, we propose a rejection sampling algorithm that does so efficiently using *articulation points* (Hopcroft and Tarjan 1973).

Algorithm 1: Efficient Neighborhood Sampling in $\mathcal{G}^{(k-1)}$

Input: $k-1-CIS$ s, Graph G
Output: $x \sim \mathrm{UNIF}(\mathbf{N}_{\mathcal{G}^{k-1}}(s))$

Let $\deg_s = \sum_{u \in V(s)} \mathbf{d}(u)$ and $\mathcal{A}_s$ be the *articulating points* of s
while *True* **do**
  Sample u from $V(s)$ w.p. $\propto \deg_s - \mathbf{d}(u)$; // u is the vertex to remove
  Sample a from $V(s)\backslash\{u\}$ w.p. $\propto \mathbf{d}(a)$
  Sample $v \sim \mathrm{UNIF}(\mathbf{N}(a))$; // v is the vertex to add
  $\mathrm{BIAS} = |N(v) \cap V(s)\backslash\{u\}|$; // v's sampling bias
  if $\mathrm{UNIF}(0, 1) \leq 1/\mathrm{BIAS}$ **then**
    $x = G\left(V(s) \cup \{v\}\backslash\{u\}\right)$
    if $u \neq v$ **and** *($u \notin \mathcal{A}_s$* **or** *x is connected)* **then**
      return x ; // Connectivity Check

Proposition 3 *Given a subgraph $s \in \mathcal{V}^{k-1}$, Algorithm 1 samples u.a.r. from $\mathbf{N}_{\mathcal{G}^{k-1}}(s)$ in $\mathrm{O}(k^2 \frac{\Delta_s + k|\mathcal{A}_s|}{k-|\mathcal{A}_s|})$ expected time, where $\Delta_s \triangleq \max_{u \in V(s)} \mathbf{d}_G(u)$ is the maximum degree of vertices in s, and $\mathcal{A}_s$ contains articulation points of s.*

Therefore, the running time of Algorithm 1 is $\in \mathrm{O}(k\Delta_s + k^2)$ when s is dense ($|\mathcal{A}_s| \approx 0$) and increases to $\mathrm{O}(k^2\Delta_s + k^4)$ for sparse subgraphs, which is faster than the naive algorithm.

From Error Bounds to Tour Counts. Ripple auto-decides the number of *RWT*s required in each stratum based on an approximate error bound ϵ provided as input such that the number of tours $\to \infty$ as $\epsilon \to 0$, and the Ripple estimate converges to the ground truth (Theorem 1). Specifically, *RWT*s are sampled until we satisfy

$$\hat{\sigma}\left(\mathcal{T}^{\dagger}_r; f_1\right) / \sqrt{|\mathcal{T}^{\dagger}_r|} \leq \epsilon\, \hat{\mu}\left(\mathcal{T}^{\dagger}_r; f_1\right), \tag{10}$$

where $\hat{\mu}(\mathcal{T}^{\dagger}_r; f_1)$ is the Ripple estimate from Eq. (8) of the number of edges in the r-th graph stratum $\mathcal{G}_r$ (i.e., $f_1(\cdot) = 1$), and $\hat{\sigma}^2(\mathcal{T}^{\dagger}_r; f_1) = \widehat{\mathrm{Var}}_{\mathbf{X} \sim \mathcal{T}^{\dagger}_r}(\hat{\mu}(\mathbf{X}; f_1))$ is the former's empirical variance over tours.

Performance Guarantees. Ignoring the complexity of loading the input graph into memory, we show that for subgraph counting, the memory and time requirements of Ripple are a polynomial in k. In "Appendix E", we state and prove a detailed version in which the complexity also depends polynomially on the diameter and maximum degree of G and is invariant to $|V|$ and $|E|$.

Proposition 4 *Assuming a constant m* RWT*s sampled per stratum and ignoring graph loading, the Ripple estimator for k*−CIS *counts detailed in Appendix "E"–Algorithm 2 has total memory and time complexity in* $\widehat{\mathrm{O}}(k^3 + |\mathcal{H}|)$ *and* $\widehat{\mathrm{O}}(k^7 + |\mathcal{H}|)$*, respectively, when all factors other than k and* $|\mathcal{H}|$ *are ignored.*

More details for subgraph counting with Ripple are provided in Appendix E.

5 Experiments and results

We now evaluate the Ripple estimator for k-node subgraph ($k-CIS$) counts on large-real-world networks. We show that Ripple outperforms the state-of-the-art method in terms of time and space and that Ripple converges to the ground truth for various pattern sizes as hyperparameters are varied. Additional experiments can be found in (Teixeira et al. 2021, Appendix F). Our code is available at https://github.com/dccspeed/ripple.

- *Execution environment.* Our experiments were performed on a dual Intel Xeon Gold 6254 CPU with 72 virtual cores (total) at 3.10 GHz and 392 GB of RAM. In addition, this machine is equipped with a fast SSD NVMe PCIex4 with 800 GB of free space available.
- *Baselines.* We use Motivo (Bressan et al. 2019), a fast and parallel C++ system for subgraph counting, as the baseline because it is the only method capable of counting large patterns ($k > 6$), to the best of our knowledge. Additionally, notice that existing *MCMC* methods for subgraph counting, such as IMPRG (Chen and Lui 2018) and RGPM (Teixeira et al. 2018), cannot count beyond $k = 5$ in practice.
- *Datasets.* We use large networks from SNAP (Leskovec and Krevl 2014), representing diverse domains, which have been used to evaluate many subgraph counting algorithms (Bressan et al. 2018, 2019). Table 1 presents the basic features of these datasets, including the order of magnitude of the Ripple estimates of the subgraph counts $|\mathcal{V}^k|$, $k = 6, 8, 10, 12$.
- *Hyper-parameters* $\mathcal{I}_1$, M *and* ϵ. Finally, we evaluate the trade-off between accuracy and resource consumption by varying $\mathcal{I}_1$, M and ϵ, detailed in Sects. 4.2 and 4.3 and (Teixeira et al. 2021, Appendix F) (M).

5.1 Scalability assessment

We start by assessing the scalability of the methods when estimating $k-CIS$ counts for $k \geq 6$. To the best of our knowledge, Motivo is the only existing method capable of estimating these patterns. Motivo has two phases: a build-up phase, which constructs an index table in the disk, and a sampling phase that queries this table. We only

Table 1 The graphs that we used along with their diameter D_G, maximum degree Δ_G and the estimated orders of magnitude of $k-CIS$ counts, $|\mathcal{V}^k|$

Graph	$\|V\|$	$\|E\|$	D_G	Δ_G	Magnitude of Est. # of $-CIS$s			
					$\|\mathcal{V}^6\|$	$\|\mathcal{V}^8\|$	$\|\mathcal{V}^{10}\|$	$\|\mathcal{V}^{12}\|$
Amazon	334,863	925,872	44	549	10^{11}	10^{15}	10^{19}	10^{22}
DBLP	317,080	1,049,866	21	343	10^{12}	10^{16}	10^{19}	10^{23}
Cit-Pat.	3,774,768	16,518,948	22	793	10^{14}	10^{18}	10^{22}	10^{26}
Pokec	1,632,803	30,622,564	11	14,854	10^{18}	10^{25}	10^{32}	10^{38}
LiveJ.	3,997,962	34,681,189	17	14,815	10^{19}	10^{25}	10^{32}	10^{38}
Orkut	3,072,441	117,185,083	9	33,313	10^{21}	10^{28}	10^{35}	10^{43}

measure the time taken by the build-up phase and the out-of-core (disk) usage because this is a bottleneck for Motivo. As such, we report the best-case scenario for Motivo, and the reported values are lower bounds for the actual time and space requirement. For Ripple, we report the total time and the RAM usage as the space cost because our method works purely in memory. Both methods were executed using all threads available. In Tables 2 and 3, we compare the running time and space usage of Ripple and Motivo. We also report their rate of increase in terms of the subgraph size k in columns $\text{Time}^{(k)}/\text{Time}^{(k-2)}$ and $\text{Space}^{(k)}/\text{Space}^{(k-2)}$. We fix $\epsilon = 0.003$, $|\mathcal{I}_1| = 10^4$ and $\text{M} = 10^7$ based on the analysis in Sect. 5.2. In (Teixeira et al. 2021, Appendix F), we also report $\frac{\max - \min}{\text{mean}}$ of the Ripple estimates to ensure that the results are not arbitrary.

For Motivo, we follow the authors' suggestions.

Running time Scalability (Table 2). Although Motivo outperforms Ripple for $k = 6, 8$, it does not scale well for $k = 10, 12$, where the execution terminates because of insufficient storage space. Particularly, for DBLP, Motivo required approximately 10 minutes to process $10-CIS$ but almost 9 h for $12-CIS$, a growth rate of 58×. On the other hand, Ripple not only succeeded in **all** configurations in less than 4 hours on average but also exhibited a smoother growth in running time, with the largest increase ratio being 2.7×, observed for DBLP and LiveJournal when k went from 8 to 10. Furthermore, $\text{Time}^{(k)}/\text{Time}^{(k-2)} < (\text{k}/(\text{k-2}))^7$ in all cases according to Proposition 10.

Space Scalability (Table 3). The trends in space usage mirror those of the running time, where we see an almost exponential increase w.r.t. k for Motivo compared to a near constant increase for Ripple despite its polynomial complexity (Proposition 10). For example, in Amazon, Motivo's space demand increases by 7.5× when k goes from 6 to 8 and increases to 12× from 10 to 12. Ripple's largest rate of increase is 1.4× when k goes from 6 to 8 for DBLP, and it saves up to 600 GB of space when Motivo does not crash.

Table 2 Running time comparison between Ripple and Motivo

Dataset	k	Motivo Build-up only		Ripple ($\epsilon = 0.003$)		Ripple gain (h)
		Time (h)	$\frac{Time^{(k)}}{\text{Time}^{(k-2)}}$	Time (h)	$\frac{\text{Time}^{(k)}}{\text{Time}^{(k-2)}}$	
Amazon	6	**0.002 ± 0.000**	–	0.020 ± 0.000	–	−0.018
	8	**0.006 ± 0.000**	3×	0.029 ± 0.000	1.4×	−0.023
	10	0.082 ± 0.000	13.7×	**0.056 ± 0.000**	1.9×	+0.026
	12	3.630 ± 0.002	44.3×	**0.095 ± 0.002**	1.7×	+3.535
DBLP	6	**0.002 ± 0.000**	–	0.013 ± 0.000	–	−0.011
	8	**0.007 ± 0.000**	3.5×	0.030 ± 0.000	2.3×	−0.023
	10	0.156 ± 0.000	22.3×	**0.082 ± 0.000**	2.7×	+0.074
	12	9.099 ± 0.002	58.3×	**0.105 ± 0.002**	1.3×	+8.994
Patents	6	**0.022 ± 0.000**	–	0.033 ± 0.000	–	−0.011
	8	0.098 ± 0.000	4.5×	**0.051 ± 0.000**	1.5×	+0.047
	10	> 1.1 h, crashed	–	**0.090 ± 0.001**	1.8×	–
	12	> 0.5 h, crashed	–	**0.117 ± 0.003**	1.3×	–
Pokec	6	**0.012 ± 0.000**	–	0.459 ± 0.142	–	−0.447
	8	**0.128 ± 0.000**	10.7×	0.759 ± 0.282	1.7×	−0.631
	10	5.965 ± 0.000	46.6×	**1.400 ± 0.592**	1.8×	+4.565
	12	> 1.5 h, crashed	–	**1.469 ± 0.334**	1×	–
LiveJ.	6	**0.024 ± 0.000**	–	0.351 ± 0.009	–	−0.327
	8	**0.205 ± 0.000**	8.5×	0.642 ± 0.074	1.8×	−0.437
	10	> 2.3 h, crashed	–	**1.76 ± 1.550**	2.7×	–
	12	> 0.7 h, crashed	–	**2.189 ± 1.350**	1.2×	–
Orkut	6	**0.032 ± 0.000**	–	0.669 ± 0.026	–	−0.637
	8	**0.585 ± 0.006**	18.3×	1.744 ± 0.983	2.6×	−1.159
	10	> 8.9 h, crashed	–	**2.633 ± 1.065**	1.5×	–
	12	> 1.8 h, crashed	–	**3.967 ± 3.162**	1.5×	–

The last column shows that for large k, Ripple provides gains of up to 9 h when Motivo can run to completion. Motivo crashes for large k on large graphs (Bold values indicate superior performance)

5.2 Accuracy and convergence assessment

Next, we evaluate the accuracy and convergence of Ripple on small and large subgraph patterns, where the former refers to subgraph sizes in which the number of isomorphic subgraphs can be exactly computed using *ESCAPE* (Pinar et al. 2017), i.e., $k \leq 5$.

Accuracy on Small k. For $k \in \{3, 5\}$, we evaluate the L2-norm between the Ripple estimate and the exact value of the count vector $\mathcal{C}^k$ (Eq. 9) of all non-isomorphic subgraph patterns. Figure 2 shows results for $k = 5$ (where the number of patterns of interest $|\mathcal{H}| = 21$) for different settings of the parameters ϵ and $|\mathcal{I}_1|$. In all datasets, we note that the L2-norm decreases as ϵ decreases from 0.3 to 0.003 and as $|\mathcal{I}_1|$ increases from 100 to 10^4. Between the worst setting, $(\epsilon, |\mathcal{I}_1|) = (0.3, 100)$, and the best $(\epsilon, |\mathcal{I}_1|) = (0.003, 10^4)$, we see an error reduction close to an order of magnitude.

Table 3 Space usage comparison between Ripple and Motivo

Dataset	k	Motivo Build-up only		Ripple ($\epsilon = 0.003$)		Ripple gain (GB)
		Space (GB)	$\frac{\text{Space}^{(k)}}{\text{Space}^{(k-2)}}$	Space (GB)	$\frac{\text{Space}^{(k)}}{\text{Space}^{(k-2)}}$	
Amazon	6	**0.53 ± 0.00**	–	4.69 ± 0.06	–	−4.16
	8	**4.00 ± 0.00**	7.5×	5.73 ± 0.12	1.2×	−1.73
	10	48.00 ± 0.00	12×	**7.38 ± 0.36**	1.3×	+40.62
	12	559 ± 0.00	11.6×	**9.09 ± 1.02**	1.2×	+549.91
DBLP	6	**0.50 ± 0.00**	–	4.58 ± 0.02	–	−4.08
	8	**4.00 ± 0.00**	8×	6.31 ± 0.00	1.4×	−2.31
	10	50.00 ± 0.00	12.5×	**7.99 ± 0.01**	1.3×	+42.01
	12	611.00 ± 0.00	12.2×	**10.45 ± 0.02**	1.3×	+600.55
Patents	6	**7.00 ± 0.00**	–	11.50 ± 0.05	–	-4.5
	8	66.00 ± 0.00	9.4×	**13.80 ± 0.03**	1.2×	+52.2
	10	> 800, crashed	–	**15.85 ± 0.08**	1.1×	> 800
	12	> 800, crashed	–	**18.12 ± 0.10**	1.1×	> 800
Pokec	6	**3.7 ± 0.00**	–	13.69 ± 0.06	–	−9.99
	8	36.00 ± 0.00	9.7×	**17.17 ± 0.03**	1.3×	18.83
	10	407.00 ± 0.00	11.3×	**20.31 ± 0.01**	1.2×	+386.69
	12	> 800, crashed	–	**22.82 ± 0.03**	1.1×	> 800
LiveJ.	6	**7.70 ± 0.00**	–	18.26 ± 0.02	–	−10.56
	8	73.00 ± 0.00	9.5×	**21.26 ± 0.00**	1.2×	+51.74
	10	> 800, crashed	–	**24.43 ± 0.72**	1.1×	> 800
	12	> 800, crashed	–	**27.75 ± 0.00**	1.1×	> 800
Orkut	6	**7.90 ± 0.00**	–	40.38 ± 0.00	–	−32.48
	8	78.00 ± 0.000	9.9×	**43.49 ± 0.00**	1.1×	+34.51
	10	> 800, crashed	–	**46.63 ± 0.00**	1.1×	> 800
	12	> 800, crashed	–	**49.73 ± 0.00**	1.1×	> 800

Motivo runs out of *disk* space for larger datasets in which $k \geq 10$, while Ripple scales almost linearly. Ripple saves up to 600 GB of space when Motivo can run (Bold values indicate superior performance)

This is due to Theorem 2 and Lemma 1 because reducing ϵ increases the number of tours, lowers the error and therefore leads to reduced error propagation. Increasing $\mathcal{I}_1$ also reduces the number of strata and therefore error propagation. Results for $k = 5$ using the L-∞ norm can be found in (Teixeira et al. 2021, Appendix F).

Convergence for Large k. When $k > 5$, subgraph counts for real-world graphs are computationally intractable. Therefore, we show that Ripple converges in these cases as we increase the computing effort. Consider the hypothesis that sparse patterns are frequent in power-law networks as k increases. To glean empirical evidence for this, we choose an appropriate pattern set $\mathcal{H}$ and equivalence relationship in Definition 10, and we use Ripple to compute the total number of $k-CIS$ s and the number of sparse subgraphs and stars. A subgraph is defined as sparse if its density lies between 0 and

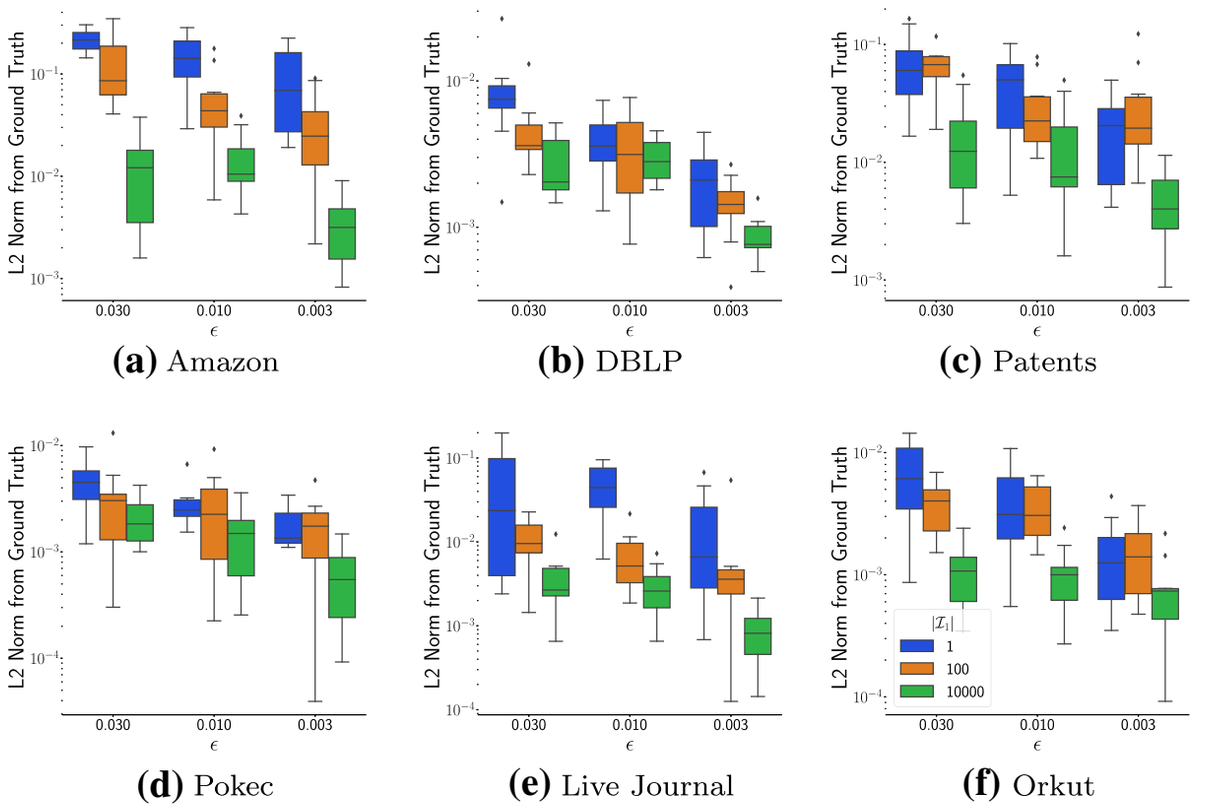


Fig. 2 Accuracy and convergence analysis for 5−*CIS* s. We plot the L2-norm between the Ripple estimate and the exact value of the count vector $\mathcal{C}^5$ (Eq. 9) of all non-isomorphic subgraph patterns against various configurations of the parameters ϵ and $|\mathcal{I}_1|$. As expected, the accuracy improves as the error bound ϵ decreases and the number of seed subgraphs $|\mathcal{I}_1|$ increases. Each box and whisker represents 10 runs

0.25, according to Liu and Wong (2008). In Fig. 3, we show that Ripple converges for all datasets, and as expected, most patterns are sparse, with close to half of the patterns in many of the studied networks being stars. This proportion is attenuated in DBLP and Patents, where dense substructures naturally emerge from collaboration/citation among the authors that these graphs represent.

6 Related work

For better presentation, we split this section into two parts: (1) parallel *MCMC* techniques and (2) methods for subgraph counting.

Parallel MCMCthrough Splitting. Since Nummelin (1978); Athreya and Ney (1978), multiple techniques have been proposed to circumvent the burn-in period by splitting the chain into i.i.d. sample paths. This approach allows practitioners to compute unbiased estimates in parallel and determine confidence intervals. Perfect sampling methods based on coupling (Propp and Wilson 1996) require the transitions to be monotonic w.r.t. some ordering over the state space, and annealing/tempering (Neal 2001) methods require some notion of temperature, which are absent in general graph random walks. Methods such as (Mykland et al. 1995; Jacob et al. 2020; Glynn and Rhee 2014) require a minorization condition to hold, albeit implicitly.

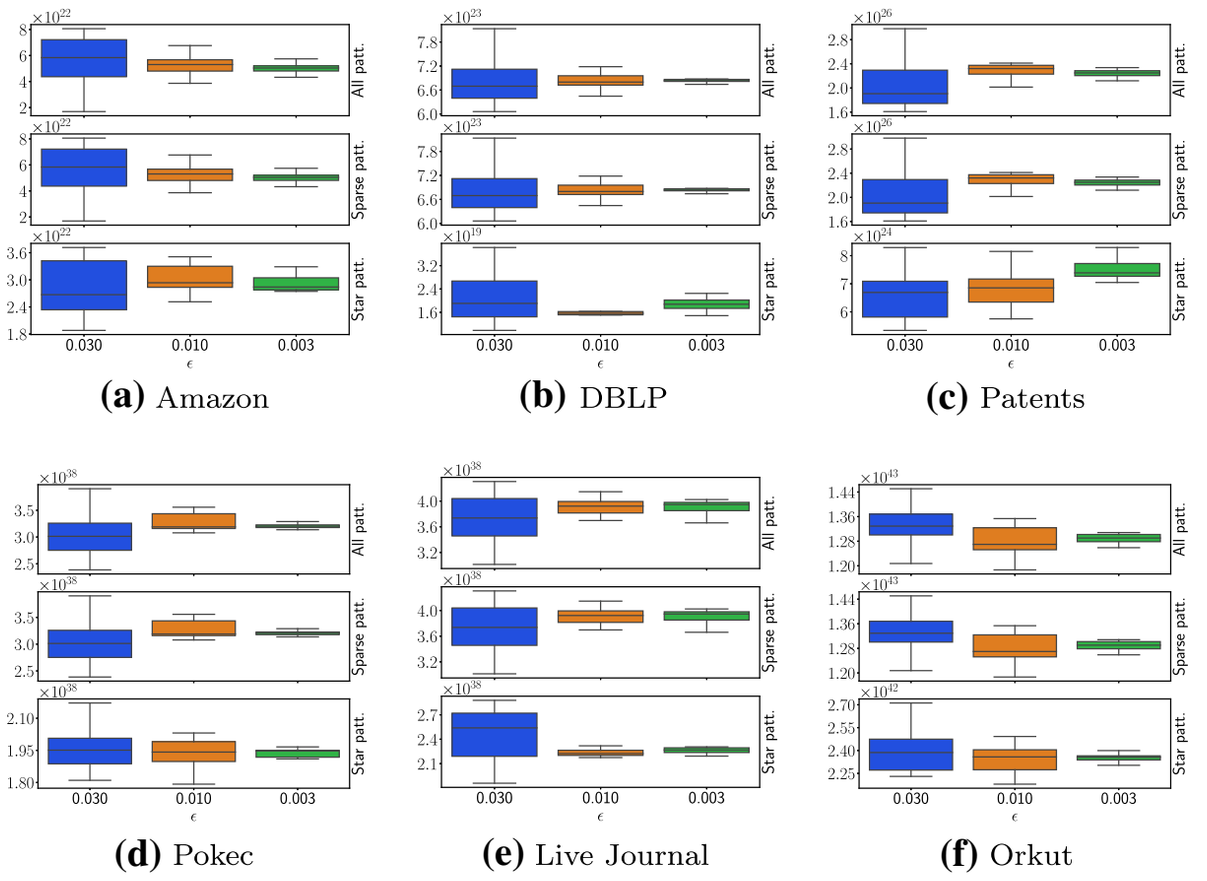


Fig. 3 Convergence of Ripple estimates of $12-$*CIS* pattern counts. We estimate the total number of subgraphs $|\mathcal{V}^{12}|$ and the number of sparse patterns and stars. Estimates over 10 runs are presented as box and whiskers plots, which exhibit a reduction in variance as ϵ increases. Indeed, almost all patterns are sparse, and the most frequent substructure is a star

Regeneration point-based methods on finite state chains (Cooper et al. 2016; Massoulié et al. 2006; Avrachenkov et al. 2016, 2018; Savarese et al. 2018; Teixeira et al. 2018) are more general because they only rely on standard ergodicity conditions. Although Cooper et al. (2016); Massoulié et al. (2006) used tours to estimate graph properties, Avrachenkov et al. (2016, 2018) proposed supernodes to reduce running times. The studies in Savarese et al. (2018); Teixeira et al. (2018) further used supernode-based tours to estimate gradients in *RBM*s and to count subgraphs. To the best of our knowledge, no existing regeneration point method controls running times through stratification.

*Subgraph Counting through Sampling*Many random walk algorithms have been proposed to sample subgraphs, with some methods only capable of estimating subgraph pattern *distributions*, which is much easier than estimating *counts*. The studies of GUISE (Bhuiyan et al. 2012) and RSS (Matsuno and Gionis 2020) use a Metropolis-Hastings (Hastings 1970) walk, and the latter improves the mixing time of the underlying Markov chain using canonical paths (Sinclair 1992). Waddling (Han and Sethu 2016) and IMPRG Chen and Lui (2018) perform a simple random walk over the input graph and use specialized estimators to sample 5-node patterns. Although PSRW (Wang et al. 2014) first proposed the $-$*HON*-based random walk

and RGPM (Teixeira et al. 2018) used tours on it to estimate subgraph counts, both are limited to $k \leq 5$ due to the size of the $-HON$.

Multiple attempts to Monte Carlo sample subgraphs have been proposed whose scaling is limited because of the complexity of computing either the importance weights, rejection rate or variance (Kashtan et al. 2004; Wernicke 2006; Iyer et al. 2018; Yang et al. 2018; Wang et al. 2018). Efficient methods that sample dense regions/subgraphs are unfortunately not extensible to sparse patterns Jain and Seshadhri (2017, 2020). Motivo (Bressan et al. 2018, 2019) is an example of color-coding methods in which an index table is built using a deterministic dynamic programming algorithm, which is then exploited to sample subgraphs uniformly and independently. However, *CC* methods suffer from the exponential time and space complexities associated with building and accessing the index table. Motivo proposed succinct index tables and efficient out-of-core I/O mechanisms to ameliorate this issue and extended the applicability of *CC* methods to larger subgraphs. Please, check Ribeiro et al. (2019) for an extensive survey on subgraph counting methods.

7 Conclusions

In this paper, we propose the Ripple estimator that uses *sequentially stratified regenerations* to control the running time of a random walk tour-based *MCMC*. We prove that the estimator is consistent (w.r.t. the number of random walk tours) and that the time and memory complexity of our implementation for the subgraph counting problem is linear in the number of patterns of interest and polynomial in the subgraph size. We empirically verify our claims on multiple graph datasets and show that Ripple can accurately estimate subgraph counts with a smaller memory footprint compared to that of the state-of-the-art Motivo (Bressan et al. 2019). Ripple is currently the only subgraph pattern count estimator that can estimate $k = 10, 12$ node patterns in million-node graphs. Beyond our specific application, Ripple provides a promising way to expand the sphere of influence of regenerative simulation in discrete reversible *MCMC*.

Acknowledgements This work was funded in part by Brazilian agencies CNPq, CAPES and FAPEMIG, by projects Atmosphere, INCTCyber, MASWeb, and CIIA-Saude, and by the National Science Foundation (NSF) awards CAREER IIS-1943364 and CCF-1918483. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

A Notation

The most important notations from the paper are summarized in Table 4.

Table 4 Table of notations

Symbol	Explanation
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	The graph where we have neighbor query access and whose edge sum is being computed
$N(u), \mathbf{d}(u)$	Neighborhood and degree of a vertex in $\mathcal{G}$ if no subscript is specified
$\mu(\mathcal{E})$	The sum over edges in $\mathcal{E}$ (or some subset) of some function f
$\boldsymbol{\Phi}$, $p_{\boldsymbol{\Phi}}(u, v)$, $\pi_{\boldsymbol{\Phi}}(u)$	The random walker on $\mathcal{G}$, its transition probability and stationary distribution
$\mathbf{X}, \xi, \mathcal{T}$	An *RWT* (tour), its length and a set of m *RWT*s
$\hat{\mu}_*(\mathcal{T}; f, \mathcal{G})$	An *RWT* Estimate of $\mu(\mathcal{E})$
δ	The spectral gap of the transition probability matrix of a chain
$\hat{\sigma}(\cdot)^2$	Empirical variance of an *RWT* Estimate
$\zeta_I, \mathcal{G}_I$	Collapsed state and graph obtained by collapsing $I \subset \mathcal{V}$
$q < r < t$	Strata ids always used in the same order $1 \leq q < r < t \leq R$
$\rho : \mathcal{V} \to \{1, \ldots, R\}$	Stratification function
$\mathcal{I}_r, \mathcal{J}_r$	r-th vertex and edge stratum
$\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$	r-th graph stratum
$\zeta_r, \mathcal{T}_r$	Supernode in each stratum and a set of m_r perfectly sampled tours from ζ_r
$\mathbf{d}(\zeta_r)$, $p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$	Degree and transition probability out of the supernode
$\widehat{\mathbf{d}}(\zeta_r)$, $\widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$	Estimated degree and transition probability out of the supernode
$\mathcal{T}_q^{\dagger}$	m_q *RWT*s samples using supernode estimates
$\widehat{\beta}_{q,r}$	The estimate of the number of edges between $\mathcal{I}_q$ and $\mathcal{I}_r$

Table 4 continued

Symbol	Explanation
$\widehat{\mathbf{U}}_{q,r}$	Multiset of states visited by $\mathcal{T}_q^{\dagger}$ that lie in $\mathcal{I}_r$
$\hat{\mu}_{Ripple}, \hat{\mu}\left(\mathcal{T}_{2:r}^{\dagger}; f\right)$	Overall and per-stratum Ripple estimate
$\delta_r, \nu_r, \lambda_r$	Spectral gap and the errors in the supernode estimates in the r-th stratum
$G = (V, E, L)$	The labelled input graph in which we want to count subgraphs
$G\left(V'\right)$	Subgraph induced by V' in G
$\mathcal{H}$	Nonequivalent (non-isomorphic) patterns of interest
$\mathcal{C}^k = (\mathcal{C}^k{}_H)_{H \in \mathcal{H}}$	The subgraph pattern count vector
$\mathcal{G}^k = (\mathcal{V}^k, \mathcal{E}^k)$	The $k-HON$ that provides neighborhood query access and is used to count subgraphs
$\gamma(u, v)$	Number of edges in $\mathcal{E}^{k-1}$ that represent the same subgraph as (u, v)
DIST(u)	Shortest path distance from $u \in V$ to any seed vertex in $V(\mathcal{I}_1)$
V^*	The largest connected subset of $V(s)$ that constitutes an intersection between s and $\ddot{s} \in \mathcal{I}_1$
M	Reservoir size
Δ_G, D_G	Maximum degree in and diameter of G
$\mathcal{A}_s$	Articulation points in s
ϵ	Per-stratum error bound used to control tour count

B Proofs for Section 2

B.1 *MCMC* Estimates

Given a graph $\mathcal{G}$, when the $|\mathcal{E}|$ is unknown, the *MCMC* estimate of $\mu(\mathcal{E})/|\mathcal{E}|$ is given by:

Proposition 5 *(*MCMC *Estimate* (Geyer 1992; Geman and Geman 1984; Hastings 1970)) *When $\mathcal{G}$ from Definition 1 is connected, the random walk $\boldsymbol{\Phi}$ is reversible and positive recurrent with stationary distribution $\pi_{\boldsymbol{\Phi}}(u) = \mathbf{d}(u)/2|\mathcal{E}|$. Then, the* MCMC *estimate $\hat{\mu}_0\left((X_i)_{i=1}^t\right) = \frac{1}{t-1}\sum_{i=1}^{t-1} f(X_i, X_{i+1})$, computed using an arbitrarily started sample path $(X_i)_{i=1}^t$ from $\boldsymbol{\Phi}$ is an asymptotically unbiased estimate of $\mu(\mathcal{E})/|\mathcal{E}|$. When $\mathcal{G}$ is non-bipartite, i.e., $\boldsymbol{\Phi}$ is aperiodic, and t is large, $\hat{\mu}_0$ converges to $\mu(\mathcal{E})/|\mathcal{E}|$ as $\left|\mathbb{E}[\hat{\mu}_0\left((X_i)_{i=1}^t\right)] - \mu(\mathcal{E})/|\mathcal{E}|\right| \leq B\,\frac{C}{t\delta(\boldsymbol{\Phi})}$, where $\delta(\boldsymbol{\Phi})$ is the spectral gap of $\boldsymbol{\Phi}$ and $C \triangleq \sqrt{\frac{1-\pi_{\boldsymbol{\Phi}}(X_1)}{\pi_{\boldsymbol{\Phi}}(X_1)}}$ such that $f(\cdot) \leq B$.*

Proof (*Asymptotic unbiasedness*) Because $\mathcal{G}$ is undirected, finite and connected, $\boldsymbol{\Phi}$ is a finite state space, irreducible, time-homogeneous Markov chain and is therefore positive recurrent (Bremaud 2001, 3-Thm.3.3). The reversibility and stationary distribution holds from the detailed balance test (Bremaud 2001, 2-Cor.6.1): $\pi_{\boldsymbol{\Phi}}(u)\,p_{\boldsymbol{\Phi}}(u,v) = \pi_{\boldsymbol{\Phi}}(v)\,p_{\boldsymbol{\Phi}}(v,u) = \frac{\mathbf{1}\{(u,v)\in\mathcal{E}\}}{2|\mathcal{E}|}$. The ergodic theorem (Bremaud 2001, 3-Cor.4.1) then applies because f is bounded and $\lim_{t\to\infty}\frac{1}{t-1}\sum_{i=1}^{t-1} f(X_i, X_{i+1}) = \sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} \pi_{\boldsymbol{\Phi}}(u)\,p_{\boldsymbol{\Phi}}(u,v)\,f(u,v) = \frac{\mu(\mathcal{E})}{|\mathcal{E}|}$. □

Proof (*Bias*) Let the i-step transition probability of $\boldsymbol{\Phi}$ be given by $p^i_{\boldsymbol{\Phi}}(u,v)$. The bias at the i-th step is given by

$$
\begin{aligned}
\text{BIAS}_i &= \left|\mathbb{E}\left[f(X_i, X_{i+1})\right] - \sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} \pi_{\boldsymbol{\Phi}}(u)\,p_{\boldsymbol{\Phi}}(u,v)\,f(u,v)\right| \\
&= \left|\sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} p^i_{\boldsymbol{\Phi}}(X_1,u)\,p_{\boldsymbol{\Phi}}(u,v)\,f(u,v) - \sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} \pi_{\boldsymbol{\Phi}}(u)\,p_{\boldsymbol{\Phi}}(u,v)\,f(u,v)\right| \\
&\leq B\left|\sum_{u\in\mathcal{V}} p^i_{\boldsymbol{\Phi}}(X_1,u)\sum_{v\in\mathcal{V}} p_{\boldsymbol{\Phi}}(u,v) - \sum_{u\in\mathcal{V}}\pi_{\boldsymbol{\Phi}}(u)\sum_{v\in\mathcal{V}} p_{\boldsymbol{\Phi}}(u,v)\right| \\
&\leq B\left|\sum_{u\in\mathcal{V}} p^i_{\boldsymbol{\Phi}}(X_1,u) - \sum_{u\in\mathcal{V}}\pi_{\boldsymbol{\Phi}}(u)\right| \leq B\sum_{u\in\mathcal{V}}\left|p^i_{\boldsymbol{\Phi}}(X_1,u) - \pi_{\boldsymbol{\Phi}}(u)\right|,
\end{aligned}
$$

where $f(\cdot) \leq B$, and the final inequality is due to Jensen's inequality. From (Diaconis and Stroock 1991, Prop-3), $\text{BIAS}_i \leq B\sqrt{\frac{1-\pi_{\boldsymbol{\Phi}}(X_1)}{\pi_{\boldsymbol{\Phi}}(X_1)}}\beta^i_*$, where $\beta_* = 1-\delta(\boldsymbol{\Phi})$ is the SLEM of $\boldsymbol{\Phi}$. Because of Jensen's inequality and by summing a GP, $\left|\mathbb{E}[\hat{\mu}_0\left((X_i)_{i=1}^t\right)] - \frac{\mu(\mathcal{E})}{|\mathcal{E}|}\right| \leq \frac{1}{t-1}\sum_{i=1}^{t-1}\text{BIAS}_i \leq \frac{B}{t-1}\sqrt{\frac{1-\pi_{\boldsymbol{\Phi}}(X_1)}{\pi_{\boldsymbol{\Phi}}(X_1)}}\frac{1-\beta^t_*}{1-\beta_*}$.

Assuming that $\beta^t_* \approx 0$ and $t-1 \approx t$ when t is sufficiently large completes the proof. □

Lemma 2 (Avrachenkov et al. 2016) *Let $\boldsymbol{\Phi}$ be a finite state space, irreducible, time-homogeneous Markov chain, and let ξ denote the return time of* RWT *started from some $x_0 \in \mathcal{S}$ as defined in Definition 2. If $\boldsymbol{\Phi}$ is reversible, then $\mathbb{E}\left[\xi^2\right] \leq \frac{3}{\pi_{\boldsymbol{\Phi}}(x_0)^2 \delta(\boldsymbol{\Phi})}$, where $\pi_{\boldsymbol{\Phi}}(x_0)$ is the stationary distribution of x_0, and $\delta(\boldsymbol{\Phi})$ is the spectral gap of $\boldsymbol{\Phi}$. When $\boldsymbol{\Phi}$ is not reversible, the second moment of return times is given by Eq.* (11).

Proof Using (Aldous and Fill 2002, Eq 2.21), we have

$$\mathbb{E}\left[\xi^2\right] = \frac{1 + 2\mathbb{E}_{\pi_{\boldsymbol{\Phi}}}(T_{x_0})}{\pi_{\boldsymbol{\Phi}}(x_0)}, \tag{11}$$

where $\mathbb{E}_{\pi_{\boldsymbol{\Phi}}}(T_{x_0})$ is the expected hitting time of x_0 from the steady state. Combining (Aldous and Fill 2002, Lemma 2.11 & Eq 3.41) and accounting for continuization yields $\mathbb{E}_{\pi_{\boldsymbol{\Phi}}}(T_{x_0}) \leq \frac{1}{\pi_{\boldsymbol{\Phi}}(x_0)\delta(\boldsymbol{\Phi})}$ and therefore, $\mathbb{E}\left[\xi^2\right] \leq \frac{1+\frac{2}{\pi_{\boldsymbol{\Phi}}(x_0)\delta(\boldsymbol{\Phi})}}{\pi_{\boldsymbol{\Phi}}(x_0)} < \frac{3}{\pi_{\boldsymbol{\Phi}}(x_0)^2\delta(\boldsymbol{\Phi})}$ because $\pi_{\boldsymbol{\Phi}}(x_0)$ and $\delta(\boldsymbol{\Phi})$ lie in the interval $(0, 1)$. □

Proposition 6 *Given a positive recurrent Markov chain $\boldsymbol{\Phi}$ over state space $\mathcal{S}$ and a set of m* RWT*s $\mathcal{T}$ and assuming an arbitrary ordering over $\mathcal{T}$, where $\mathbf{X}^{(i)}$ is the ith* RWT *in $\mathcal{T}$, $\mathbf{X}^{(i)}$ and $|\mathbf{X}^{(i)}|$ are i.i.d. processes such that $\mathbb{E}[|\mathbf{X}^{(i)}|] < \infty$, and when the tours are stitched together as defined next, the sample path is governed by $\boldsymbol{\Phi}$. For $t \geq 1$, define $\Phi_t = X^{N_t}_{t-R_{N_t}}$, where $R_i = \sum_{i'=1}^{i-1} |\mathbf{X}^i|$ when $i > 1$ and $R_1 = 0$ and $N_t = \max\{i : R_i < t\}$.*

Proof R_i is a sequence of stopping times. Therefore, the strong Markov property (Bremaud 2001, 2-Thm.7.1) states that sample paths before and after R_i are independent and are governed by $\boldsymbol{\Phi}$. Because $\boldsymbol{\Phi}$ is positive recurrent and x_0 is visited i.o., the regenerative cycle theorem (Bremaud 2001, 2-Thm.7.4) states that these trajectories are identically distributed and are equivalent to the tours $\mathcal{T}$ sampled according to Definition 2. $\mathbb{E}[|\mathbf{X}(i)|] < \infty$ due to positive recurrence. □

B.2 Proof of Lemma 1

Proof (*Unbiasedness and Consistency*) Because $\mathcal{G}$ is connected, $\boldsymbol{\Phi}$ is positive recurrent with steady state $\pi_{\boldsymbol{\Phi}}(u) \propto \mathbf{d}(u)$ due to Proposition 5. Consider the reward process $F^{(i)} = \sum_{j=1}^{|\mathbf{X}^{(i)}|} f(X^{(i)}_j, X^{(i)}_{j+1})$, $i \geq 1$. From Proposition 6, $F^{(i)}$ and $|\mathbf{X}^{(i)}|$ are i.i.d. sequences with finite first moments, because $F^{(i)} \leq B|\mathbf{X}^{(i)}|$. Let N_t and R_i be as defined in Proposition 6.

Therefore, from the renewal reward theorem (Bremaud 2001, 3-Thm.4.2), we have

$$\frac{\mathbb{E}\left[F^{(i)}\right]}{\mathbb{E}\left[|\mathbf{X}^{(i)}|\right]} = \lim_{t\to\infty} \frac{\sum_{i=1}^{N_t} F^{(i)}}{t} = \lim_{t\to\infty} \frac{\sum_{i=1}^{N_t} F^{(i)}}{R_{N_t}} \cdot \frac{R_{N_t}}{t} = \frac{\sum_{i=1}^{N_t} F^{(i)}}{R_{N_t}},$$

where the final equality holds because $\lim_{t\to\infty} \frac{R_{N_t}}{t} = 1 - \lim_{t\to\infty} \frac{t-R_{N_t}}{t}$, and $\lim_{t\to\infty} \frac{t-R_{N_t}}{t}$ converges to 0 as $t \to \infty$ because $|\mathbf{X}^{(\cdot)}| < \infty$ w.p. 1 because $\boldsymbol{\Phi}$ is positive recurrent.

From Proposition 6 and the definition of $F^{(i)}$, $\sum_{i=1}^{N_t} F^{(i)} = \sum_{j=1}^{R_{N_t}} f(\Phi_j, \Phi_{j+1})$, and because f and $\pi_{\boldsymbol{\Phi}}$ are bounded, we have from the ergodic theorem (Bremaud 2001, 3-Cor.4.1),

$$\frac{\mathbb{E}\left[F^{(i)}\right]}{\mathbb{E}\left[|\mathbf{X}^{(i)}|\right]} = \lim_{t\to\infty} \frac{\sum_{j=1}^{R_{N_t}} f\left(\Phi_j, \Phi_{j+1}\right)}{R_{N_t}} \overset{a.s}{=} \sum_{(u,v)\in\mathcal{V}\times\mathcal{V}} \pi_{\boldsymbol{\Phi}}(u)\, p_{\boldsymbol{\Phi}}(u,v)\, g(u,v) = \frac{2\mu(\mathcal{E})}{2|\mathcal{E}|}\,.$$

From Kac's formula (Aldous and Fill 2002, Cor.2.24), $1/\mathbb{E}[|\mathbf{X}^{(i)}|] = \pi_{\boldsymbol{\Phi}}(x_0) = \frac{\mathbf{d}(x_0)}{2|\mathcal{E}|}$, and

$$\mathbb{E}\left[\frac{\mathbf{d}(x_0)}{2} F^{(i)}\right] \overset{a.s}{=} \mu(\mathcal{E})\,.$$

$\hat{\mu}_*(\mathcal{T}; f, \mathcal{G})$ is unbiased by linearity of expectations on the summation over $\mathcal{T}$, and consistency is a consequence of Kolmogorov's SLLN (Bremaud 2001, 1-Thm.8.3). □

Proof (*Running Time*) From Kac's formula (Aldous and Fill 2002, Cor.2.24), $\mathbb{E}[|\mathbf{X}^{(i)}|] = \frac{2|\mathcal{E}|}{\mathbf{d}(x_0)}$. From Proposition 6, tours can be sampled independently and thus parallelly. All cores will sample an equal number of tours in expectation, yielding the running time bound. □

Proof (*Variance*) Because $f(\cdot) < B$, and tours are i.i.d., the variance is

$$\mathrm{Var}\left(\hat{\mu}_*(\mathcal{T})\right) = \mathrm{Var}\left(\frac{\mathbf{d}(x_0)}{2m} \sum_{\mathbf{X}\in\mathcal{T}} \sum_{j=1}^{|\mathbf{X}|} f(X_j, X_{j+1})\right) \leq \frac{\mathbf{d}(x_0)^2 B^2}{4m} \mathrm{Var}\left(|\mathbf{X}|\right)\,.$$

From Lemma 2 and Kac's formula (Aldous and Fill 2002, Cor.2.24), Var $(|\mathbf{X}|)$ is given by

$$\mathrm{Var}\left(|\mathbf{X}|\right) \leq \frac{3}{\pi_{\boldsymbol{\Phi}}(x_0)^2 \delta(\boldsymbol{\Phi})} - \frac{1}{\pi_{\boldsymbol{\Phi}}(x_0)^2} \leq \frac{3}{\pi_{\boldsymbol{\Phi}}(x_0)^2 \delta(\boldsymbol{\Phi})} = \frac{12|\mathcal{E}|^2}{\mathbf{d}(x_0)^2 \delta(\boldsymbol{\Phi})}\,.$$

□

C Proofs for Section Crefsec.estimator

Assumption 2 For each $\mathcal{G}_r$, $1 < r \leq R$ from Definition 5, assume $\mathbf{d}(\zeta_r)$ is known and that $p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$ can be sampled from.

Proposition 7 *(*RWT*s in* $\boldsymbol{\Phi}_r$*) Under Assumption 2, given access only to the original chain* $\boldsymbol{\Phi}$ *and stratifying function* ρ*, let* $\boldsymbol{\Phi}_r$ *be the random walk in the graph stratum*

$\mathcal{G}_r$ from Definition 5. To sample an RWT $(X_i)_{i=1}^{\xi}$ *over* $\boldsymbol{\Phi}_r$ *from the supernode* ζ_r*, we set* $X_1 = \zeta_r$*, sample* $X_2 \sim p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$*, and then, until* $\rho(X_{\xi+1}) < r$*, we sample*

$$X_{i+1} \sim \text{UNIF}\left(\mathbf{N}_{G_r}(X_i)\right) \equiv \begin{cases} \text{UNIF}\left(\mathbf{N}_{\mathcal{G}}(X_i)\right) & \text{if } \rho(X_i) = r \\ \text{UNIF}\left(\mathbf{N}_{\mathcal{G}}(X_i) \cap \mathcal{I}_r\right) & \text{if } \rho(X_i) > r \end{cases}.$$

Proof The proof is a direct consequence of Definitions 5 and 1. □

Proposition 8 (Perfectly Stratified Estimate) *Under Assumption 2, given the* EPS *(Definition 6) stratum $\mathcal{G}_r$ (Definition 5), bounded $f : \mathcal{E} \to \mathbb{R}$ and a set of m* RWT*s $\mathcal{T}_r$ over $\boldsymbol{\Phi}_r$ from ζ_r from Proposition 7, the per stratum estimate is given by*

$$\hat{\mu}(\mathcal{T}_r; f, \mathcal{G}_r) = \frac{\mathbf{d}(\zeta_r)}{2m} \sum_{\mathbf{X} \in \mathcal{T}_r} \sum_{j=2}^{|\mathbf{X}|-1} f\left(X_j, X_{j+1}\right), \tag{12}$$

where X_j is the jth state visited in the RWT $\mathbf{X} \in \mathcal{T}_r$*. For all $r > 1$, $\hat{\mu}(\mathcal{T}_r; f, \mathcal{G}_r)$ is an unbiased and consistent estimator of $\mu(\mathcal{J}_r) = \sum_{(u,v)\in\mathcal{J}_r} f(u, v)$, where $\mathcal{J}_r$ is the r-th edge stratum defined in Definition 4.*

Proof Define $f' : \mathcal{E}_r \to \mathbb{R}$ as $f'(u, v) \triangleq \mathbf{1}\{u, v \neq \zeta_r\} f(u, v)$. By Definition 2, in each *RWT* $\mathbf{X} \in \mathcal{T}_r$, $f'(X_1, X_2) = f'(X_{|\mathbf{X}|}, X_{|\mathbf{X}|+1}) = 0$, and therefore, $\hat{\mu}(\mathcal{T}_r; f, \mathcal{G}_r) = \hat{\mu}_*(\mathcal{T}; f', \mathcal{G}_r)$, where $\hat{\mu}_*$ is the *RWT* Estimate from Lemma 1. As $\mathcal{G}_r$ is connected, $\mathbb{E}\left[\hat{\mu}(\mathcal{T}_r; f, \mathcal{G}_r)\right] = \mathbb{E}\left[\hat{\mu}_*(\mathcal{T}; f', \mathcal{G}_r)\right] = \sum_{(u,v)\in\mathcal{E}_r} f'(u, v) = \sum_{(u,v)\in\mathcal{J}_r} f(u, v)$, where the final equality holds because $\mathcal{E}_r$ is the union of $\mathcal{J}_r$ and edges incident on the supernode. Consistency is also due to Lemma 1. □

C.1 Proof of Proposition 1

Proof Proposition 1 (a) is necessary because when Proposition 1 (a) does not hold, there exists a component such that the minimum value of ρ in that component is $\ddot{r} > 0$ such that in $\mathcal{G}_{\ddot{r}}$ (Definition 5), and the supernode $\zeta_{\ddot{r}}$ will be disconnected from all vertices. If Proposition 1 (b) is violated, a vertex $\ddot{u}$ exists that is disconnected in $\mathcal{G}_{\rho(\ddot{u})}$, and if Proposition 1 (c) is violated, the supernode is disconnected. Finally, it is easily seen that these conditions sufficiently guarantee that each stratum is connected, and the stratification is an *EPS*. □

C.2 Proof of Theorem 1

We begin by defining the multi-set containing the end points of edges between vertex strata.

Definition 12 Given $\mathcal{G}$ stratified into R strata, $\forall 1 \leq q < t \leq R$ define border multi-sets as $\mathcal{B}_{q,t} \triangleq \{v \,\forall (u, v) \in \mathcal{E} : u \in \mathcal{I}_q \text{ and } v \in \mathcal{I}_t\}$. The degree of the supernode in $\mathcal{G}_r$ (Definition 5) is then given by $\mathbf{d}(\zeta_r) = \sum_{q=1}^{r-1} |\mathcal{B}_{q,r}|$, and transitions out of ζ_r can be sampled by sampling $q \in \{1, \ldots, r-1\}$ w.p. $\propto |\mathcal{B}_{q,r}|$ and then by uniformly sampling from $\mathcal{B}_{q,r}$.

Proposition 9 *Given the setting in Definitions 8 and 9, for all* $1 \leq r < t \leq R$,

$$\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_r^\dagger|\to\infty} \widehat{\beta}_{r,t} \overset{a.s}{=} |\mathcal{B}_{r,t}|\,, \tag{13}$$

$$\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_r^\dagger|\to\infty} \widehat{\mathbf{U}}_{r,t} \sim \text{UNIF}(\mathcal{B}_{r,t})\,, \tag{14}$$

$$\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_r^\dagger|\to\infty} p_{\widehat{\boldsymbol{\Phi}}_r}(\mathbf{X}) = p_{\boldsymbol{\Phi}_r}(\mathbf{X})\,, \quad \forall \mathbf{X} \in \mathcal{T}_r^\dagger\,. \tag{15}$$

i.e., each tour in $\mathcal{T}_r^\dagger$ *is perfectly sampled from* $\boldsymbol{\Phi}_r$.

Proof (*By Strong Induction*) The base case for $r = 1$ holds by the base case in Definition 8. Now assume that Proposition 9 holds for all strata up to and including $r-1$. Because of the inductive claim and by Definition 12, $\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_{r-1}^\dagger|\to\infty}, \widehat{\mathbf{d}}(\zeta_r) = \sum_{q=1}^{r-1} \widehat{\beta}_{q,r} \overset{a.s}{=} \sum_{q=1}^{r-1} |\mathcal{B}_{q,r}| = \mathbf{d}(\zeta_r)$, and similarly, $\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_{r-1}^\dagger|\to\infty}, \widehat{p}_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot) \equiv p_{\boldsymbol{\Phi}_r}(\zeta_r, \cdot)$ because the inductive claim makes the procedure of sampling transitions out of ζ_r in Definition 8 equivalent to Definition 12. Equation (15) holds because transition probabilities at all states other than ζ_r are equivalent in $\boldsymbol{\Phi}_r$ and $\widehat{\boldsymbol{\Phi}}_r$ according to Definition 7. Now recall that $\widehat{\beta}_{r,t} = \frac{\widehat{\mathbf{d}}(\zeta_r)}{|\mathcal{T}_r^\dagger|} \sum_{\mathbf{X}\in\mathcal{T}_r^\dagger} \sum_{j=2}^{|\mathbf{X}|} \mathbf{1}\{\rho(X_j) = t\}$. Because $\widehat{\mathbf{d}}(\zeta_r) = \mathbf{d}(\zeta_r)$ and the tours are sampled perfectly, $\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_{r-1}^\dagger|\to\infty} \widehat{\beta}_{r,t} = \hat{\mu}_*\left(\mathcal{T}_r^\dagger; f'\right)$, where $f'(u,v) = \mathbf{1}\{\rho(v) = t\}$ and $\hat{\mu}_*$ is from Lemma 1, from which we also use the consistency guarantee to show that under an *EPS*, Eq. (13) holds as $\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_r^\dagger|\to\infty} \widehat{\beta}_{r,t} \overset{a.s}{=} \sum_{(u,v)\in\mathcal{E}_r} f'(u,v) = |\mathcal{B}_{r,t}|$. Because of Proposition 6, concatenating tours $\mathbf{X} \in \mathcal{T}_q^\dagger$ yields a sample path from $\boldsymbol{\Phi}_r$, and these samples are distributed according to $\pi_{\boldsymbol{\Phi}_r}$ as $|\mathcal{T}_{r'}^\dagger| \to \infty$, $r' \leq r$. Therefore, $\lim_{|\mathcal{T}_2^\dagger|\to\infty} \dots \lim_{|\mathcal{T}_r^\dagger|\to\infty} \uplus_{\mathbf{X}\in\mathcal{T}_q^\dagger} \uplus_{j=2}^{|\mathbf{X}|} \{X_j : \rho(X_j) = t\} \sim \pi'_{\boldsymbol{\Phi}_r}$, where $\pi'_{\boldsymbol{\Phi}_r}(u) \propto \mathbf{1}\{\rho(u) = t\}\mathbf{d}_{\mathcal{G}_r}(u)$, which is equivalent to $\text{UNIF}(\mathcal{B}_{r,t})$ by Definitions 5 and 12, thus proving Eq. (14). □

Proof (*Main Theorem*) Combining Propositions 9 and 8 proves Theorem 1. □

C.3 Proof of Theorem 2

Definition 13 (L^2 *Distance between* $\widehat{\pi}$ *and* π Aldous and Fill 2002) The L^2 distance between discrete probability distribution $\widehat{\pi}$ and reference distribution π with sample space Ω is given by $\|\widehat{\pi} - \pi\|_2 = \sum_{i\in\Omega} \frac{(\widehat{\pi}(i)-\pi(i))^2}{\pi(i)}$.

Definition 14 *(Distorted chain)* Given a Markov chain $\boldsymbol{\Phi}$ over finite state space $\mathcal{S}$ and an arbitrary $x_0 \in \mathcal{S}$, let $\widehat{\boldsymbol{\Phi}}$ be the distorted chain such that $\forall u \neq x_0$, $p_{\widehat{\boldsymbol{\Phi}}}(u,\cdot) = p_{\boldsymbol{\Phi}}(u,\cdot)$, and $p_{\widehat{\boldsymbol{\Phi}}}(x_0,\cdot)$ is an arbitrary distribution with support $\text{supp}(p_{\widehat{\boldsymbol{\Phi}}}(x_0,\cdot)) \subseteq \text{supp}(p_{\boldsymbol{\Phi}}(x_0,\cdot))$. The distortion is given by $\|p_{\widehat{\boldsymbol{\Phi}}}(x_0,\cdot) - p_{\boldsymbol{\Phi}}(x_0,\cdot)\|$ as defined in Definition 13.

Lemma 3 *Given a finite state, positive recurrent Markov chain $\boldsymbol{\Phi}$ over state space $\mathcal{S}$, let $\widehat{\boldsymbol{\Phi}}$ be the chain distorted at some $x_0 \in \mathcal{S}$ from Definition 14.*

Let $\mathcal{X} = \left\{(X_1, \ldots, X_\xi) \colon X_1 = x_0\,,\ \xi = \min\{t > 0 \colon X_{t+1} = x_0\}\,,\ p_{\boldsymbol{\Phi}}(X_1, \ldots, X_\xi) > 0\right\}$*, denote the set of all possible arbitrary lengths* RWT*s that begin and end at x_0 from Definition 2. Given a tour $\mathbf{Y} \in \mathcal{X}$ sampled from $\boldsymbol{\Phi}$ and a bounded function $F \colon \mathcal{X} \to \mathbb{R}$,*

$$\mathbb{E}_{\boldsymbol{\Phi}}\left[\frac{p_{\widehat{\boldsymbol{\Phi}}}\,(Y_1, Y_2)}{p_{\boldsymbol{\Phi}}\,(Y_1, Y_2)} F(\mathbf{Y})\right] = \mathbb{E}_{\widehat{\boldsymbol{\Phi}}}\left[F(\mathbf{Y})\right]\,, \tag{16}$$

where $\mathbb{E}_{\boldsymbol{\Phi}}$ and $\mathbb{E}_{\widehat{\boldsymbol{\Phi}}}$ are expectations under the distribution of tours sampled from $\boldsymbol{\Phi}$ and $\widehat{\boldsymbol{\Phi}}$.

Proof All tours in $\mathcal{X}$ are of finite length because of the positive recurrence of $\boldsymbol{\Phi}$. The ratio of the probability of sampling the tour $\mathbf{Y} = (Y_1, \ldots, Y_{\xi'})$ from the chain $\widehat{\boldsymbol{\Phi}}$ to $\boldsymbol{\Phi}$ is given by

$$\frac{p_{\widehat{\boldsymbol{\Phi}}}(\mathbf{Y})}{p_{\boldsymbol{\Phi}}(\mathbf{Y})} = \frac{\prod_{j=1}^{\xi'} p_{\widehat{\boldsymbol{\Phi}}}\left(Y_j, Y_{j+1}\right)}{\prod_{j=1}^{\xi'} p_{\boldsymbol{\Phi}}\left(Y_j, Y_{j+1}\right)} = \frac{p_{\widehat{\boldsymbol{\Phi}}}\,(Y_1, Y_2)}{p_{\boldsymbol{\Phi}}\,(Y_1, Y_2)}\,, \tag{17}$$

because $p_{\boldsymbol{\Phi}}(Y_j, \cdot) = p_{\widehat{\boldsymbol{\Phi}}}(Y_j, \cdot), \forall 1 < j \leq \xi'$ because $Y_j \neq x_0$ by the definitions of $\mathcal{X}$ and $\widehat{\boldsymbol{\Phi}}$. Because $\text{supp}(p_{\widehat{\boldsymbol{\Phi}}}(x_0, \cdot)) \subseteq \text{supp}(p_{\boldsymbol{\Phi}}(x_0, \cdot))$, $\text{supp}(p_{\widehat{\boldsymbol{\Phi}}}(\mathbf{Y})) \subseteq \text{supp}(p_{\boldsymbol{\Phi}}(\mathbf{Y}))$. The theorem statement therefore directly draws from the definition of importance sampling (Robert and Casella 2013, Def 3.9) with the importance weights derived in Eq. (17).

□

Lemma 4 *Given a simple random walk $\boldsymbol{\Phi}$ on the connected non-bipartite graph $\mathcal{G}$ from Definition 1, let $\widehat{\boldsymbol{\Phi}}$ be the chain distorted at some $x_0 \in \mathcal{S}$ from with distortion ν Definition 14. Let $\lambda = \widehat{\mathbf{d}}(x_0)/\mathbf{d}(x_0)$. Let $f \colon \mathcal{E} \to \mathbb{R}$ bounded by B, and $F(\mathbf{X}) = \sum_{j=1}^{|\mathbf{X}|} f(X_j, X_{j+1})$, where $\mathbf{X}$ is an* RWT *as defined in "Appendix B.2". The bias of an* RWT *Estimate*(Eq. 2) *computed using tours sampled over $\widehat{\boldsymbol{\Phi}}$ and using $\widehat{\mathbf{d}}(x_0)$ as the degree is given by* BIAS $= \left|\mathbb{E}_{\widehat{\boldsymbol{\Phi}}}\left[\frac{\widehat{\mathbf{d}}(x_0)}{2} F(\mathbf{X})\right] - \mu(\mathcal{E})\right| \leq (\lambda\nu + |1 - \lambda|)\,\frac{\sqrt{3}B|\mathcal{E}|}{\sqrt{\delta}}$*, where δ is the spectral gap of $\boldsymbol{\Phi}$, and B is the upper bound of f.*

Proof From Lemmas 3 and 1 we have

$$\mathbb{E}_{\widehat{\boldsymbol{\Phi}}}\left[\frac{\widehat{\mathbf{d}}(x_0)}{2} F(\mathbf{X})\right] = \mathbb{E}_{\boldsymbol{\Phi}}\left[\frac{\widehat{\mathbf{d}}(x_0)}{2}\frac{p_{\widehat{\boldsymbol{\Phi}}}\,(X_1, X_2)}{p_{\boldsymbol{\Phi}}\,(X_1, X_2)} F(\mathbf{X})\right],$$

$$\mu(\mathcal{E}) = \mathbb{E}_{\boldsymbol{\Phi}}\left[\frac{\mathbf{d}(x_0)}{2} F(\mathbf{X})\right].$$

Subtracting the two, squaring both sides and using the Cauchy–Schwarz inequality decomposes the squared bias into

$$\text{BIAS} = \left| \mathbb{E}_{\boldsymbol{\Phi}} \left[\left(\frac{\widehat{\mathbf{d}}(x_0)}{\mathbf{d}(x_0)} \frac{p_{\widehat{\boldsymbol{\Phi}}}(X_1, X_2)}{p_{\boldsymbol{\Phi}}(X_1, X_2)} - 1 \right) \frac{\mathbf{d}(x_0)}{2} F(\mathbf{X}) \right] \right| .$$

$$\text{BIAS}^2 \leq \underbrace{\mathbb{E}\left[\left(\frac{\widehat{\mathbf{d}}(x_0)}{\mathbf{d}(x_0)} \frac{p_{\widehat{\boldsymbol{\Phi}}}(x_0, X_2)}{p_{\widehat{\boldsymbol{\Phi}}}(x_0, X_2)} - 1 \right)^2 \right]}_{\text{BIAS}_{\text{dist}}} \underbrace{\mathbb{E}\left[\left(\frac{\mathbf{d}(x_0)}{2} F(\mathbf{X}) \right)^2 \right]}_{\text{BIAS}_{\text{spectral}}},$$

where the expectation is under $\boldsymbol{\Phi}$. Using definitions from the theorem statement,

$$\begin{aligned} \text{BIAS}_{\text{dist}} &= \frac{\widehat{\mathbf{d}}(x_0)^2}{\mathbf{d}(x_0)^2} \mathbb{E}\left[\left(\frac{p_{\widehat{\boldsymbol{\Phi}}}(x_0, X_2)}{p_{\boldsymbol{\Phi}}(x_0, X_2)} \right)^2 \right] + 1 - 2 \frac{\widetilde{\mathbf{d}}(x_0)}{\mathbf{d}(x_0)} \mathbb{E}\left[\frac{p_{\widehat{\boldsymbol{\Phi}}}(x_0, X_2)}{p_{\boldsymbol{\Phi}}(x_0, X_2)} \right] \\ &= \lambda^2(1 + \nu^2) + 1 - 2\lambda = \lambda^2 + \lambda^2\nu^2 + 1 - 2\lambda \\ &= \lambda^2\nu^2 + (1 - \lambda)^2 \leq (\lambda\nu + |1 - \lambda|)^2 . \end{aligned}$$

Because $F(\mathbf{X}) \leq B\xi$, the tour length, from Lemma 2, we see that

$$\text{BIAS}_{\text{spectral}} \leq \frac{\mathbf{d}(x_0)^2 B^2}{4} \frac{3}{\pi_{\boldsymbol{\Phi}}(x_0)^2 \delta} = \frac{3B^2 |\mathcal{E}|^2}{\delta} ,$$

and combining both biases completes the proof for BIAS. □

Proof (*Main Theorem*) Note that by linearity of expectations

$$\begin{aligned} \mathbb{E}\left[\hat{\mu}\left(\mathcal{T}^{\dagger}_{2:r}; f \right) | \mathcal{T}^{\dagger}_{2:r-1} \right] &= \mathbb{E}\left[\frac{\widehat{\mathbf{d}}(\zeta_r)}{2|\mathcal{T}^{\dagger}_r|} \sum_{\mathbf{X} \in \mathcal{T}^{\dagger}_r} \sum_{j=2}^{|\mathbf{X}|-1} f(X_j, X_{j+1}) \right] , \\ &= \mathbb{E}_{\mathbf{X} \sim \widehat{\boldsymbol{\Phi}}_r}\left[\frac{\widehat{\mathbf{d}}(\zeta_r)}{2} \sum_{j=1}^{|\mathbf{X}|} f'(X_j, X_{j+1}) \right] , \end{aligned}$$

where $\mathbf{X}$ is an *RWT* on $\widehat{\boldsymbol{\Phi}}_r$ that depends on $\mathcal{T}^{\dagger}_{2:r-1}$ and $f'(u, v) \triangleq \mathbf{1}\{u, v \neq \zeta_r\} f(u, v)$. Applying Lemma 4 completes the proof because $\widehat{\boldsymbol{\Phi}}_r$ is a distorted chain by Definition 14. □

D Proofs for Section 4

D.1 Proof of Proposition 2

Proof From Wang et al. (2014, Thm-3.1), we know that each disconnected component of G leads to a disconnected component in $\mathcal{G}^{k-1}$, and if $\mathcal{I}_1$ contains a subgraph in each connected component, Proposition 1 (a) is satisfied. We now prove that $\forall s \in \mathcal{V}^{k-1}$,

if $\rho(s) = r > 1, \exists\, s' \in \mathbf{N}(s)\colon \rho(s') < r$ which simultaneously satisfies Proposition 1 (b) and Proposition 1 (c) .

W.l.o.g. let the vertex with the smallest distance from the seed vertices be denoted by $\hat{u} = \text{argmin}_{u \in V(s)} \text{DIST}(u)$. When $\text{DIST}(\hat{u}) > 0$, there exists $v \in \mathbf{N}_G(\hat{u})$ such that $\text{DIST}(v) < \text{DIST}(\hat{u})$ by the definition of DIST. More concretely, v would be the penultimate vertex in the shortest path from the seed vertices to $\hat{u}$. Let $v' \neq \hat{u}$ be a nonarticulating vertex of s, which is possible because any connected graph has at least 2 nonarticulating vertices. Let $s_1 = G\left(V(s)\backslash\{v'\} \cup \{v\}\right) \in \mathcal{V}^{k-1}$. Now, $\rho(s_1) < \rho(s)$ because v' has been replaced with a vertex at necessarily a smaller distance and because the indicator in the definition of rho will always be 0 in this case. Moreover, $\circ s_1 s = G\left(V(s) \cup \{v\}\right) \in \mathcal{V}^k$, and hence an edge exists between the two.

When $\text{DIST}(\hat{u}) = 0$, there exists $v \in \mathbf{N}_G(\hat{u})$ such that $\text{DIST}(v) = 0$. There exists a nonarticulating $v' \in V(s)\backslash V^*$ because otherwise V^* would have been disconnected. Observing that $\text{DIST}(v') + \mathbf{1}\left\{v' \in V(\mathcal{I}_1)\backslash V^*\right\} > 0$ completes the proof of ergodicity. □

D.2 Proof of Proposition 3

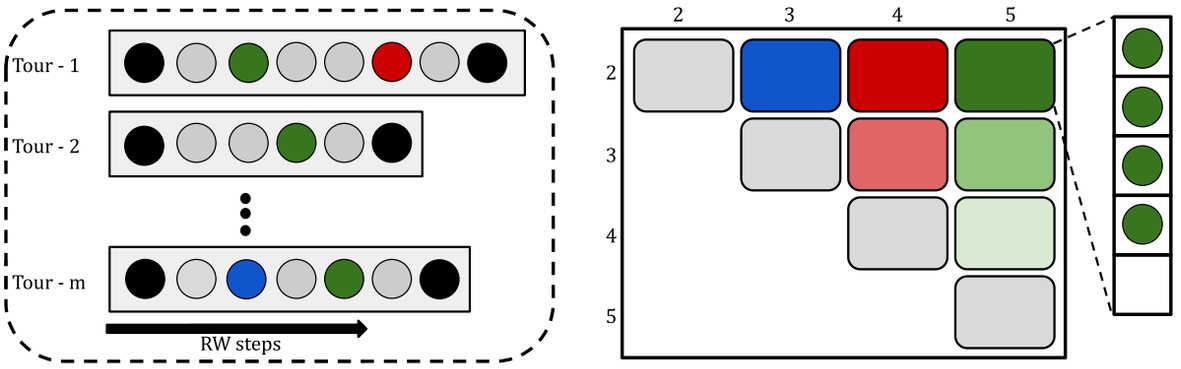
Proof (*Sampling Probability*) Consider the lines Lines 3 to 5. The probability of sampling the pair (u, v) from $V(s) \times \mathbf{N}_G\left(V(s)\right)$ is given by

$$\begin{aligned} P(u, v) &= \sum_{a \in V(s)\backslash\{u\}} P(v|a, u) P(a|u) P(u) \\ &= \sum_{a \in V(s)\backslash\{u\}} \frac{\mathbf{1}\{v \in \mathbf{N}(a)\}}{\mathbf{d}(a)} \frac{\mathbf{d}(a)}{\deg_s - \mathbf{d}(u)} \frac{\deg_s - \mathbf{d}(u)}{(k-1-1)\deg_s} \\ &\propto \sum_{a \in V(s)\backslash\{u\}} \mathbf{1}\{v \in \mathbf{N}(a)\} = |N(v) \cap V(s)\backslash\{u\}| = \text{BIAS}\,, \end{aligned}$$

where BIAS is defined in Line 6 and corrected for in Line 7. After the rejection, therefore, $(u, v) \sim \text{UNIF}(V(s) \times \mathbf{N}_G\left(V(s)\right))$.

Line 9 constitutes an importance sampling with unit weight for pairs (u, v), where removing u from and adding v to $V(s)$ produces a $k-1$−*CIS* and zero otherwise. In Line 9, because removing a nonarticulating vertex and adding another vertex to s cannot lead to a disconnected subgraph, we can avoid a DFS when $u \notin \mathcal{A}_s$. This completes the proof. □

Proof (*Time Complexity*) Assuming access to a precomputed vector of degrees, the part up to Line 1 is $\text{O}(k-1^2)$. In each proposal, Lines 3 and 4 are $\text{O}(k-1)$, and Line 5 is $\text{O}(\Delta_s)$. Line 6 is $\text{O}(k-1)$, and the expected complexity of Line 9 is $\text{O}(k-1^2\,|\mathcal{A}_s|/k-1)$ because in expectation only $|\mathcal{A}_s|/k-1$ graph traversals will be required. The acceptance probability is $\geq 1/k-1$ is Line 7 and $\geq \frac{k-1-|\mathcal{A}_s|}{k-1}$. The expected number of proposals is therefore $\leq \frac{k-1^2}{k-1-|\mathcal{A}_s|}$. As such, the expected time complexity is $\text{O}(k-1^2(1 + \frac{\Delta_s + k-1|\mathcal{A}_s|}{k-1-|\mathcal{A}_s|}))$. □



(a) $\mathcal{T}_2$, the set of *RWT*s sampled in $\mathcal{G}_2$. **(b)** The reservoir matrix, $\widehat{\mathbf{U}}_{r,t}$ for $2 \leq r < t \leq R$.

Fig. 4 Parallel *RWT*s and reservoirs: **a** The set of m *RWT*s sampled on $\mathcal{G}_2$ in parallel, where the supernode ζ_2 is colored black. The gray, blue, red and green colors represent states in stratum 2–5, respectively. **b** The upper triangular reservoir matrix in which the cell in the r-th row and t-th column contains samples from $\widehat{\mathbf{U}}_{r,t}$

E Additional implementation details

E.1 Parallel sampling with a reservoir matrix

Given a reasonably large M and the number of strata R, we initialize an upper triangular matrix of empty reservoirs $[\widehat{\mathbf{U}}_{r,t}]_{2\leq r<t\leq R}$ and a matrix of atomic counters $[\hat{\mathrm{M}}_{q,r}]_{2\leq r<t\leq R}$ initialized to 0. In each stratum r, while being sampled in parallel whenever a tour enters the t-th stratum, $\hat{\mathrm{M}}_{r,t}$ is incremented, and with a probability $\min(1, \mathrm{M}/\hat{\mathrm{M}}_{r,t})$, the state is inserted into a random position in the reservoir $\widehat{\mathbf{U}}_{r,t}$ and rejected otherwise. The only contention between threads in this scheme is at the atomic counter and in the rare case where two threads choose the same location to overwrite, wherein ties are broken based on the value of the atomic counter at the insertion time, guaranteeing thread safety. The space complexity of a reservoir matrix is therefore $O(R^2\mathrm{M})$.

A toy example of this matrix is presented in Fig. 4, where $R = 5$, and the *RWT*s are being sampled on the graph stratum $\mathcal{G}_2$. Whenever (non-gray) states in $\mathcal{I}_{3:5}$ are visited, they are inserted into the corresponding reservoirs–$\widehat{\mathbf{U}}_{2,5}$ is depicted in detail.

E.2 PSRW neighborhood

The neighborhood of a $k-CIS$ s in $\mathcal{G}^k$ is the set of all vertices $u, v \in V$ such that replacing u with v in s yields a $k-CIS$. Formally,

$$\mathbf{N}_{\mathcal{G}^k}(s) \equiv \left\{(u, v) \in V(s) \times \mathbf{N}_G\left(V(s)\right) : G\left(V(s) \cup \{v\} \backslash \{u\}\right) \in \mathcal{V}^k\right\}, \qquad (18)$$

where $\mathbf{N}_G(V(s)) = \cup_{x\in V(s)}\mathbf{N}_G(x)$ is the union of the neighborhood of each vertex in s. The size of the neighborhood is then $\mathrm{O}(k\,\mathbf{N}_G(V(s))) \in \mathrm{O}(k^2\Delta_G)$ because $\mathbf{N}_G(V(s)) \in \mathrm{O}(k\Delta_G)$, where Δ_G is the maximum degree in G. Each potential neigh-

bor further requires a connectivity check in the form of a BFS or DFS, which implies that the naive neighborhood sampling algorithm requires $O(k^4 \Delta_G)$ time.

E.2.1 Articulation points

Apart from the rejection sampling algorithm from Algorithm 1, we use articulation points to efficiently compute the subgraph bias γ from Eq. (9). Specifically, given the $k-1-CIS$, s, $\gamma(s) = \binom{\kappa - \mathcal{A}_s}{2}$, $\mathcal{A}_s$ is the set of articulation points of s. This draws directly from (Wang et al. 2014, Sec-3.3) and the definition of articulation points. Hopcroft and Tarjan (1973) showed that for any simple graph s the set of articulation points can be computed in $O(|V(s)| + |E(s)|)$ time.

Algorithm 2: Ripple for Subgraph Counting

Input: Input graph G, Order k, Set of subgraph patterns $\mathcal{H}$ of interest
Input: Initial vertex stratum $\mathcal{I}_1$, Reservoir Size M and Error Bound ϵ
Output: $\hat{\mu}$, an asymptotically unbiased estimate of $\mathcal{C}^k$

$\hat{\mu} = 0, \widehat{\beta}_{q,t} = 0, \widehat{\mathbf{U}}_{q,t} = \emptyset, \forall 1 \le q < t \le R$;
Run BFS for stratification $\rho\colon \mathcal{V}^{k-1} \to \{1, \ldots, R\}$, with $\mathcal{I}_1$ (Proposition 2)
/* Exact computation in the first stratum */
foreach $u \in \mathcal{I}_1, v \in \mathbf{N}_{\mathcal{G}^{k-1}}(u)$ **do**
  Update $\widehat{\beta}_{1,\rho(v)} += 1$, $\widehat{\mathbf{U}}_{1,\rho(v)} \cup = v$
  Update $\hat{\mu} += \left(\frac{\mathbf{1}\{\circ uv \sim H\}}{\gamma(\circ uv)}\right)_{H \in \mathcal{H}}$; // Equation (9)
for $r \in 2, \ldots, R$ **do**
  Initialize $\hat{\mu}_r = 0, m_r = 0$
  parallel while *Equation (10) is not satisfied* **do**
    Sample q from $\{1, \ldots, r-1\}$ w.p. $\widehat{\beta}_{q,r}$
    Sample u from $\widehat{\mathbf{U}}_{q,r}$; // Equation (6)
    Sample $v \sim \text{UNIF}(\mathbf{N}_{\mathcal{G}^{k-1}}(u))$; // Algorithm 1
    while $\rho(v) \ge r$ **do**
      Update $\hat{\mu}_r += \left(\frac{\mathbf{1}\{\circ uv \sim H\}}{\gamma(\circ uv)}\right)_{H \in \mathcal{H}}$; // Equation (8)
      if $\rho(v) > r$ **then**
        Update $\widehat{\beta}_{r,\rho(v)} += 1$; // Equation (4)
        Update $\widehat{\mathbf{U}}_{r,\rho(v)} \cup = v$; // Equation (6)
      $u := v$
      if $\rho(u) = r$ **then**
        Sample $v \sim \text{UNIF}(\mathbf{N}_{\mathcal{G}^{k-1}}(u))$
      else
        while $\rho(v) \ne r$ **do**
          Sample $v \sim \text{UNIF}(\mathbf{N}_{\mathcal{G}^{k-1}}(u))$
    $m_r += 1$
  Compute $\widehat{\deg}_r = \sum_{q=1}^{r-1} \widehat{\beta}_{q,r}$; // Equation (5)
  $\hat{\mu} += \frac{\widehat{\deg}_r}{2 m_r} \hat{\mu}_r$; // Equation (8) and (7)
  Update $\widehat{\beta}_{r,t} * = \frac{\widehat{\deg}_r}{m_r}, \forall t > r$; // Equation (4)
return $\hat{\mu}$

E.3 Proof of Proposition 4

Proposition 10 (Extended Version of Proposition 4) *We assume a constant number of tours m in each stratum and ignore graph loading. The Ripple estimator of $k-$*CIS *counts described in Algorithm 2 has space complexity in*

$$\mathrm{O}\left(k^3 D_G^2 \mathrm{M} + |\mathcal{H}|\right) \equiv \widehat{\mathrm{O}}\left(k^3 + |\mathcal{H}|\right),$$

where $\widehat{\mathrm{O}}$ ignores all factors other than k and $|\mathcal{H}|$, M *is the size of the reservoir from Sect. 4.3, D_G is the diameter of G, and $|\mathcal{H}|$ is the number of patterns of interest.*

The total number of random walk steps is given by $\mathrm{O}(k^3 m D_G \Delta_G C_{\mathrm{REJ}})$, where C_{REJ} is the number of rejections in Line 21 of Algorithm 2, Δ_G is the largest degree in G, and the total time complexity is $\widehat{\mathrm{O}}(k^7 + |\mathcal{H}|)$.

Remark 2 In practice, we adapt the proposals in Algorithm 1 to minimize C_{REJ} using heuristics over the values of $\textsc{dist}(\cdot)$ from Proposition 2.

Lemma 5 *Given a graph stratum $\mathcal{G}_r$ from Definition 5, for some $r > 1$, define $\alpha_r = |\{u \in \mathcal{I}_r : \mathbf{N}(u) \cap \mathcal{I}_{1:r-1} \neq \emptyset\}| / |\mathcal{I}_r|$ as the fraction of vertices in the r-th vertex stratum that share an edge with a previous stratum. The return time ξ_r of the chain $\boldsymbol{\Phi}_r$ to the supernode $\zeta_r \in \mathcal{V}_r$ follows $\mathbb{E}_{\boldsymbol{\Phi}_r}[\xi_r] \leq \frac{2\bar{\mathbf{d}}_r}{\alpha_r}$, where $\bar{\mathbf{d}}_r$ is the average degree in $\mathcal{G}$ of all vertices in $\mathcal{I}_r$.*

Proof Because $\alpha_r \mathcal{I}_r$ vertices have at least one edge incident on ζ_r, $\mathbf{d}_{\mathcal{G}_r}(\zeta_r) \geq \alpha_r \mathcal{I}_r$. From Definition 5, because all edges not incident on $\mathcal{I}_r$ are removed from $\mathcal{G}_r$, $\mathrm{Vol}(\mathcal{G}_r) \leq 2 \sum_{u \in \mathcal{I}_r} \mathbf{d}_{\mathcal{G}}(u)$. Therefore, from Lemma 1,

$$\mathbb{E}_{\boldsymbol{\Phi}_r}[\xi_r] = \frac{\mathrm{Vol}(\mathcal{G}_r)}{\mathbf{d}(\zeta_r)} \leq \frac{2 \sum_{u \in \mathcal{I}_r} \mathbf{d}_{\mathcal{G}}(u)}{\alpha_r \mathcal{I}_r} = \frac{2\bar{\mathbf{d}}_r}{\alpha_r}.$$

□

Proposition 11 *The Ergodicity-Preserving Stratification from Proposition 2 is such that $\alpha_r = 1$ for all $r > 1$ as defined in Lemma 5, and consequently, the diameter of each graph stratum is ≤ 4. The total number of strata $R \in \mathrm{O}(k\, D_G)$, where D_G is the diameter of G.*

Proof We show in "Appendix D.1" that for each vertex $s \in \mathcal{V}^{k-1}$, if $\rho(s) = r > 1$, there exists $s' \in \mathbf{N}(s)$ such that $\rho(s') < r$. This implies that $\alpha_r = 1$. In $\mathcal{G}_r$, therefore, from ζ_r, all vertices in $\mathcal{I}_r$ are at unit distance from ζ_r, and vertices in $\mathbf{N}(\mathcal{I}_r) \backslash \mathcal{I}_r$ are at a distance of 2 from ζ_r. Because no other vertices are present in $\mathcal{G}_r$, this completes the proof of the first part. Trivially, $R \leq (k-1) \cdot \max_{u \in V} \textsc{dist}(u) \in \mathrm{O}(k \cdot D_G)$. □

Proof (*Memory Complexity*) From Algorithm 2, we compute a single count estimate per stratum and maintain reservoirs and inter-partition edge count estimates for each $2 \leq q < t \leq R$. Because a reservoir $\widehat{\mathbf{U}}_{q,t}$ needs $\mathrm{O}(k\mathrm{M})$ space ("Appendix E.1"), the total memory requirement is $\mathrm{O}(R^2\, k\mathrm{M})$, where R is the number of strata. From Proposition 11, plugging $R \in \mathrm{O}(kD_G)$, and because storing the output $\hat{\mu}$ requires $\mathrm{O}(|\mathcal{H}|)$ memory the proof is completed. □

Proof (*Time Complexity*) The stratification requires a single BFS $\in \mathrm{O}(|V|+|E|)$ from Sect. 4.2. In Line 3, the estimation phase starts by iterating over the entire higher-order neighborhood of each subgraphs in $\mathcal{I}_1$. Based on "Appendix E.2.1", Line 5 is in $\mathrm{O}(k^2)$. Because the size of the higher-order neighborhood of each subgraph is $\mathrm{O}(k^2\Delta_G)$ from "Appendix E.2", the initial estimation phase will require $\mathrm{O}(|\mathcal{I}_1|\,k^4\Delta_G)$ time.

In all other strata $r = 2, \ldots, R$, we assume that m tours are sampled in Line 8. Starting each tour (Lines 9 to 11) requires order of magnitude R time, leading to a total time of $\mathrm{O}(m\,R^2) \in \mathrm{O}(mk^2D_G^2)$ because $R \in \mathrm{O}(k\,D_G)$ from Proposition 11. The total time for these ancilliary procedures is $\mathrm{O}(mk^2D_G^2 + |\mathcal{I}_1|\,k^4\Delta_G)$

Therefore, the time complexity of bookkeeping and setup is $\mathrm{O}(mk^2D_G^2 + |\mathcal{I}_1|\,k^4\Delta_G + |V| + |E|) \in \widehat{\mathrm{O}}(k^4)$. The time complexity at each random walk step is $\mathrm{O}(k-1^2\Delta_G + k-1^4) \in \widehat{\mathrm{O}}(k^4)$ from "Appendix D.2" D.2 and "Appendix E.2.1". We assume that the expected number of rejections in Line 21 is given by C_{REJ}. The total number of random walk steps is given by $\mathrm{O}(R\,m\,C_{\mathrm{REJ}})$ times the expected tour length. By Lemma 5 and Proposition 11, the expected tour length is $\mathrm{O}(\Delta_{\mathcal{G}^{k-1}}) \equiv \mathrm{O}(k^2\Delta_G)$. Therefore, the total number of random walk steps is $\mathrm{O}(k^3 m D_G \Delta_G C_{\mathrm{REJ}})$.

$\mathrm{O}(|\mathcal{H}|)$ time is to print the output $\hat{\mu}$. We assume that updating $\hat{\mu}$ is amortized in constant order if we use a hashmap to store elements of the vector, and because updating a single key in said hashmap is by Eq. (9) increments, the proof is completed. □

References

Aldous D, Fill JA (2002) Reversible Markov chains and random walks on graphs. Unfinished monograph, recompiled 2014. Available at http://www.stat.berkeley.edu/~aldous/RWG/book.html

Athreya KB, Ney P (1978) A new approach to the limit theory of recurrent Markov chains. Trans Am Math Soc 245:493–501

Avrachenkov K, Ribeiro B, Sreedharan JK (2016) Inference in osns via lightweight partial crawls. In: ACM SIGMETRICS, pp 165–177

Avrachenkov K, Borkar VS, Kadavankandy A, Sreedharan JK (2018) Revisiting random walk based sampling in networks: evasion of burn-in period and frequent regenerations. Comput Soc Netw 5(1):1–19

Bhuiyan MA, Rahman M, Rahman M, Al Hasan M (2012) Guise: uniform sampling of graphlets for large graph analysis. In: IEEE 12th international conference on data mining. IEEE, pp 91–100

Bremaud P (2001) Markov chains: Gibbs Fields, Monte Carlo simulation, and queues. Texts in applied mathematics. Springer, New York

Bressan M, Chierichetti F, Kumar R, Leucci S, Panconesi A (2018) Motif counting beyond five nodes. ACM TKDD 12(4):1–25. https://doi.org/10.1145/3186586

Bressan M, Leucci S, Panconesi A (2019) Motivo: fast motif counting via succinct color coding and adaptive sampling. In: Proc VLDB Endow

Chen X, Lui JC (2018) Mining graphlet counts in online social networks. ACM TKDD 12(4):1–38

Cooper C, Radzik T, Siantos Y (2016) Fast low-cost estimation of network properties using random walks. Internet Math 12:221–238. https://doi.org/10.1080/15427951.2016.1164100

Diaconis P, Stroock DW (1991) Geometric bounds for eigenvalues of Markov chains. Ann Appl Probab 1:36–61

Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE PAMI PAMI 6(6):721–741

Geyer CJ (1992) Practical Markov chain Monte Carlo. Stat Sci 7:473–483

Glynn PW, Rhee Ch (2014) Exact estimation for Markov chain equilibrium expectations. J Appl Probab 51(A):377–389

Han G, Sethu H (2016) Waddling random walk: fast and accurate mining of motif statistics in large graphs. In: ICDM. IEEE, pp 181–190
Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications
Hopcroft J, Tarjan R (1973) Algorithm 447: efficient algorithms for graph manipulation. Commun ACM 16(6):372–378
Iyer AP, Liu Z, Jin X, Venkataraman S, Braverman V, Stoica I (2018) {ASAP}: fast, approximate graph pattern mining at scale. In: OSDI, pp 745–761
Jacob PE, O'Leary J, Atchadé YF (2020) Unbiased Markov chain monte Carlo methods with couplings. JRSS Ser B 82(3):543–600
Jain S, Seshadhri C (2017) A fast and provable method for estimating clique counts using turán's theorem. In: WWW, WWW '17, pp 441–449
Jain S, Seshadhri C (2020) Provably and efficiently approximating near-cliques using the turán shadow: peanuts. WWW 2020:1966–1976
Kashtan N, Itzkovitz S, Milo R, Alon U (2004) Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics 20(11):1746–1758
Leskovec J, Krevl A (2014) SNAP datasets: stanford large network dataset collection. http://snap.stanford.edu/data
Liu G, Wong L (2008) Effective pruning techniques for mining quasi-cliques. In: ECML-PKDD
Massoulié L, Le Merrer E, Kermarrec AM, Ganesh A (2006) Peer counting and sampling in overlay networks: random walk methods. In: PODC
Matsuno R, Gionis A (2020) Improved mixing time for k-subgraph sampling. In: Proceedings of the (2020) SIAM international SIAM, conference on data mining, pp 568–576
Mykland P, Tierney L, Yu B (1995) Regeneration in Markov Chain samplers. J Am Stat Assoc 90(429):233–241
Neal RM (2001) Annealed importance sampling. Stat Comput 11(2):125–139
Neiswanger W, Wang C, Xing EP (2014) Asymptotically exact, embarrassingly parallel MCMC. UAI
Nummelin E (1978) A splitting technique for Harris recurrent Markov Chains. Mag Probab Theory Relat Areas 43(4):309–318
Pinar A, Seshadhri C, Vishal V (2017) Escape: efficiently counting all 5-vertex subgraphs. In: WWW, pp 1431–1440
Propp JG, Wilson DB (1996) Exact sampling with coupled Markov chains and applications to statistical mechanics. Random Struct Algor 9(1-2):223–252
Ribeiro B, Towsley D (2012) On the estimation accuracy of degree distributions from graph sampling. In: CDC
Ribeiro P, Paredes P, Silva ME, Aparicio D, Silva F (2019) A survey on subgraph counting: concepts, algorithms and applications to network motifs and graphlets. arXiv:191013011
Robert C, Casella G (2013) Monte Carlo statistical methods. Springer, Berlin
Rosenthal JS (1995) Minorization conditions and convergence rates for Markov Chain Monte Carlo. J Am Stat Assoc
Savarese P, Kakodkar M, Ribeiro B (2018) From Monte Carlo to Las Vegas: Improving restricted Boltzmann machine training. In: AAAI
Sinclair A (1992) Improved bounds for mixing rates of Markov Chains and multicommodity flow. Comb Probab Comput 1(4)
Teixeira CH, Cotta L, Ribeiro B, Meira W (2018) Graph pattern mining and learning through user-defined relations. In: ICDM. IEEE, pp 1266–1271
Teixeira CHC, Kakodkar M, Dias V, Meira Jr W, Ribeiro B (2021) Sequential stratified regeneration: MCMC for large state spaces with an application to subgraph count estimation. arXiv:2012.03879
Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Software (TOMS) 11(1):37–57
Wang P, Lui JCS, Ribeiro B, Towsley D, Zhao J, Guan X (2014) Efficiently estimating motif statistics of large networks. ACM TKDD 9(2):1–7
Wang P, Zhao J, Zhang X, Li Z, Cheng J, Lui JCS, Towsley D, Tao J, Guan X (2017) MOSS-5: a fast method of approximating counts of 5-node graphlets in large graphs. IEEE Trans Knowl Data Eng 30(1):73–86
Wernicke S (2006) Efficient detection of network motifs. IEEE/ACM Trans Comput Biol Bioinf 3(4):347–359
Wilkinson DJ (2006) Parallel Bayesian computation. Statist Textbooks Monogr 184:477

Yang C, Lyu M, Li Y, Zhao Q, Xu Y (2018) Ssrw: a scalable algorithm for estimating graphlet statistics based on random walk. In: International Springer, conference on database systems for advanced applications, pp 272–288

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

TipTap: Approximate Mining of Frequent k-Subgraph Patterns in Evolving Graphs

MUHAMMAD ANIS UDDIN NASIR, King Digital Entertainment Ltd
CIGDEM ASLAY, Aarhus University
GIANMARCO DE FRANCISCI MORALES, ISI Foundation
MATTEO RIONDATO, Amherst College

"Perhaps he could dance first and think afterwards, if it isn't too much to ask him."

S. Beckett, *Waiting for Godot*

Given a labeled graph, the collection of k-vertex induced connected subgraph patterns that appear in the graph more frequently than a user-specified minimum threshold provides a compact summary of the characteristics of the graph, and finds applications ranging from biology to network science. However, finding these patterns is challenging, even more so for dynamic graphs that evolve over time, due to the streaming nature of the input and the exponential time complexity of the problem.

We study this task in both incremental and fully-dynamic streaming settings, where arbitrary edges can be added or removed from the graph. We present TipTap, a suite of algorithms to compute high-quality approximations of the frequent k-vertex subgraphs w.r.t. a given threshold, at any time (i.e., point of the stream), with high probability. In contrast to existing state-of-the-art solutions that require iterating over the entire set of subgraphs in the vicinity of the updated edge, TipTap operates by efficiently maintaining a uniform sample of connected k-vertex subgraphs, thanks to an optimized neighborhood-exploration procedure. We provide a theoretical analysis of the proposed algorithms in terms of their unbiasedness and of the sample size needed to obtain a desired approximation quality. Our analysis relies on sample-complexity bounds that use Vapnik–Chervonenkis dimension, a key concept from statistical learning theory, which allows us to derive a sufficient sample size that is independent from the size of the graph. The results of our empirical evaluation demonstrates that TipTap returns high-quality results more efficiently and accurately than existing baselines.

A preliminary version of this work appeared in the proceedings of ACM CIKM'18 as [2].
Part of the work done while M. A. U. Nasir was at KTH Royal Institute of Technology.
Part of the work done while C. Aslay was at Aalto University.
Cigdem Aslay is supported by the Academy of Finland projects 286211, 313927, and 317085, and the EC H2020 RIA project "SoBigData" 654024 (https://cordis.europa.eu/project/rcn/197882/en). Matteo Riondato is supported in part by the National Science Foundation project 2006765 (https://www.nsf.gov/awardsearch/showAward?AWD_ID=2006765).
Authors' addresses: M. A. U. Nasir, King Digital Entertainment Ltd, Midasplayer AB, Sveavägen 44, 111 34 Stockholm, Sweden; email: anis.nasir@king.se; C. Aslay, Department of Computer Science, Aarhus University Abogade 34, Aarhus, 8200, Denmark; email: cigdem@cs.au.dk; G. De Francisci Morales, ISI Foundation, Via Chisola 5, Turin, 10126, Italy; email: gdfm@acm.org; M. Riondato, Amherst College, 25 East Drive, Amherst, MA 01002, USA; email: mriondato@amherst.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1556-4681/2021/04-ART48 $15.00
https://doi.org/10.1145/3442590

CCS Concepts: • **Information systems → Data stream mining**; • **Theory of computation → Dynamic graph algorithms**; **Sketching and sampling**; • **Mathematics of computing → Approximation algorithms**; **Graph enumeration**;

Additional Key Words and Phrases: Edge streams, graph streams, reservoir sampling, random pairing, VC-dimension

ACM Reference format:
Muhammad Anis Uddin Nasir, Cigdem Aslay, Gianmarco De Francisci Morales, and Matteo Riondato. 2021. TipTap: Approximate Mining of Frequent k-Subgraph Patterns in Evolving Graphs. *ACM Trans. Knowl. Discov. Data* 15, 3, Article 48 (April 2021), 35 pages.
https://doi.org/10.1145/3442590

1 INTRODUCTION

Counting the number of occurrences of different subgraphs in a network is a fundamental graph-mining task with applications in various disciplines, including bioinformatics, security, and social sciences. There are many variants of the problem (see Section 2), depending on the definition of "occurrence," the set of considered subgraphs, and the assumed computational model. We focus on the variant requiring to find the set of k-vertex induced *connected* subgraph patterns (which we refer to as "k-patterns") that appear in a *labeled* graph with a frequency higher than a user-specified minimum threshold. Such subgraphs might be indicative of important protein interactions, common social norms, or frequent activity between users. This problem also finds applications in graph classification and indexing, as the frequency spectrum can be used as a fingerprint for the graph [6].

The bottleneck for subgraph mining algorithms on a single large graph is the computational complexity incurred by the two core operations: (i) efficient generation of all subgraphs of the given graph; and (ii) frequency evaluation which requires to determine whether a subgraph is an exact match of another. Thus, existing algorithms for subgraph mining do not scale to the large graphs that are now common in many domains. Moreover, these graphs usually arise from dynamic processes, hence are subject to continuous changes. For example, consider a brain network where the vertices represent areas of the brain that are associated with neurons or functional areas in the brain. Given electroencephalogram signal recordings for each vertex (i.e., area), an edge is present between two vertices at time t if the correlation between the two signals is above a user-defined threshold, with the label of the edge denoting the stimuli upon its creation, e.g., reading, resting, or watching a video [42]. The edge is removed when the correlation falls below the threshold. The graph changes with time, and we may be interested in which functional areas of the brain co-activate when subject to a given stimulus. As a further example, consider the knowledge graph DBpedia, which gets updated every day according to a stream of change logs from Wikipedia [21]. Vertices in this graph represent entities, with labels denoting their associated elements in a taxonomy, and edges represent the relationships between the entities. Frequent patterns represent frequent kind of relationships between multiple entities and may allow to discover interesting phenomena. The graph is subject to frequent updates, and any pattern extraction query cast on this knowledge graph should be answered on the latest snapshot of the graph. As another example of the importance of frequent patterns and their practical uses, consider a music streaming service, and a graph of its users, where users (nodes) are *temporarily* connected by an edge when they listen to the same song within a certain interval of time, only to become disconnected after some number of time steps. Analyzing the behavior of frequent patterns over time allows to study the evolving taste of users in music (see, for example, the study on the evolution of the number of triangles in such a graph by De Stefani et al. [13, Section 5.2]) and use this information for song recommendations.

Whenever the graph changes, a large number of new subgraphs can be created, and existing subgraphs can be modified or destroyed. Keeping track of the exact changes in the frequencies of all subgraph patterns at all time is thus highly challenging. Additionally, chasing the exact frequencies (and therefore the exact set of frequent patterns) in a dynamic environment has very limited value, due to the volatility of the frequencies: in these cases, approximations are usually sufficient, provided they come with strong (probabilistic) guarantees on their quality.

Contributions. We address the problem of computing high-quality approximations to the collection of the frequent k-vertex induced connected subgraphs (k-patterns) in an evolving graph represented as a stream of edge updates—additions or deletions. Concretely, our main contributions are the following.

- —We introduce TipTap (Section 4), the first suite of approximation algorithms for the collection of frequent k-patterns in evolving graphs. Few existing works consider this setting: most of them are actually limited to incremental streams, where edges can only be added, or they consider a different computational model (see Section 2).
- —Differently from existing works, TipTap keeps a fixed-sized sample of k-*subgraphs* rather than edges. This choice enables sampling any k-subgraph with equal probability, and thus simplifies dramatically the design of the unbiased frequency estimators. To maintain the uniformity of the sample after an edge update, TipTap relies on a principled scheme (Section 4.2), derived from reservoir sampling [39]. TipTap employs *random pairing* (RP) [16] to handle deletions in fully-dynamic streams (see Section 4.5).
- —We increase the efficiency of the sampling procedure during the exploration of the local neighborhood of the updated edge by modifying the "skip optimization" proposed by Vitter [39] for reservoir sampling (see Section 4.4) and by Gemulla et al. [17] for RP (see Section 4.4.1). Our efficient exploration is based on a novel label-agnostic partitioning of the subgraphs that become connected (resp. disconnected) after the insertion (resp. deletion) of an edge into structural equivalence classes whose size is easy to compute. This novel contribution, which is also of independent interest, is of key importance in order for TipTap to benefit from skip optimization.
- —TipTap computes high-quality ε-approximations to the collections of frequent subgraph patterns, where $\varepsilon \in (0, 1)$ is a desired quality parameter fixed by the user (see Definition 1). We show the unbiasedness of the estimators and prove an upper bound to the sample size sufficient to obtain an approximation of user-specified quality $\varepsilon \in (0, 1)$ with high probability. Our analysis (Section 4.1) is based on Vapnik–Chervonenkis (VC) dimension [38], a fundamental concept form statistical learning theory [37]: we define an appropriate range space for the task and show a strict bound on its VC-dimension, which is, perhaps surprisingly, independent from any property of the graph (see the proof of Lemma 4.3). As a result, we derive an upper bound to the sufficient sample size to obtain an ε-approximation that is also independent from the size of the graph (see Lemma 4.3). This result allows TipTap to be used on ever-growing graphs, a net advantage over other approaches.
- —In experiments, TipTap outclasses an edge-sampling baseline on all evaluation metrics, both in the approximation of the true pattern frequencies (e.g., average error and correlation coefficient) and in finding the most frequent patterns (e.g., precision and recall) (Section 5).

A preliminary version of our work [2] provided a first theoretical and experimental treatment of the problem with an emphasis on three patterns. In this article, we expand our preliminary results with several improvements and extensions. First, we provide tighter bounds to the sample size sufficient for computing a high-quality approximation to the collection of frequent subgraph

patterns. In particular, our analysis now employs VC-dimension: we prove that for our task, the VC-dimension is independent of any property of the graph, hence, translating to a sample size requirement that solely depends on the desired level of accuracy and failure probability. In contrast, our conference version used much looser Chernoff bounds and the union bound, resulting in a sample size dependent on k. Second, we provide improved algorithms in which we decouple the sample maintenance operations from the neighborhood exploration procedure for generic k-patterns. Accordingly, we propose a novel neighborhood exploration procedure, made more efficient, thanks to a thorough case analysis, and we also present efficient algorithms to count and sample newly (dis-)connected for four patterns with every edge update. Finally, we updated the experimental results for three patterns to integrate the aforementioned changes, and we provide a novel set of results for four patterns.

2 RELATED WORK

We now discuss the previous contributions most related to ours. Specifically, we focus on algorithms for subgraph counting on dynamic graphs represented as edge streams, and on Frequent Subgraph (pattern) Mining (FSM). For clarity and to keep the discussion focused, we do not discuss works on different settings, such as the transactional setting where the input is a stream of small graphs [5, 8], as the definition of frequency and the methods used in this setting are entirely different from the stream-of-edge-updates that we consider. We also do not discuss subgraph mining problems different from frequent pattern mining, such as finding the densest subgraphs [18], as density and frequency are quite orthogonal measures of interestingness.

Subgraph counting from edge streams. Most of the work in this area has focused on the traditional data stream setting assuming that the whole graph cannot fit into memory at any time but the earliest stages. The consequence of this assumption is that the algorithms need to build a sketch of the data that fits into the available space and use it to answer queries about the counts of subgraphs. Given the limited space, the answers would necessarily be approximations of the exact counts. Contributions in this area include, for example, many algorithms for triangle counting in incremental edge streams [23, 29, 32, 36] and in fully-dynamic streams [13]. Beyond triangles, Wang et al. [41] propose an algorithm that estimates the number of appearances of connected k-subgraphs from a uniform sample of edges in incremental streams, and De Stefani et al. [14] show how to use a multi-layered sample to count k-subgraphs. A recent work by Chen and Lui [11] examines approximate counting of incremental streams for different choice of edge sampling and probabilistic counting methods. Rossi et al. [34] also propose a sampling-based graphlet-counting framework which is amenable to streaming input, although it does not provide any approximation guarantee.

TipTap differs from these previous contributions in multiple ways. First, all but one of these works (De Stefani et al. [13]'s TRIÈST, which is only for triangles) are concerned with *insertion-only* streams, while we present algorithms for both insertion-only and fully-dynamic streams. Second, we deal with *labeled* graphs and subgraphs: although some but not all of the methods in previous works can be extended to this case, this requirement adds significant complexity to the algorithms. Obviously TipTap can also work on unlabeled graphs, as these can be seen as graphs with a single label for vertices and a single label for edges. Third, and perhaps most important, we do not assume the aforementioned *limited-memory* setting that is common to all of these works. Rather, in our model, it is possible to store and access the whole graph at all times. The limited-memory setting is interesting from a theoretical point of view, and realistic for many applications, such as real-time analysis on small-memory network devices, but in many relevant use cases of large-graph analytics, it is possible to store the entire graph in main memory (see, e.g., the experimental setting used to study the effective diameter of the entire Facebook graph [4].) Having direct access to the

Table 1. Main Notation Used in This Work

Symbol	Meaning
$G_S = (S, E(S))$	the subgraph of G induced by $S \subseteq V$.
$\mathcal{N}^t_{u,h}$	h-hop neighborhood of u at time t. We use $\mathcal{N}^t_u$ for the 1-hop neighborhood.
$C_{k,i}$	the set of k-subgraphs isomorphic to pattern P_i (in a specified graph).
C_k	the set of *all* k-subgraphs (in a specified graph).
N^t	the size of the population of k-subgraphs at time t, i.e., the size of C_k in G^t.
$f(P_i)$	the frequency of the pattern P_i (in a specified graph).
$\tilde{f}(P_i)$	the estimate of $f(P_i)$.
$\mathcal{F}^t_k(\tau)$	the collection of frequent k-patterns w.r.t. τ in the graph G^t (see Problem 3.1).
$\widetilde{\mathcal{F}}^t_k(\tau, \varepsilon)$	an ε-approximation to $\mathcal{F}^t_k(\tau)$ (see Def. 1).
$\mathcal{S}$	the sample of k-subgraphs.
M	the (maximum) size of the subgraph sample $\mathcal{S}$.

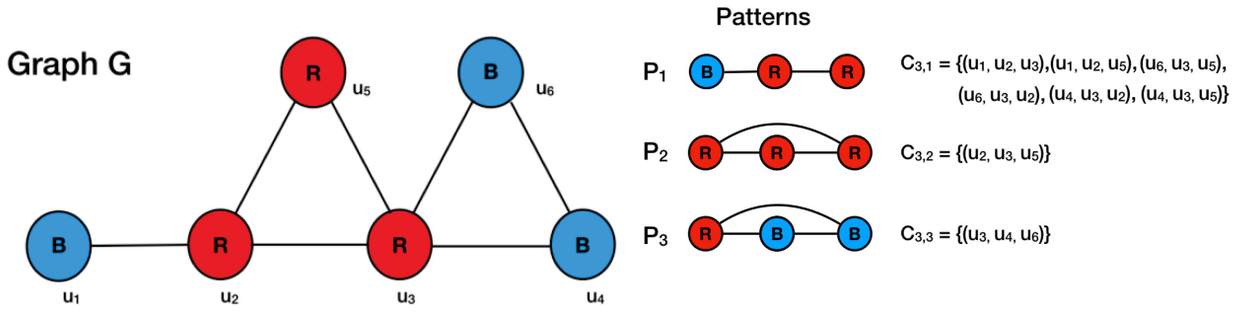
entire graph may in principle allow for exact (although extremely time consuming) computation of the frequencies of all subgraph patterns of interest, but the dynamic nature of the graph, which changes at all times, makes "chasing" these frequencies of little value, given both their volatility and the computational cost of keeping track of them. For this reason, we compute high-quality *approximations* of the frequencies by maintaining a uniform sample of *subgraphs*, rather than of *edges* as in all the works mentioned above.

Frequent Subgraph Mining. The problem of FSM was introduced by Inokuchi et al. [22] in the *transactional* setting. Here, the goal is to find all the frequent connected subgraph patterns *of all sizes* in a given dataset of many, usually small, graphs. A number of algorithms have been proposed for this task (see the survey by Jiang et al. [24]). This variant of FSM is very different from the problem we study, where there is a single graph that evolves over time and we are only interested in connected subgraph patterns over a fixed number k of vertices.

FSM has also been studied on a single static graph. None of proposed approaches [9, 10, 15, 25, 27], either exact or approximate, are applicable to streaming graphs. The closest to our setting is the work by Ray et al. [33], who consider a single graph with continuous updates. Their approach, however, is a heuristic applicable only to incremental streams and comes without any provable guarantee. TipTap works for fully-dynamic streams and offers strong probabilistic guarantees on the quality of the computed approximations. Abdelhamid et al. [1] propose an exact algorithm for FSM which borrows from the literature on incremental pattern mining. The algorithm keeps track of "fringe" subgraph patterns, which have frequency close to the frequency threshold, and all their possible expansions/contractions (by adding/removing one edge). While the algorithm uses clever indexing heuristics to reduce the runtime, an exact algorithm still needs to enumerate and track an exponential number of candidate subgraphs. We solve this issue by using random sampling. Finally, Borgwardt et al. [7] look at the problem of finding dynamic patterns in graphs, i.e., patters over a graph time series, where *persistence* in time is the key property that makes the pattern interesting. Dynamic graph patterns capture the time-series nature of the evolving graph, while in our streaming scenario, only the latest instance of the graph is of interest.

3 PRELIMINARIES AND PROBLEM DEFINITION

We now define the concepts and notations used throughout the work (the main notation is summarized in Table 1), and formally define the problem under study.



(a) An example toy graph G with a set L= {R, B} of vertex labels.

(b) All the 3-patterns that appear in the graph G on the left: for each pattern P_i, the equivalence class $C_{3,i}$ contains the 3-vertex subgraphs of G that are isomorphic to pattern P_i, for i = 1, 2, 3.

Fig. 1. Toy example graph and patterns.

Let $G = (V, E, L, Q, \ell, q)$ be a labeled graph, where L (resp. Q) denotes the set of possible vertex (resp. edge) labels and ℓ (resp. q) is the function that associates a label to a vertex (resp. edge). Unlabeled graphs can be represented as graphs with labels set of size one. As these sets and functions do not change, we drop them from the notation, and use the simpler $G = (V, E)$. All graphs we deal with are labeled (unless otherwise specified), simple, and undirected, so we avoid repeating these properties in the following. An example graph is shown in Figure 1(a).

We denote the *neighborhood* of a vertex $u \in V$ as

$$\mathcal{N}_u(G) = \{v \ : \ (u, v) \in E\}.$$

Similarly, we define the h-hop neighborhood $\mathcal{N}_{u,h}(G)$ of u as the set of the vertices *at distance exactly h* from u, by following edges in E, i.e.,

$$\mathcal{N}_{u,h}(G) = \{v \ : \ d_G(u, v) = h\},$$

where $d_G(u, v)$ is the shortest path distance between u and v in G. It holds $\mathcal{N}_{u,0}(G) = \{u\}$, $\mathcal{N}_u(G) = \mathcal{N}_{u,1}(G)$, and $\mathcal{N}_{u,h}(G) \cap \mathcal{N}_{u,j}(G) = \varnothing$ for $h \neq j$.

A graph G is *connected* if and only if for all pairs of distinct vertices $u, v \in V$, there is a path from u to v, i.e., an ordered sequence of distinct vertices $(u, w_1, \ldots, w_z, v)$ such that $(u, w_1) \in E$, $(w_z, v) \in E$, and $(w_i, w_{i+1}) \in E$, $1 \leq i \leq z - 1$.

Let $S \subseteq V$ be a subset of vertices, and let $E(S) = \{(u, v) \in E \ : \ u, v \in S\}$. We say that $G_S = (S, E(S))$ is the subgraph of G *induced* by S.

Two graphs $G' = (V', E')$ and $G'' = (V'', E'')$ are *isomorphic* if there exists a bijection $I : V' \rightarrow V''$ such that $(u, v) \in E'$ if and only if $(I(u), I(v)) \in E''$ and the mapping I *preserves the vertex and edge labels*, i.e., $\ell(u) = \ell(I(u))$ and $q(u, v) = q(I(u), I(v))$, for all $u \in V'$ and for all $(u, v) \in E'$. We write $G' \simeq G''$ to denote that G' and G'' are isomorphic.

For $k > 0$, we define $\mathcal{C}_k$ to be the set of all connected induced subgraphs with k vertices in G, which we refer to as *k-subgraphs*. All subgraphs we consider are connected induced subgraphs, unless stated otherwise.

The isomorphism relation $\simeq$ partitions the set of subgraphs $\mathcal{C}_k$ into T_k equivalence classes,[1] denoted by $C_{k,1}, \ldots, C_{k,T_k}$, where the labeling is arbitrary. For each equivalence class $C_{k,i}$, we call the generic graph P_i that is isomorphic to all the members of $C_{k,i}$ the *k-pattern of* $C_{k,i}$. Figure 1(b) shows an example of these concepts for the graph in Figure 1(a).

[1]The value of T_k is uniquely determined by k, $|L|$, and $|Q|$.

Computing the k-pattern P_i given a k-subgraph $G_S \in C_{k,i}$ requires computing a *canonical form* of the subgraph, i.e., a representation such that two subgraphs are isomorphic iff they have the same canonical form. There are several possible canonical forms [26, 31, 43], but most of them use a string representation of the graph that is (lexicographically) minimal. Finding a canonical form is clearly as complex as graph isomorphism, and thus expensive to compute (although not NP-hard [3]).

For any k-pattern P_i, we define its *frequency* $f(P_i)$ as the fraction of k-vertex subgraphs of G that belong to $C_{k,i}$, i.e.,

$$f(P_i) = \frac{|C_{k,i}|}{|C_k|}. \tag{1}$$

Consider the toy graph G illustrated in Figure 1(a). As shown in Figure 1(b), there are three different 3-patterns that appear in G, named, as P_1, P_2, and P_3. G has a total of $|C_3| = 8$ different 3-vertex subgraphs, among which 6 of them are isomorphic to pattern P_1. Thus, the frequency of pattern P_1 is given by $f(P_1) = 6/8$.

The definition of frequency in Equation (1) does not satisfy *anti-monotonicity*: simpler patterns (e.g., wedges, for $k = 3$) may have lower frequency than more complex patterns that contain them (e.g., triangles) since we are considering *induced* subgraphs. As an easy example, consider a complete graph on three vertices. For $k = 3$, the triangle pattern will have a frequency of 1, while the wedge pattern will have a frequency of 0.

We now define the problem of *mining frequent k-patterns*.

Problem 3.1. *Given a graph $G = (V, E)$, an integer k, and a frequency threshold τ, find the collection $\mathcal{F}_k(\tau)$ of the k-patterns that have frequency at least τ, i.e.,*

$$\mathcal{F}_k(\tau) = \{P_i \ : \ 1 \leq i \leq T_k, f(P_i) \geq \tau\}.$$

The set $\mathcal{F}_k(\tau)$ is the collection of the *frequent k-patterns w.r.t. τ*. In practice, the chosen value of τ is application dependent and, given the *exploratory* nature of the task, the choice is often somewhat arbitrary (e.g., choosing $\tau = 0.30$ instead of $\tau = 0.295$ is a relatively arbitrary choice).

Computing $\mathcal{F}_k(\tau)$ *exactly* is hard on static graphs as it first requires to enumerate all subgraphs of size k, which takes $O(|V|^k)$ time, followed by the isomorphism evaluation for each of the enumerated subgraphs, translating to at most $|V|^k$ isomorphism evaluations. Thus, we define the following concept of an ε-approximation to the collection of interest.

Definition 1. For any $\varepsilon \in (0, 1)$, an ε-approximation $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ to $\mathcal{F}_k(\tau)$ is a collection of pairs $(p, \tilde{f}(p))$ with p being a subgraph pattern and $\tilde{f}(p)$ being an estimation of $f(p)$, such that

(1) for each $P_i \in \mathcal{F}_k(\tau)$ there exists a pair $(P_i, \tilde{f}(P_i))$ in $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$; and
(2) there is no pair $(p, \tilde{f}(p)) \in \widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ such that $f(p) < \tau - \varepsilon$;
(3) For every $(p, \tilde{f}(p)) \in \widetilde{\mathcal{F}}_k(\tau, \varepsilon)$, it holds $|\tilde{f}(p) - f(p)| \leq \varepsilon/2$.

Our goal is to compute ε-approximations of this collection on *evolving graphs*. Computing approximations of frequent patterns is reasonable when they come with quality guarantees, as it is the case here. Additionally, the aforementioned relative arbitrarily in the choice of τ makes the approximation even more useful, as it gives information about patterns whose real frequency is "around" τ.

Evolving graphs. For any time $t \geq 0$, we let $G^t = (V^t, E^t)$ be the graph considered at time t, where V^t represents the set of vertices and E^t represents the set of edges. At time $t = 0$, it holds $V^0 = E^0 = \varnothing$. The evolution of the graph is represented as a stream of update tuples: for any time $t > 0$, there is an update tuple $z^t = \langle \star, e = (u, v), \psi, \ell(u), \ell(v) \rangle$, where:

— $\star \in \{+, -\}$ represents an update operation, addition ($\star = +$) or deletion ($\star = -$);
— $e = (u, v)$ is an edge (u and v are two nodes);
— $\psi \in Q$ is an edge label; and
— $\ell(u), \ell(v) \in L$ are vertices labels.

The graph $G^t = (V^t, E^t)$ is obtained from G^{t-1} using z^t. The edge set is updated as

$$E^t = \begin{cases} E^{t-1} \cup \{e\} \text{ and } q(e) = \psi & \text{if } \star = + \\ E^{t-1} \setminus \{e\} & \text{if } \star = -. \end{cases}$$

We assume that each operation has an effect, i.e., $E^{t-1} \neq E^t$, for any $t \geq 1$. If the vertices u and/or v are not present in V^{t-1}, they are added to V^t, with the labels specified in the last components of the update tuple z^t. If a vertex $v \in V^{t-1}$ becomes isolated due to an edge deletion at time t, that vertex is removed from V^t.

We also assume that vertex/edge labels do not change as long as that vertex/edge is not removed (i.e., if $u \in V^{t-1}$, any update tuple adding an edge incident to u must have the same vertex label $\ell(u)$ for u). One can imagine a model involving single vertices being added or deleted, or having their label modified. Such operations can be defined as sequences of edge operations, hence, we discuss only edge additions and deletions for simplicity of presentation. We also assume that the order of updates can be chosen by an adversary that has complete knowledge of our algorithms but not of the random bits they use.

For $u \in V^t$, we use $\mathcal{N}_u^t$ to denote $\mathcal{N}_u(G^t)$, and $\mathcal{N}_{u,h}^t$ to denote $\mathcal{N}_{u,h}(G^t)$. If $u \notin V^t$, then both notations denote the empty set.

Our model represents a *fully-dynamic stream of edges*, rather than the stream of graphs considered in other works [40]. For each time t, we denote with C_k^t the set of k-subgraphs in G^t, and with $\mathcal{F}_k^t(\tau)$ the collection of frequent k-patterns in G^t w.r.t. τ. Since at each time step this collection may change significantly, computing it exactly, in addition to being very expensive, has *little value*, due to its volatility. Thus, we focus on computing *ε-approximations* of this collection.

Given user-specified failure probability $\delta \in (0, 1)$ and error tolerance $\varepsilon \in (0, 1)$, for each time t TipTap computes, with probability at least $1 - \delta$ (over its executions), an ε-approximation to the collection $\mathcal{F}_k^t(\tau)$ by efficiently estimating $f(P_i)$, for all $i = 1, \ldots, T_k$, from a *uniform (random) sample* $\mathcal{S}$ of C_k^t.[2] All the samples we consider are *without replacement*, i.e., they may never contain multiple copies of the same element of C_k. Our goal is to maintain an *approximate* collection of frequent k-patterns at each time t without having to recompute it from scratch after each addition or deletion.

We assume to have complete access to the graph G^t at all times,[3] i.e., we do not restrict the available space, as it is typical in some streaming problems. For the problem we study, the size of the solution space is much larger than the graph itself, thus, having enough space to store the graph is necessary to obtain good approximations to the solution. This requirement is common to many other graph mining algorithms and commodity hardware is now able to satisfy it [4, 35].

4 TIPTAP: APPROXIMATION ALGORITHMS FOR k-PATTERNS

This section describes TipTap. We first explain how to compute an ε-approximation from a uniform sample of the collection C_k of k-subgraphs, and how the sample size M and the error tolerance δ impact ε. Our analysis of this relationship relies on VC-dimension, and the resulting sample size is completely independent from any property of the input graph. We then introduce a basic variant

[2]Given any set A and a sample size M, a uniform sample $\mathcal{S}$ of A of size M can be obtained by selecting a subset of A of size M with equal probability among all subsets of A of size M.
[3]TipTap requires access only to the current graph G^t, not the whole history that led to it.

Table 2. Variants of TipTap and Their Capabilities

	Capabilities		
Name	Insertion-Only Streams	Skip-optimization	Fully-Dynamic Streams
TipTap-b	✓	✗	✗
TipTap-i	✓	✓	✗
TipTap-d	✓	✓	✓

TipTap-b, which is useful to understand how the different components in play, from reservoir sampling [39] to neighborhood exploration, come together to obtain a correct algorithm. Once these important blocks are in place, we show how to greatly improve the efficiency in a variant called TipTap-i, which uses a *skip-optimized* version of reservoir sampling and a smart counting procedure to avoid materializing many subgraphs. Finally, we discuss the case of fully-dynamic streams by introducing TipTap-d, which uses RP [17] in place of reservoir sampling. Table 2 summarizes the variants of the algorithm.

4.1 Computing an ε-approximation from a Uniform Sample

In this section, we show how to obtain an ε-approximation to $\mathcal{F}_k(\tau)$ from a uniform sample $\mathcal{S} = \{H_1, \ldots, H_M\}$ of M k-subgraphs from $\mathcal{C}_k$.

Given $\mathcal{S}$, the proportion

$$\tilde{f}(P_i) = \frac{|\{H \in \mathcal{S} : H \simeq P_i\}|}{|\mathcal{S}|}$$

of k-subgraphs in $\mathcal{S}$ that are isomorphic to P_i, $i \leq 1 \leq T_k$, is an *unbiased* estimate of $f(P_i)$, i.e.,

$$\mathbb{E}[\tilde{f}(P_i)] = f(P_i),$$

where the expectation is taken over the set of all uniform samples of size M.

TipTap computes, on the basis of a uniform sample $\mathcal{S}$, an approximation quality $\varepsilon \in (0, 1)$. Using this quantity and the estimated frequencies $\tilde{f}(P_i)$ obtained from $\mathcal{S}$, for $1 \leq i \leq T_k$, it outputs the collection

$$\widetilde{\mathcal{F}}_k(\tau, \varepsilon) = \left\{ \left(P_i, \tilde{f}(P_i)\right) \ : \ \tilde{f}(P_i) \geq \tau - \frac{\varepsilon}{2} \right\} \tag{2}$$

of pairs $(P_i, \tilde{f}(P_i))$ such that $\tilde{f}(P_i) \geq \tau - \varepsilon/2$. The observant reader will notice that if ε is too large, then $\tau - \varepsilon/2$ could be non-positive, thus the set of all k-subgraphs would be an ε-approximation. This fact should not be surprising: if the sample is too small (low M), then it may be impossible to obtain a high-quality approximation (low ε). There is obviously a relation between M and ε, which indeed is the main topic of this section. Such vacuous approximations can be avoided completely by imposing constraints to the maximum *relative* (i.e., multiplicative) frequency error, rather than to the maximum *absolute* frequency error. Essentially all the results presented in this section can be extended to the relative-error case, but we do not present them here to avoid making the presentation excessively heavy. The interested reader can use the results by Li et al. [28] to obtain relative-error guarantees.

4.1.1 Sufficient Condition for ε-approximation. The following lemma states a sufficient condition for $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ from Equation (2) to be an ε-approximation to $\mathcal{F}_k(\tau)$.

Lemma 4.1. *Let $\mathcal{S}$ and ε as above and $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ as in (2). If*

$$|\tilde{f}(P_i) - f(P_i)| \leq \frac{\varepsilon}{2} \text{ for all } 1 \leq i \leq T_k,$$

then $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ is an ε-approximation to $\mathcal{F}_k(\tau)$.

Proof. We need to show that the three properties from Definition 1 hold. Property 3 from Definition 1 is the easiest to show, as it holds by hypothesis.

We now show Property 1 from Definition 1. Let $P_i \in \mathcal{F}_k(\tau)$ be any frequent pattern. Given the hypothesis and the fact that $f(P_i) \geq \tau$, it follows that

$$\tilde{f}(P_i) \geq f(P_i) - \frac{\varepsilon}{2} \geq \tau - \frac{\varepsilon}{2},$$

thus $(P_i, \tilde{f}(P_i)) \in \widetilde{\mathcal{F}}_k(\tau, \varepsilon)$.

To show Property 2 from Definition 1, assume by contradiction that there is a pair $(P_i, \tilde{f}(P_i)) \in \widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ s.t. $f(P_i) < \tau - \varepsilon$. It follows from the hypothesis that

$$\tilde{f}(P_i) \leq f(P_i) + \frac{\varepsilon}{2} < \tau - \frac{\varepsilon}{2}.$$

We reach a contradiction since we assumed that $(P_i, \tilde{f}(P_i)) \in \widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ but $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ only contains pairs with second component *at least* $\tau - \varepsilon/2$. Thus Property 2 from Definition 1 must hold. □

4.1.2 Determining ε. We now show how to compute, given a failure probability δ and a sample size M, a value ε such that Lemma 4.1 holds with probability at least $1 - \delta$ over the choice of $\mathcal{S}$ among all subgraph samples of size M. Our analysis uses the notion of *VC dimension* [38], a key concept from statistical learning theory [37], hence, we first introduce this notion and related results.

Range spaces and VC-dimension. A *range space* is a pair $(X, \mathcal{R})$ where X is a (finite or infinite) *ground set* and $\mathcal{R}$ is a (finite or infinite) family of subsets of X. The elements of X and $\mathcal{R}$ are called *points* and *ranges* respectively. For any $A \subseteq X$, $P_{\mathcal{R}}(A) = \{r \cap A \mid r \in \mathcal{R}\} \subseteq 2^A$ denotes the *projection* of $\mathcal{R}$ on A, where 2^A is the *powerset* of A, i.e., the collection of all the (proper and improper) subsets of A. If $P_{\mathcal{R}}(A)$ contains all subsets of A, i.e., if $P_{\mathcal{R}}(A) = 2^A$, then A is said to be *shattered* by $\mathcal{R}$.

The *VC dimension* of the range space $(X, \mathcal{R})$ is the cardinality of the largest shattered subset of X [38]. If X has arbitrarily large shattered subsets, then the VC-dimension of $(X, \mathcal{R})$ is equal to ∞.

As an example, let $X = \mathbb{R}$ and $\mathcal{R}$ be the set of all closed intervals on the real line, i.e.,

$$\mathcal{R} = \{[a, b] \ : \ a \leq b \in \mathbb{R}\}.$$

It is easy (and left as exercise to the reader) to shatter any set of two points in $\mathbb{R}$. On the other hand, no set $A = \{x_1, x_2, x_3\}$ can be shattered. Indeed, let w.l.o.g., $x_1 < x_2 < x_3$, then there is no closed interval in $\mathbb{R}$ that contains x_1 and x_3 but does not contain x_2. Thus $\{x_1, x_3\} \notin P_{\mathcal{R}}(A)$, which implies that A cannot be shattered. So the VC-dimension of $(X, \mathcal{R})$ is 2.

We are interested in VC-dimension because it will allow us to obtain samples of X from which to estimate the relative sizes of the ranges in $\mathcal{R}$, as formalized in the following definition.

Definition 2 (η-sample). Let $(X, \mathcal{R})$ be a range space and let A be a finite subset of X. For any $0 < \eta < 1$, a subset $B \subset A$ is an η-sample for A if it holds

$$\left| \frac{|A \cap r|}{|A|} - \frac{|B \cap r|}{|B|} \right| \leq \eta \text{ for any } r \in \mathcal{R}.$$

The following result connects VC-dimension and sampling.

Theorem 4.2 (20, Theorem 2.124[4]). *Let $(X, \mathcal{R})$ be a range space with VC-dimension at most d and let A be a finite subset of X. Given a sample size m and a failure probability $\delta \in (0, 1)$, there exists a*

[4]We present an equivalent form of the theorem presented by Har-Peled and Sharir [20].

constant $c > 0$ such that, for

$$\eta = \sqrt{\frac{c(d + \ln \frac{1}{\delta})}{m}},$$

a subset B of A of cardinality m, chosen uniformly at random, is an η-sample for A with probability at least $1 - \delta$.

The constant c in the previous result was shown experimentally to be at most 0.5 [30], so we use this value in our experiments.

Determining ε. We now use Theorem 4.2 to show how to determine an ε such that, with probability at least $1 - \delta$ over all the samples of size M, it is possible to obtain an ε-approximation to $\mathcal{F}_k(\tau)$ from a uniform random sample $\mathcal{S}$ with a given size M.

Given the set C_k of k-subgraphs of G, we define the associated range space $(C_k, \mathcal{R})$ where the set $\mathcal{R}$ of ranges is defined from the equivalence classes of C_k, i.e., $\mathcal{R} = \{C_{k,i} : i = 1, \ldots, T_k\}$.

Lemma 4.3. *Given a sample size M and a failure probability $\delta \in (0, 1)$, let*

$$\varepsilon = \sqrt{\frac{4c(1 + \ln \frac{1}{\delta})}{M}} \tag{3}$$

for some constant $c > 0$, and let $\mathcal{S}$ be a uniform sample of C_k with cardinality M. Then, with probability at least $1 - \delta$ (over the choice of $\mathcal{S}$), $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ computed from $\mathcal{S}$ as in (2) is an ε-approximation to $\mathcal{F}_k(\tau)$.

Proof. We show that the VC-dimension of $(C_k, \mathcal{R})$ is 1. The thesis will then follow with an application of Theorem 4.2 with $\eta = \varepsilon/2$.

The set $\mathcal{R}$ of ranges contains T_k *disjoint* sets. Hence, the only subsets of C_k that can be shattered are subsets of cardinality 1. Indeed, if $A = \{G'_S, G''_S\} \subset C_k$ with $G'_S \neq G''_S$, one of the following two cases must happen:

(1) if G'_S is isomorphic to G''_S then there exists no range $C_{k,i} \in \mathcal{R}$ such that $A \cap C_{k,i} = \{G'_S\}$; and
(2) if G'_S is *not* isomorphic to G''_S then there exists no range $C_{k,i} \in \mathcal{R}$ such that $A \cap C_{k,i} = \{G'_S, G''_S\}$.

In either case, A cannot be shattered. Thus, the VC-dimension of $(C_k, \mathcal{R})$ is 1.

Assume now that $\mathcal{S}$ is an $\varepsilon/2$-sample for $(C_k, \mathcal{R})$, which, by Theorem 4.2, happens with probability at least $1 - \delta$. It holds

$$|f(P_i) - \tilde{f}(P_i)| = \left| \frac{|C_k \cap C_{k,i}|}{|C_k|} - \frac{|\mathcal{S} \cap C_{k,i}|}{|\mathcal{S}|} \right| \leq \frac{\varepsilon}{2}, \text{ for each } 1 \leq i \leq T_k.$$

Thus, Lemma 4.1 holds for $\mathcal{S}$, and $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ obtained from it is an ε-approximation to $\mathcal{F}_k(\tau)$. □

Once again, the constant c in the previous result was shown experimentally to be at most 0.5 [30], so we use this value in our experiments. The sample size from Lemma 4.3 does not in any way depend on the minimum frequency threshold τ or on properties of the graph. While this result may be surprising at a first read, it is reasonable, due to fact that we are essentially estimating a probability distribution: the pattern frequencies (and also their estimates) are non-negative and must sum to one. These constraints are quite strong but enable us to avoid any dependence on the graph. This lack of dependency is extremely beneficial as it allows us to study ever-increasing and constantly-changing dynamic graphs even if they have a large number of edge and/or vertex

labels and for high value of k, as the VC-dimension (and thus the sample size) is a constant with respect to these quantities.

4.2 Reservoir Sampling for Incremental Edge Streams

Now that we have discussed how to use the sample of subgraphs to compute an ε-approximation, let us turn our attention to how to create and *maintain* a uniform sample in a streaming environment. For now, we assume an *incremental* stream of edge updates, i.e., only *addition* operations are allowed. We deal with the fully-dynamic version of the problem in Section 4.5. Our goal is to maintain, at each time step t, a uniform sample $\mathcal{S}$ of fixed size M of k-subgraphs from G^t.

4.2.1 The Meta-stream. The idea at the core of our approach is to transform the stream of update tuples into a *meta-stream* of k-subgraphs, or more precisely, of *k-tuples of vertices*. Consider the update tuple z^t on the stream at time $t > 0$ and let (u, v) be the edge involved in this update tuple. Recall that the update tuple must prescribe the *insertion* of (u, v), as we are looking at incremental streams. The update tuple z^t "injects" on the meta-stream *every* k-tuple μ of k distinct vertices from V^t such that

(1) the k-tuple μ contains both u and v; and
(2) the subgraph induced by the vertices in the $k$tuple μ is connected in G^t and was not connected in G^{t-1}, where with this expression we include the possibility that u or v may not have been in V^{t-1}.

The order in which the k-tuples satisfying these conditions are injected on the meta-stream can be arbitrary or chosen by an adversary. A k-tuple $\mu = (u, v, w_1, \ldots, w_{k-2})$ of vertices is injected on the meta-stream at most once: if an update tuple z^t injects μ on the meta-stream at time t and another update tuple $z^{t'}$ for $t' > t$ involves, w.l.o.g., an edge (u, w_i) or (w_i, w_j), the tuple μ is not again injected on the meta-stream, despite the fact that $z^{t'}$ modifies the subgraph induced by μ, as the subgraph was already connected at time $t < t'$. Figure 2 shows an example of the meta-stream.

It should be clear that z^t may inject zero, one, or more k-tuple of vertices on the meta-stream. For example, if $k > 2$ and both u and v do not belong to V^{t-1}, then zero k-tuples are injected on the meta-stream at time t. This fact causes the "clocks" of the stream and the meta-stream to increase at different rates, in the sense that the number $N^t = |C_k^t|$ of k-tuples of vertices seen on the meta-stream from the start of the stream up to and including time t, i.e., the population size, may be larger, smaller, or equal to t. It is only guaranteed that $N^t = 0$ when $t < k$. In the following, when we talk about *time*, we will always refer to the time t in the original stream of update tuples, and we refer to N^t as the number of k-tuples of vertices seen on the meta-stream up to time t.

The fact below follows easily from the description of the meta-stream model.

Fact 4.4. *The set of the N^t k-tuples of vertices seen on the meta-stream up to time t is exactly the collection of k-tuples whose induced subgraphs in G^t form the set C_k^t.*

We want to be able to create a uniform sample of C_k^t for each time t, or, more precisely, to maintain, over time, a set $\mathcal{S}$ of $|\mathcal{S}| = M$ subgraphs that can be seen, at time t, as a uniform sample of C_k^t. At a high level, we achieve this goal by performing *reservoir sampling* [39] on the meta-stream of k-tuples. It is important to remark that the meta-stream is never actually materialized, rather TipTap *simulates* it in an efficient way, as described in the following sections.

4.2.2 Reservoir Sampling. Reservoir sampling is a classic randomized algorithm to create a uniform sample $\mathcal{S}$ of fixed size M from a stream of elements [39]. It works as follows. Let N be the number of elements that have been on the stream so far, including the one currently on the stream.

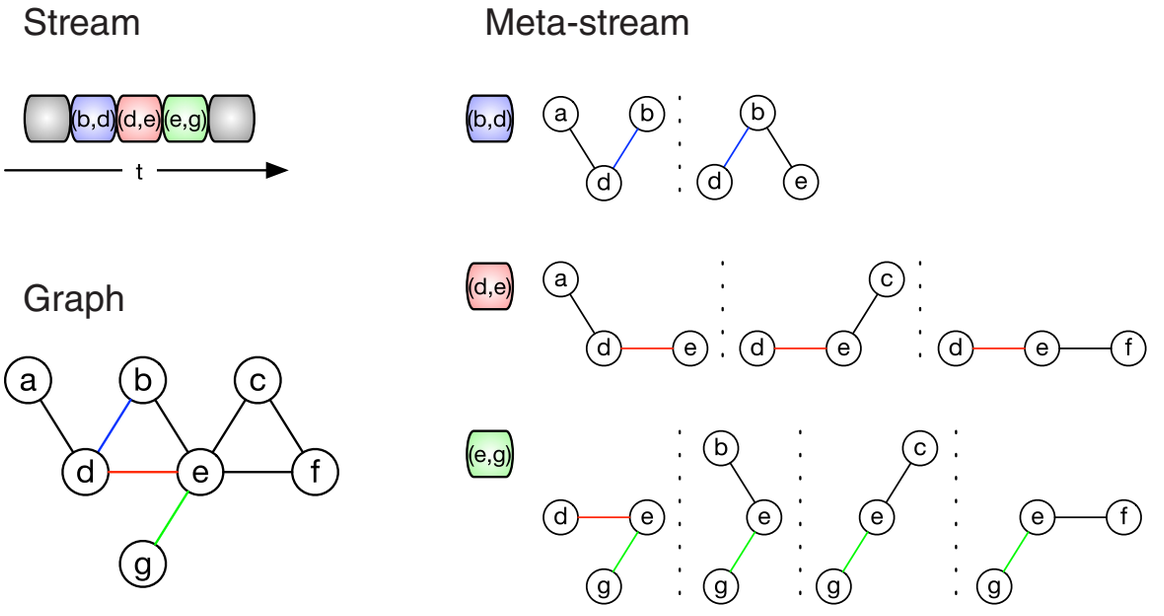


Fig. 2. An example of how the meta-stream of 3-tuples is generated from the stream of edges. For clarity, we show the induced subgraphs of the tuples. The order of the tuples injected on the meta-stream after the arrival of an edge is arbitrary or even adversarial. Two 3-tuples are injected when (b, d) arrives, three when (d, e) arrives, and four when (e, g) arrives. The tuple (b, d, e) is injected when (b, d) arrives and not again when (d, e) arrives because its induced subgraph was already connected.

- If $N \leq M$, deterministically add the element currently on the stream to the sample $\mathcal{S}$.
- Otherwise, flip a biased coin with tail probability M/N. If the outcome is head, do nothing. If the outcome is tail, choose uniformly at random an element already in the sample and replace it with the one currently on the stream.

Lemma 4.5 ([39, Sect. 2]). *For any $t > M$, let A be any subset of size $|A| = M$ of the elements on the stream between time 0 and time t (included). Then, at the end of time step t,*

$$\Pr(\mathcal{S} = A) = \frac{1}{\binom{t}{M}},$$

i.e., the set of elements in $\mathcal{S}$ at the end of time t is a subset of size M chosen uniformly *at random from all subsets of the same size.*

It is important to remark that the order of the elements in the stream up to time t is not relevant for Lemma 4.5.

4.3 A Basic Version of TipTap

We now discuss TipTap-b, a *preliminary, basic, inefficient* version of TipTap. Despite these downsides, TipTap-b contains most of the major ideas that will be incorporated in the final version of TipTap, so it is an important step for a clear understanding of our approach. The pseudocode of the algorithm is presented in Algorithm 1. Before the stream starts, the algorithm initializes (procedure init) some counters and data structures. Among these, there is the graph G (initially empty) and the array $\mathcal{S}$ of size M which will contain the sample. Using the input parameters M and δ, the algorithm also initializes ε: the sample is large enough to allow the computation of an ε-approximation of $\mathcal{F}_k^t(\tau)$, with probability at least $1 - \delta$ (see Lemma 4.3). Computing the approximation can be done at the end of each time step t as described in Section 4.1, and ε is returned

together with the approximation. The array $\mathcal{S}$ will contain *subgraphs*, although the elements on the meta-stream are *k-tuples*. The reasons for this discrepancy is explained in the next paragraph.

The core of the algorithm is in the procedure HANDLEINSERT which is called at each time t to handle the update tuple z^t involving the insertion of the edge (u, v) (for ease of notation the pseudocode only mentions (u, v), not the entire tuple). The procedure calls NEWLYCONNECTEDTUPLES$_k$ which returns the set $\mathcal{W}$ of k-tuples of vertices whose induced subgraphs were not connected at time $t-1$ but are connected at time t thanks to the edge (u, v) (line 8). These are the k-tuples appearing on the meta-stream due to the insertion of (u, v). TIPTAP-B then essentially performs reservoir sampling on the meta-stream by iterating over $\mathcal{W}$ (lines 9–12), with the difference, from our previous description, that the sample $\mathcal{S}$, instead of storing k-tuples, stores the *induced* subgraphs of the sampled tuples. The reason for this choice is that the edge (u, v), in addition to creating newly connected k-subgraphs in G^t (returned by NEWLYCONNECTEDTUPLES$_k$), may change the induced subgraphs of some k-tuples whose induced subgraphs were already connected in G^{t-1}. Since what we care about is the frequency of k-patterns, which depends on the number of induced connected k-subgraphs, it is more convenient to store k-subgraphs in $\mathcal{S}$. This design choice, although reasonable, requires the algorithm to perform some additional work in HANDLEINSERT: TIPTAP-B must update the subgraphs stored in $\mathcal{S}$ that were already connected in G^{t-1}, by adding the edge (u, v) to them (lines 14 and 15). These subgraphs are in the sample because they became connected at a time *earlier than* t, at which point their k-tuples were injected onto the meta-stream and they were sampled. Thus, these k-tuples of vertices are not seen on the meta-stream at time t, because they were on the meta-stream at an earlier time.

Analysis. As shown in Section 4.1, the sample frequencies of k-patterns in the sample are unbiased estimates of their exact frequencies. The following theorem states the even stronger probabilistic quality guarantees offered by TIPTAP-B.

THEOREM 4.6. *Fix a time t. With probability at least $1-\delta$ (over TIPTAP-B's runs), the collection $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ computed as in (2) from the TIPTAP-B's sample $\mathcal{S}$ at time t is an ε-approximation to $\mathcal{F}_k(\tau)$.*

PROOF. The thesis follows by combining Lemma 4.3 with Theorem 4.2, as TIPTAP-B keeps a uniform sample of the meta-stream of size M as in Equation (3). □

Efficient implementation of reservoir sample operations. Our algorithms access the reservoir sample in two ways, both of which must be implemented efficiently:

(1) random access, to replace a subgraph in the sample with a newly connected one; and
(2) access by vertex id, to identify subgraphs in the sample that have both u and v as vertices, where u and v are the endpoints of the edge being modified. These subgraphs need to be modified (by adding or removing the edge (u, v)), so we need to be able to find them quickly.

In order to support both operations in constant time, we use an array for the basic random access, supplemented by a *hash-based index* for the access by vertex id.

The basic array is straightforward, as the sample size M is fixed, and the size of its elements is $O(k^2)$ to store both vertices and edges.[5] On top of this basic array, we maintain an index $\mathcal{I}$ that maps every vertex $v \in V$ to the indices of the subgraphs in the array that have v as vertex. When an edge (u, v) is modified at time t, we can retrieve the set of subgraphs affected by this update by computing $\mathcal{I}(u) \cap \mathcal{I}(v)$. Each of the subgraphs in this intersection must be updated by having (u, v) added or deleted, their pattern needs to be recomputed, and the corresponding counters must

[5]Technically, we only need to store the k vertices, as the graph G is already in memory. However, this would require re-materializing each induced subgraph upon modification to recompute its pattern, which takes $O(k^2)$ time.

ALGORITHM 1: TipTap-b

procedure Init(M, δ, κ)
  $k \leftarrow \kappa$
  $\varepsilon \leftarrow \sqrt{\frac{4c(1+\ln \frac{1}{\delta})}{M}}$
  $\mathcal{S} \leftarrow$ empty array of size M
  $G \leftarrow$ empty graph
  $N \leftarrow 0$
procedure handleInsert($t, (u, v)$)
  $\mathcal{W} \leftarrow$ newlyConnectedTuples$_k(G, (u, v))$
  for each k-tuple $b \in \mathcal{W}$ **do**
    $N \leftarrow N + 1$
    if uniform(0,1) $< \frac{M}{N}$ **then**
      updateSample($\mathcal{S}, b$)
  Insert (u, v) into G
  for each subgraph $H = (V_H, E_H) \in \mathcal{S} \setminus \mathcal{W}$ s.t. $\{u, v\} \subset V_H$ **do**
    $E_H \leftarrow E_H \cup \{(u, v)\}$
procedure newlyConnectedTuples$_k(G, (u, v))$
  $\mathcal{W} \leftarrow \varnothing$
  for each $h \in [0, k-2]$ **do**
    $j \leftarrow k - 2 - h$
    $Z_{u,h} \leftarrow \bigcup_{i=0}^{h} \mathcal{N}_{u,i}(G)$
    $Z_{v,j} \leftarrow \bigcup_{i=0}^{j} \mathcal{N}_{v,i}(G)$
    $X \leftarrow \left\{S \subseteq Z_{u,h} \ : \ |S| = h + 1, u \in S, G_S \text{ is connected}\right\}$
    $Y \leftarrow \left\{S \subseteq Z_{v,j} \ : \ |S| = j + 1, v \in S, G_S \text{ is connected}\right\}$
    $\mathcal{W} \leftarrow \mathcal{W} \cup \{A \cup B \ : \ (A, B) \in X \times Y, |A \cup B| = k, G_{A \cup B} \text{ is } \textit{not} \text{ connected}\}$
  return $\mathcal{W}$

also be updated. This index update takes $O(M)$ time. The index also needs to be updated when a subgraph is replaced in $\mathcal{S}$. In this case, the index update takes $O(k)$ time.

Discussion. TipTap-b relies on the procedure newlyConnectedKTuples to simulate the meta-stream of newly-connected k-tuples of vertices. This procedure requires to materialize every single k-subgraph that can potentially become connected because of the insertion of (u, v). Thus TipTap-b would incur almost the same cost as an exact algorithm (modulo the computation of the canonical form of each subgraph to extract its pattern P_i, which is not strictly needed for the maintenance of the sample but it is needed to compute the approximation of the collection of frequent k-patterns). Our goal, which we achieve in the next section, is to materialize as few k-subgraphs as possible.

4.4 TipTap-i: A Refined Algorithm for Incremental Edge Streams

We now discuss how to overcome the issues affecting TipTap-b in our refined algorithm for incremental edge streams, which we call TipTap-i. To do so, we first need to introduce an optimized version of the reservoir sampling scheme, which was originally introduced by Vitter [39]. This improved version requires us to be able to compute, at each time t, the *number* of newly-connected k-tuples of vertices that are injected in the meta-stream when the tuple z^t involving the insertion of the edge (u, v) appears on the stream. We also need an efficient procedure to sample one (or

more) such k-tuples uniformly at random. Computing the number of newly-connected k-tuples, and sampling from their set can be done much more efficiently than the exhaustive enumeration done by TipTap-b.

4.4.1 Reservoir Sampling with Skip Optimization. Vitter's idea to speed up reservoir sampling is to model the distribution of the random number of elements on the stream that will be skipped between two modifications of the reservoir sample [39]. In concrete: suppose that at some time $t > M$, we insert the element currently on the stream into the sample (removing another element chosen uniformly at random), and let $t + x$ be the *next time*, we insert the element currently on the stream into the sample; The quantity x is a random variable, whose distribution depends on both the time t and the sample size M. Vitter [39] gives formulas for sampling from this distribution, which we denote with $\mathcal{X}(t, M)$. He then uses this idea to propose the following modification to reservoir sampling:

- At time $t \leq M$, deterministically insert the element currently on the stream into $\mathcal{S}$. At the end of time $t = M$, sample $x \sim \mathcal{X}(M, M)$, and let $n = t + x$. Then, move to the next element.
- At time $t > M$, if $t < n$ *skip* the element, i.e., do nothing. At time $t = n$, deterministically insert the element currently on the stream into the sample by replacing an element in $\mathcal{S}$ chosen uniformly at random. Then, sample $x \sim \mathcal{X}(t, M)$ and let $n = t + x$. Finally, move to the next element on the stream (i.e., move to time $t + 1$).

We refer to this sampling scheme as *skip-optimized reservoir sampling*, as it allows to *skip* elements on the stream.

ALGORITHM 2: TipTap-i

```
1:  procedure Init(M, δ, κ)
2:      k ← κ
3:      ε ← sqrt( 4c(1+ln 1/δ) / M )
4:      S ← empty array of size M
5:      G ← empty graph
6:      N ← 0
7:      n ← 1
8:  procedure handleInsert(t, (u, v))
9:      W ← ∅
10:     for each i ← 1, ..., ω_k do
11:         N ← N + classSize_{k,i}(G, (u, v))
12:         r ← 0
13:         while n ≤ N do
14:             r ← r + 1
15:             n ← nextIndexToSampleRS(n, M)
16:         Z ← sampleFromClass_{k,i}(r)
17:         for each k-tuple b ∈ Z do
18:             updateSample(S, b)
19:             W ← W ∪ {b}
20:     Insert (u, v) into G
21:     for each subgraph H = (V_H, E_H) ∈ S \ W s.t. {u, v} ⊆ V_H do
22:         E_H ← E_H ∪ {(u, v)}
```

4.4.2 Skip-optimized Reservoir Sampling on the Meta-stream. To use the skip-optimized reservoir sampling scheme on the meta-stream, we need to be able to count the number c^t of

newly-connected k-tuples that are injected into the meta-stream at time t, when the update tuple z^t involving the insertion of the edge (u, v) is on the stream. More precisely, let N^{t-1} be the number of k-tuples that have been on the meta-stream before z^t appears on the stream (and thus $c^t = N^t - N^{t-1}$). For ease of presentation, assume that $N^{t-1} > M$, thus also $N^t > M$ (the other cases are either trivial or can be easily derived from the following presentation). Let n be the index on the meta-stream of the next element that must be sampled, as computed by the skip-optimized reservoir sampling scheme, and let $e = n - N^{t-1}$. We need to be able to

(1) compute c^t to understand whether $n \leq N^t = N^{t-1} + c^t$, and therefore whether we need to insert in the reservoir the e-th k-tuple injected in the stream due to z^t; and
(2) identify such k-tuple efficiently.

The main idea behind our approach to achieve these two goals is to not consider the newly-connected k-tuples as a single "block" but rather as the union of ω_k disjoint *structural equivalence classes* $\{Y_1, \ldots, Y_{\omega_k}\}$, described in the following. The newly-connected k-tuples are injected on the stream by equivalence class, i.e., first all the k-tuples belonging to one equivalence class are injected, then all those belonging to another equivalence class, and so on. Computing the size of each equivalence class is straightforward, as we show in the following. Thanks to the property of reservoir sampling, the order according to which the equivalence classes are injected is irrelevant to the correctness of the algorithm and so is the order of the k-tuples in each equivalent class. Thus, if the next k-tuple to be inserted in the sample belongs to the equivalence class that we are currently examining, we can just sample such tuple uniformly at random from the equivalence class. Hence, we achieve both goals: we can efficiently count the k-tuples injected into the meta-stream and we can efficiently "identify" the k-tuple(s) to be inserted in the sample when needed.

TipTap-i's pseudocode is presented in Algorithm 2. The INIT procedure is essentially the same as in TipTap-b, with the exception of the variable n, which represents the index, on the meta-stream, of the next element that should be included in the sample. It is initialized to 1 and updated as needed with a call to NEXTINDEXTOSAMPLERS (line 15), which first samples x from the distribution $\mathcal{X}(n, M)$, as required by the skip-optimized reservoir sampling scheme (see Section 4.4.1) and then returns $n + x$. The HANDLEINSERT procedure of TipTap-i iterates over the ω_k equivalence classes for the specific value of k and computes the size of the current class Y_i using CLASSSIZE$_{k,i}$ (line 11). TipTap-i then determines how many k-tuples should be sampled from this class, by repeatedly checking whether n is less than the number N of k-tuples that would have been seen on the meta-stream after all those from Y_i have been injected (lines 12–15). The rk-tuples to be sampled (without replacement) from Y_i are obtained with the procedure SAMPLEFROMCLASS$_{k,i}$ (line 16), and their induced subgraphs are added, one by one, to the sample $\mathcal{S}$, replacing a subgraph already there (subgraphs inserted at later iterations of this loop may replace subgraphs inserted at earlier iterations). Finally, the edge is inserted into G and the subgraphs that were in $\mathcal{S}$ before this call to HANDLEINSERT and are still there, are updated with the edge (u, v) as needed.

Structural equivalence class. We are now ready to describe in depth the partitioning of the newly-connected k-tuples into the structural equivalence classes. Let us first describe how the classes are defined, then explain how a k-tuple is assigned to a class, and finally comment on how to compute the size of each class and sample from it.

First we describe how these equivalence classes are defined for a generic k (examples for $k = 3, 4$ are given below). The intuition behind the definition of these classes is the following: we put in the same class all the newly-connected k-tuples that can be found by exploring the neighborhoods of u and of v with truncated breadth-first searches with the same maximum depths ("same" across the k-tuples, but the two searches from u and v have potentially different maximum depths). Consider

all the *unlabeled*[6] k-patterns (e.g., the wedge and the triangle for $k = 3$) and consider the subset of them that have at least one cut of size 1, i.e., such that they have at least one edge whose removal would make the pattern disconnected (e.g., just the wedge for $k = 3$, while for $k = 4$ we have to consider the "line" with four nodes and three edges, the "3-pointed star" with edges only from one "central" vertex to the other three vertices, and the triangle with a "tail" of one edge). We can ignore the k-patterns that do not have such a cut because they can only be obtained by adding an edge to an already connected subgraph with k vertices (e.g., the triangle can only be obtained by adding an edge to a wedge), thus a k-tuple on the meta-stream would never be isomorphic to such patterns. Let q denote any of the patterns that have at least one cut of size 1. Each cut of size 1 splits the pattern q into two (internally connected) components, which we denote as q' and q''. Let χ' be the number of nodes in q' and similarly for χ'', and let a' be the vertex adjacent to the cut edge in q', and similarly for a'', and finally ψ' be the maximum (shortest-path) distance of a vertex in q' from a' (i.e., the height of the breadth-first search tree rooted at a'), and similarly for ψ''. For example, the wedge is split into two components, one with one vertex and one with two vertices connected by an edge, so $\chi' = 1$, $\chi'' = 2$, $\psi' = 0$ and $\psi'' = 1$. Consider now the *ordered* pairs of ordered pairs

$$((\chi', \psi'), (\chi'', \psi'')) \quad \text{and} \quad ((\chi'', \psi''), (\chi', \psi')).$$

There are cases where these two pairs may be the same (see an example below). The reason for considering both orderings of the elements of the pairs is that there are two ways of assigning the "names" q' and q'' to the two connected components, but there is one big exception: if either χ' or χ'' are equal to 1, (thus ψ' or ψ'' respectively must be equal to 0, and χ'' or χ' respectively must be equal to $k - 1$), we consider only the pair of the form $((1, 0), (k - 1, z))$, where z is the maximum distance from the non-isolated vertex incident to the cut edge. This exception covers the case when one of the two vertices incident to the newly-added edge is a new vertex, thus it does not make sense to consider the two ways of assigning the "names" q' and q''. The wedge we consider in our example is one of such cases, thus, only the ordered pair $((1, 0), (2, 1))$ is considered in this case.

Apart from the aforementioned exception, each cut of size 1 of q gives origin to two such pairs, some of which could be the same. For example, consider $k = 4$ and the line graph with four vertices and three edges. W.l.o.g., denote the vertices as 1, 2, 3, and 4, and assume that there are only the edges $(i, i + 1)$, for $i = 1, 2, 3$. The edge $(1, 2)$ is a cut of size 1, resulting in the pairs

$$((1, 0), (3, 2)) \quad \text{and} \quad ((3, 2), (1, 0))$$

but these same pairs result from the cut corresponding to the edge $(3, 4)$. Additionally, for this line graph, we also have to consider the cut corresponding to the edge $(2, 3)$, resulting in the *single* pair

$$((2, 1), (2, 1)).$$

Consider now the set of all these pairs across every k-subgraph that has at least one cut of size 1. There is one structural equivalence class for each pair in this set. For example, it should be clear from the description and the examples that in the case of $k = 3$ there is a single equivalence class Y_1 corresponding to the pair $((1, 0), (2, 1))$. For $k = 4$, there are five structural equivalence classes. We show in Figure 3 the patterns whose pairs give origin to the each class. For some of the classes, multiple 4-patterns correspond to the same class (see the caption of the figure).

The class to which a k-tuple is assigned to depends on the *disconnected and unlabeled graph* obtained by (i) considering the induced subgraph of the k-tuple without the newly-added edge (u, v) (but still with all the k nodes, even if either u or v is a newly-added vertex); and (ii) ignoring the

[6]This part is the only component of our work where labels are ignored, for the simple fact that they are not needed.

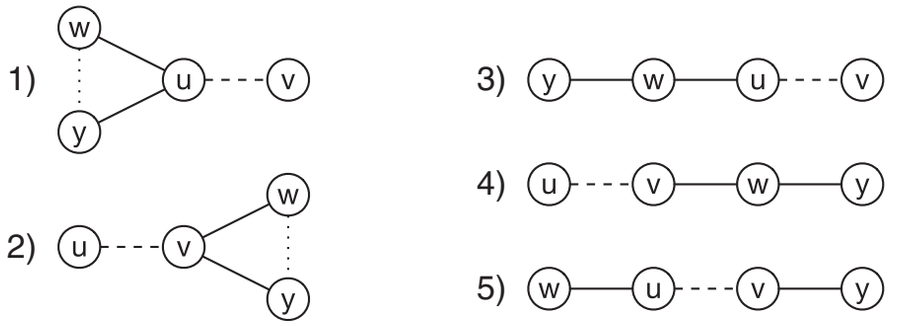


Fig. 3. The graphs corresponding to the five possible structural equivalence classes for newly-connected 4-subgraphs. Solid lines represent edges in G^{t-1}, dashed lines represent the incoming edge (u, v), and dotted lines represent edges that might or not be in the subgraph. Multiple graphs correspond to the same equivalence class whenever there is a dotted line, or when either u or v have no other neighbor than v or u respectively. Thus, class 5 is the only class to which a single graph corresponds.

labels on the nodes/edges. This disconnected and unlabeled graph identifies a structural equivalence class, and the same structural equivalence class may be identified by multiple such graphs, as shown in Figure 3. Constructing the equivalence classes for $k = 5$ is a good exercise, left to the interested reader.

The equivalence classes, like the k-tuples and the meta-stream, are *never materialized* during the execution of the algorithm, rather TipTap-i efficiently computes their *sizes* and *samples* from them. These procedures are specific to each value of k, but they are relatively straightforward to derive. Additionally it is often possible to combine the procedures to compute the size of each equivalence class and to sample k-tuples from each class into one, for additional efficiency: lines 11–19 of Algorithm 2 can be replaced with a call to a single procedure $\textsc{countAndSample}_{k,i}$. The following examples for $k = 3, 4$ make this idea concrete, and can serve as guidelines for higher values of k.

Case $k = 3$. The single equivalence class Y_1 is defined as

$$Y_1 = \left\{(u, v, w) \ : \ w \in (\mathcal{N}_u^{t-1} \cup \mathcal{N}_v^{t-1}) \setminus (\mathcal{N}_u^{t-1} \cap \mathcal{N}_v^{t-1})\right\} \ .$$

Efficiently computing (the size of) this class and sampling from it is straightforward. The $\textsc{countAndSample}_{3,1}$ procedure is shown in Algorithm 3.[7]

Case $k = 4$. The pseudocode for the procedures $\textsc{countAndSample}_{4,i}$ for $i = 1, 3, 5$ w.r.t. the numbering in Figure 3 is presented in Algorithm 4, from which the procedures for $i = 2, 4$ can be derived.

As can be seen in the case of $k = 4$ (and the same would hold for higher values of k), the step consisting of computing the sizes of and sampling from the equivalence classes still requires some exploration of the neighborhood, but is faster than enumerating all the subgraphs as done by TipTap-b.

Analysis. A result similar to Theorem 4.6 holds for TipTap-i, thanks to the fact that the skip-optimized variant of reservoir sampling guarantees that, at every time t, the sample $\mathcal{S}$ is a uniform sample of the meta-stream.

4.5 TipTap-d: Mining Frequent k-patterns from Fully-dynamic Edge Streams

We now describe TipTap-d, our algorithm for computing high-quality approximation of the collection of frequent k-subgraphs from *fully-dynamic* edge streams, i.e., when update tuples can

[7]The skipAndSample procedure is presented separately in Algorithm 3 because it is used for $k = 4$.

ALGORITHM 3: Count and sample newly-connected 3-tuples

```
1:  procedure countAndSample_{3,1}(G, u, v)
2:      Y ← (N_u(G) ∪ N_v(G)) \ (N_u(G) ∩ N_v(G))
3:      Z ← skipAndSample(Y)
4:      return {(u, v, w) : w ∈ Z}
5:  procedure skipAndSample(W)
6:      N ← N + |W|
7:      r ← 0
8:      while n ≤ N do
9:          r ← r + 1
10:         n ← nextIndexToSampleRS(n, M)
        return sampleWithoutReplacement(r, W)
```

ALGORITHM 4: Count and sample newly-connected 4-tuples

```
1:  procedure countAndSample_{4,1}(G, u, v)        ▷ countAndSample_{4,2} is identical, with u and v switched.
2:      Y ← N_u(G) \ N_v(G)
3:      Y' ← (Y × Y) \ {(w, w) w ∈ Y}
4:      Z ← skipAndSample(Y')
5:      return {(u, v, w, y) : (w, y) ∈ Z}
6:  procedure countAndSample_{4,3}(G, u, v)        ▷ countAndSample_{4,4} is identical, with u and v switched.
7:      R ← ∅
8:      for w ∈ N_u(G) \ N_v(G) do
9:          Y ← N_w(G) \ (N_u(G) ∪ N_v(G))
10:         Z ← skipAndSample(Y)
11:         R ← R ∪ {(u, v, w, y) : y ∈ Z}
12:     return R
13: procedure countAndSample_{4,5}(G, u, v)
14:     R ← ∅
15:     for w ∈ N_u(G) \ N_v(G) do
16:         Y ← (N_v(G) \ N_u(G)) \ N_w(G)
17:         Z ← skipAndSample(Y)
18:         R ← R ∪ {(u, v, w, y) : y ∈ Z}
19:     return R
```

represent either the insertion of a new edge or the deletion of an existing edge. When the update tuple on the edge stream involves the insertion of an edge, pairs of the form $(+, Z)$ are injected on the fully-dynamic meta-stream, where Z is a newly-connected k-tuple of vertices, in a fashion similar to the case for insertion-only streams. When the update tuple involves the deletion of an edge, the pairs injected on the meta-stream have the form $(-, Z)$, where Z is a *newly-disconnected* k-tuple of vertices: before the deletion of the edge, the induced subgraph of the k vertices in Z was connected, and becomes disconnected when the edge in the update tuple is removed.

Similarly to how TipTap-b and TipTap-i rely on reservoir sampling, TipTap-d relies on RP [17], a sampling scheme that extends reservoir sampling to fully-dynamic streams. We start by giving a short description of RP, and then show how TipTap-d uses it.

ALGORITHM 5: TipTap-d

procedure INIT(M, δ, κ)
  $k \leftarrow \kappa$
  $\mathcal{S} \leftarrow$ empty array of size M
  $G \leftarrow$ empty graph
  $c_\in \leftarrow 0$, $c_\notin \leftarrow 0$
  $N \leftarrow 0$
  $n_{\text{RS}} \leftarrow 0$, $n_{\text{RP}} \leftarrow 0$
procedure HandleInsert(t, (u, v))
  $\mathcal{W} \leftarrow \varnothing$
  for each $i \leftarrow 1, \ldots, \omega_k$ **do**
    $\rho \leftarrow N$
    $N \leftarrow N +$ classSize$_{k,i}(G, (u, v))$
    $r \leftarrow 0$
    while $c_\in + c_\notin > 0$ **and** $n_{\text{RP}} < N$ **do**
      $r \leftarrow r + 1$
      $c_\in \leftarrow c_\in - 1$
      $c_\notin \leftarrow c_\notin - (n_{\text{RP}} - \rho - 1)$
      $\rho \leftarrow n_{\text{RP}}$
      $n_{\text{RP}} \leftarrow$nextIndexToSampleRP($n_{\text{RP}}, c_\in, c_\notin$)
    if $c_\in + c_\notin = 0$ **then**
      while $n_{\text{RS}} \leq N$ **do**
        $r \leftarrow r + 1$
        $n_{\text{RS}} \leftarrow$nextIndexToSampleRS(n_{RS}, M)
    $Z \leftarrow$ sampleFromClass$_{k,i}(r)$
    for each k-tuple $b \in Z$ **do**
      updateSample($\mathcal{S}, b$)
      $\mathcal{W} \leftarrow \mathcal{W} \cup \{b\}$
  Insert (u, v) into G
  for each subgraph $H = (V_H, E_H) \in \mathcal{S} \setminus \mathcal{W}$ s.t. $\{u, v\} \subseteq V_H$ **do**
    $E_H \leftarrow E_H \cup \{(u, v)\}$
procedure handleDeletion(t, (u, v))
  $z \leftarrow 0$
  for each subgraph $H = (V_H, E_H) \in \mathcal{S}$ s.t. $(u, v) \in E_H$ **do**
    $E_H \leftarrow E_H \setminus \{(u, v)\}$
    if H is now disconnected **then**
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{H\}$
      $z \leftarrow z + 1$
  $c_\in \leftarrow c_\in + z$
  Delete (u, v) from G
  $u \leftarrow 0$
  for each $i \leftarrow 1, \ldots, \omega_k$ **do**
    $u \leftarrow u +$ classSize$_{k,i}(G, (u, v))$
  $c_\notin \leftarrow c_\notin + u - z$
  $N \leftarrow N - u$
  $n_{\text{RP}} \leftarrow$ nextIndexToSampleRP($N, c_\in, c_\notin$)

4.5.1 RP. We only describe, at a high level, the *skip-optimized* version of RP [17, Section 3.4].[8] This description may seem a little convoluted, but RP is an impressive algorithm that is very carefully tuned to ensure the uniformity of the sample at all times, so it is not suprising that some

[8]Our description is slightly different than the one in the original work by Gemulla et al. [17], to make it easier to adapt RP to the meta-stream of k-tuples, but the properties of the algorithm and its fundamental workings do not change.

complicated steps are necessary. Some of the steps will become more clear when we show how to apply them to our problem of interest in the next section.

The main intuition behind RP is that, in order to maintain the uniformity of the sample across time, deletions need to be *compensated* by insertions, but this "pairing" of deletions and insertions can be random. At any time t, the *population size* N^t is the *total* number of elements inserted and not deleted up to time t, i.e., it is the size of C_k^t. The population size is not monotonically increasing in fully-dynamic streams, in stark contrast with incremental streams. The fully-dynamic nature of the stream decouples the time t from the population size: there may be (and will be, as long as there is at least one deletion) distinct time steps t' and t'' such that $N^{t'} = N^{t''}$. This decoupling is extremely important: the time steps actually become less relevant in fully-dynamic streams, and what becomes (or better, remains) of key importance is the population size. The indices that are mentioned in this section refer to the population size, not the time steps.

At every time step t, there are $d^t \geq 0$ *uncompensated* deletions, which can be split into two classes: there are $c_\in^t$ uncompensated deletions each involving an element that was in the sample at the time the deletion appeared on the stream, and $c_\notin^t$ uncompensated deletions each involving an element that was *not* in the sample at the time the deletion appeared on the stream. Clearly $d^t = c_\in^t + c_\notin^t$, and at time $t = 0$, all these quantities are zero. At every time t, RP also keeps two indices n_{RP} and n_{RS}, whose role we describe in the following. The sample $\mathcal{S}$ has size *at most* M, but it may have smaller size at different time steps, due to deletions: at time t it has size equal to $\min\{N^t, M - d^t\}$.

When a *deletion* appears on the stream at time t, RP first checks whether the element being deleted belongs to the sample $\mathcal{S}$. If it does not, then RP increases $c_\notin$ by one. Otherwise, RP increases $c_\in$ by one and removes the element from the sample, whose size is now one less than at the previous time step. Finally, RP decreases the population size N^t by one, and updates n_{RP} by sampling from a random distribution that is a function of N^t, $c_\in^t$, and $c_\notin^t$. The details are not relevant for our discussion, and we refer the interested reader to the in-depth description by Gemulla et al. [17, Section 3.4], but we remark that n_{RP} is "connected" to the population size, not to the time steps, as it will be clear in the following paragraph.

The case for insertions is a little more complicated. RP essentially runs two exclusive regimes at the same time, choosing between them depending on whether there are uncompensated deletions (i.e., $d^t = 0$). When an *insertion* appears on the stream at time t, RP first increments the population size N^t by one and then checks d^t to select the regime.

If there are no uncompensated deletions (i.e., $d^t = 0$), then RP enters the "reservoir sampling regime" and mimics the skip-optimized version of reservoir sampling for the element at index $N^t + 1$. The algorithm uses n_{RS} as the index of the next element to sample, where the indexing happens with respect to the population size N^t, i.e., it is computed using the NEXTINDEXTOSAMPLERS function with N^t as the first argument and M as the second argument (see Algorithm 2, line 15). In other words, the next element to be sampled *in the reservoir sampling regime* is the one that would make the population size N^t equal to n_{RS}. Thus, if the current value of $N^t = n_{\text{RS}}$, then the current element on the stream is sampled and the sample $\mathcal{S}$ modified as needed: if $N^t \leq M$, the element is just added to the sample, otherwise the element replaces a randomly chosen element already in the sample $\mathcal{S}$. At this point, n_{RS} is updated as described above and the algorithm moves to the next element on the stream.

If instead there are uncompensated deletions (i.e, $d^t > 0$), RP enters the "compensating regime." The algorithm uses n_{RP} to denote the population size when the next element on the stream must be added to the sample in the compensating regime, similarly to what n_{RS} denotes for the reservoir sampling regime. In this regime, the algorithm first checks whether $N^t = n_{\text{RP}}$. If that is the case, then it decreases $c_\in$ by one, inserts in the sample the element in the update tuple currently on the

stream (there will always be an empty slot in the sample where to insert such element), and updates n_{RP} in the same way as discussed for the deletion case. If instead $N^t \neq n_{\text{RP}}$, RP just decreases $c_{\notin}$, and moves to the next element on the stream.

Lemma 4.7 ([17, Theorem 1]). *For any $t > 1$, the set of elements in $\mathcal{S}$ at the end of time t is a subset of the population chosen* uniformly *at random from all subsets of the same size.*

It is important to remark that the *size* of the sample $\mathcal{S}$ at every time t is a random variable. Gemulla et al. [17, Theorem 2] present an analysis of the statistical properties of this quantity.

4.5.2 RP for the Meta-stream. We now describe how TipTap-d adapts the RP scheme to the fully-dynamic meta-stream arising from the fully-dynamic edge stream. The pseudocode is presented in Algorithm 5. TipTap-d uses the same concept of equivalence classes used by TipTap-i. We already discussed why this approach makes sense for insertions in Section 4.4.2. In case of deletions, we give the intuition in the following paragraph.

Deletions. Let (u, v) be the edge involved in the deletion prescribed by the update tuple on the stream at time t. TipTap-d essentially runs the RP procedure for deletions on the meta-stream with little modifications. It first updates the subgraphs currently in the sample, removing (u, v) from those that contain it, and deleting from $\mathcal{S}$ those that become disconnected as a result (lines 33–37 of Algorithm 5). The counter $c_{\in}$ of the uncompensated deletions from the sample is updated as needed. To update the counter $c_{\notin}$ of the uncompensated deletions of k-tuples not in the sample, we need to compute the number of k-tuples that become disconnected because of the removal of (u, v). The collection of such k-tuples is the same as the collection of k-tuples that would become *connected* (again) if (u, v) was inserted at time $t + 1$, just after being removed. Thus, to know how many k-tuples of each class are injected (to be removed) into the meta-stream, TipTap-d removes the edge (u, v) from G (line 39) and then computes the number of k-tuples of each class that would become connected if (u, v) were added again (lines 41 and 42). Exactly the same code for the class size computation can therefore be used in both the addition and the deletion case. The sum of the class sizes includes the z k-tuples that were already removed from the sample, so the update to $c_{\notin}$ must take this quantity into consideration (line 43). After having updated the population size N (i.e., the number of connected k-subgraphs in the graph), TipTap-d samples the index of the next k-tuple that would be included in the sample (provided no additional deletions occur). Finally, it moves to the next update tuple on the stream.

Insertions. Let (u, v) be the edge involved in the addition prescribed by the update tuple on the stream at time t. As in the case of deletions, TipTap-d mimics RP on the meta-stream of k-tuples with minor changes, but is quite different than TipTap-i on insertion-only streams. TipTap-d iterates over the equivalence classes (for-loop starting on line 10 of Algorithm 5) and updates the population size N by adding to it the size of the class under consideration, i.e., the number of k-tuples of this class that are injected on the meta-stream because they become connected thanks to the insertion of (u, v). Before that, it saves in a variable ρ the size of the population *before* the k-tuples of this class are injected. The value of N after the update is therefore the size of the population *after* all k-tuples of the class under consideration have been injected. Some of these k-tuples though may need to be sampled, which TipTap-d does next. The algorithm behaves differently depending on whether there are uncompensated deletions (i.e., $c_{\in} + c_{\notin} > 0$) or not, exactly as RP does. This condition may change as a class is being processed if the class contains more than $c_{\in} + c_{\notin}$ k-tuples (for the values of these counters at the beginning of the for-loop iteration for this class). Thus, TipTap-d first considers the case of having uncompensated deletions (while-loop starting on line 14) and then the case of no uncompensated-deletion (if-test on line 20). In either case, TipTap-d simulates RP on the meta-stream by computing how many k-tuples of this class

should be sampled (this quantity is represented by the variable r in the pseudocode), using the appropriate indices n_{RP} and n_{RS} in each case (which, we recall, are compared to the population size, not to the time step), and appropriately decreasing the counters $c_{\in}$ and $c_{\notin}$ in the case there are still uncompensated deletions. It does not matter how the number of k-tuples from this class to be inserted in the sample is computed, because the correctness of RP is independent from the order of the elements on the (meta-)stream. Thus, once the final number r is available, TipTap-d can sample r k-tuples from the current class and update $\mathcal{S}$ as follows. If $\mathcal{S}$ has size less than M, then the induced subgraphs corresponding to $r' = \min\{r, M - |\mathcal{S}|\}$ of the sampled k-tuples are inserted into $\mathcal{S}$ without replacing any subgraph already in $\mathcal{S}$. Then, the subgraphs corresponding to the remaining $r - r'$ sampled k-tuples are inserted in $\mathcal{S}$ one by one, at each time replacing a subgraph already in $\mathcal{S}$ chosen uniformly at random, in the same way as in the reservoir sampling scheme. Finally, it moves to the next update tuple on the stream.

Finally, TipTap-d inserts the edge (u, v) in the graph G and in the subgraphs that contain the nodes u and v and were already in the sample $\mathcal{S}$ *before* the insertion of (u, v) and are still in $\mathcal{S}$ after the injection of the k-tuples (lines 28–30). The k-tuples of vertices corresponding to these subgraphs were already connected before the insertion of (u, v) but the corresponding induced subgraphs changed because of this insertion, and so they must be updated.

Computation of the ε-approximation. The sample $\mathcal{S}$ maintain by TipTap-d has *maximum* size M, for M given in input by the user, but throughout the execution of the algorithm, the number of connected k-subgraphs in the sample may be smaller than M, due to deletions (for TipTap-i, the sample would have size less than M only at the beginning of the stream, just because there would have been fewer than k-tuples on the meta-stream). Nevertheless, the sample is still a uniform random sample, over the set of samples of that size.[9] Different sample sizes imply different quality guarantees for the approximation to $\mathcal{F}_k^t(\tau)$ obtained from the sample as described in Section 4.1, thus at each time t (or whenever an approximation is requested), TipTap-d uses the *current* sample size to compute the approximation quality ε_t using Equation (3), and outputs it together with the approximation.

Analysis. Given the description of how ε_t is computed at each time t, a result similar to Theorem 4.6 also holds for TipTap-d, thanks to the properties of the RP sampling scheme.

5 EXPERIMENTS

We conduct an extensive empirical evaluation of TipTap, to measure its performacnee and to compare it with existing solutions in terms of accuracy of frequency estimation and of quality of the returned collection of patterns.

5.1 Experimental Setup

Datasets. We use three real-world graphs whose basic statistics are summarized in Table 3. These datasets are publicly available. Patent (PT) [19] contains citations among US patents from January 1963 to December 1999: the vertices represent the patents and there is an undirected edge between two patents if one cites the other. The label of a patent is the decade it was granted, i.e., $L = \{1960, 1970, 1980, 1990\}$, and there are no edge labels. The YouTube (YT) [12] graph has videos as vertices, and two videos are connected if they are related. The vertex label is a combination of a video's rating and length. For the first two datasets (PT and YT), the streams are generated by permuting the edges in a random order because we want to be able to simulate different scenarios

[9] The size of the sample at each meta-stream element is a random variable, but that is not problematic. For a complete analysis of the sample size, see [17, Section 3.3].

Table 3. Statistics of Graph Datasets

Dataset	Symbol	$\|V\|$	$\|E\|$	$\|L\|$
Patents	PT	3M	14M	4
Youtube	YT	4.6M	43M	11
DBLP	DB	4.89M	39.9M	10480

across different runs (and, at least for YT, there is no natural order of the edges). DBLP (DB)[10] contains citations among scientific articles from the DBLP databases. The vertices represent the articles. Each vertex has a label representing the venue in which the article was published, and two vertices are connected by an edge if one of the two corresponding papers cites the other. The fact that publication year is available in the dataset allows us to perform the experiments using a "natural" ordering of the edges and of vertex insertions.

Parameters. We run our algorithms TipTap-i and TipTap-d (see Table 2 for their capabilities) on the datasets, with $k = 3$ and $k = 4$. We used a sample size of 132,103, which leads to an approximation quality $\varepsilon = 0.01$, for $\delta = 0.1$.

Experimental environment. We conduct our experiments on a machine with 2 Intel Xeon Processors E5–2698 and 128 GiB of memory, running Linux. All the algorithms are implemented in Java, are sequential, and use a single processor. The source code is available online.[11]

Baselines. We use two baseline algorithms for comparison.

(1) Exact counting (EC) performs exhaustive exploration of the neighborhood of the updated edge up to $k - 2$ hops away and counts all new subgraph patterns.
(2) Edge reservoir (ER), an algorithm inspired by Trièst [13], maintains a reservoir sample of edges during the edge updates: for the case of fully-dynamic setting, RP [17] is used to ensure uniformity of the edge sample. Given the uniform sample of edges, the algorithm applies EC on the sample to produce exact counts of the patterns on the sample. Lastly, appropriate correcting factors are applied on these counts to estimate the original pattern frequencies [13]. For $k = 3$, the estimations are unbiased, while for $k = 4$ they are not, because computing the appropriate correcting factors to remove bias is an open problem. For $k = 4$ we use $(M/t)^{-r}$ as the correcting factor for patterns with r vertices at time t (M is the size of the reservoir sample), as this quantity is, in first approproximation, appropriate but still not formally correct. Additionally, even for $k = 3$, the estimates come with no probabilistic guarantees on their accuracy, a striking difference with those produced by TipTap. In the fully-dynamic setting, we implement ER only for $k = 3$, as for $k > 3$, the algorithm requires complex frequency estimators which, to the best of our knowledge, have not been studied yet. To ensure a fair comparison of TipTap-i with ER, we set the size of the ER from the maximum number of edges used in the subgraph reservoir, averaged over five runs. This approach was suggested in the evaluation section of Trièst [13]. The resulting sample sizes are approximately equal to 300k and 350k edges for $k = 3$ and 4, respectively, for both the datasets.

ER and TipTap assume different computational models. ER works in the traditional streaming model, where the algorithm has limited memory and can access an edge on the stream only once.

[10] https://www.aminer.org/citation.
[11] https://github.com/anisnasir/frequent-patterns.

Therefore, it needs to keep a small sample of edges to operate on. Conversely, TipTap assumes that the latest snapshot of the graph G^t can be kept in memory in its entirety. The sample of subgraphs is then used to speed up the estimation of pattern frequencies. While the traditional streaming model is theoretically interesting, from a practical point of view, there are very few graphs that do not fit in the memory of a large commodity server [4, 35]. In addition, given the exponential nature of the problem, the size of the solution space is usually much larger than the input. Therefore, while the comparison with ER is not completely fair, given the difference in computational models, we argue that this difference in approach is a strong point of TipTap, as the algorithm uses the availalbe computational resources to achieve its approximation guarantees. While other graphlet counting algorithms with computational models similar to the one we consider, have been recently published [34], and could in theory be adapted to our more difficult task, this adaptation is not trivial and there is no publicly available code.

The EC and ER algorithms are more competitive than any offline algorithm, e.g., GraMi [15], which requires processing the whole graph upon any edge update. EC takes less than 2×10^{-5} seconds to process an edge of the PT dataset on average, while one execution of GraMi on the same dataset takes around 30 seconds, which is several orders of magnitude larger, and would have to be multiplied by the number of edges.

Evaluation Metrics. We use two classes of metrics to evaluate the performance of the algorithms. Metrics in the first class measure the quality of the point-wise pattern frequency estimates of the algorithms, thus do not depend on the chosen minimum frequency threshold τ. The metrics in the second class evaluate the returned collection of patterns (depending on τ) and are frequently used to evaluate classification algorithms.

The metrics in the first class for the evaluation of the frequency estimates (τ-independent) are:

— *Mean Absolute Error (MAE):* measures the average absolute error in the frequency estimations of the subgraph patterns

$$\text{MAE} = \frac{1}{T_k} \sum_{i=1}^{T_k} \left| f(P_i) - \tilde{f}(P_i) \right|.$$

The MAE of TipTap is probabilistically guaranteed to be at most ε, because TipTap guarantees that none of the T_k summands is greater than ε (see Definition 1).

— *Kendall's τ* rank correlation coefficient.

— *Spearman's ρ* rank correlation coefficient.

For the evaluation of the returned collection of patterns (second class, τ-dependent) we report:

— *Precision:* measures the fraction of (actually) frequent subgraph patterns among the patterns returned by the algorithm

$$P = \frac{|\mathcal{F}_k^t(\tau) \cap \widetilde{\mathcal{F}}_k^t(\tau, \varepsilon)|}{|\widetilde{\mathcal{F}}_k^t(\tau, \varepsilon)|}.$$

— *Recall:* measures the fraction of (actually) frequent subgraph patterns returned by the algorithm over all the actually frequent subgraph patterns

$$R = \frac{|\mathcal{F}_k^t(\tau) \cap \widetilde{\mathcal{F}}_k^t(\tau, \varepsilon)|}{|\mathcal{F}_k^t(\tau)|}.$$

TipTap probabilistically guarantees $R = 1$, as it returns, with probability at least $1 - \delta$, an ε-approximation (see Definition 1).

— *F1 score:* the harmonic mean of precision and recall

$$\text{F1 score} = \frac{2}{\frac{1}{P} + \frac{1}{R}}.$$

— *AUPR:* the area under the precision-recall curve, also known as average precision. Let $P(R)$ be the value of precision obtained at a given recall level R, then

$$\text{AUPR} = \int_0^1 P(R)\ dR.$$

The evaluation measures in the second class depend on the threshold τ which defines which patterns are frequent, and which is a free parameter in our algorithm. In order to take into account all possible values of τ, we design an evaluation strategy that "integrates τ out of the equation." For each measure $\Phi(\mathcal{F}_k(\tau), \widetilde{\mathcal{F}}_k(\tau, \varepsilon))$, we want to compute $\int_0^{+\infty} \Phi(\mathcal{F}_k(\tau), \widetilde{\mathcal{F}}_k(\tau, \varepsilon))\ d\tau$, where $\Phi(\cdot, \cdot)$ is an evaluation metric between two rankings (such as AUPR) or two binary classifications (such as precision and recall), with arguments $\mathcal{F}_k(\tau)$ being the true collection of frequent patterns (ranked by their exact frequency) and $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ being the approximate collection of frequent patterns (ranked by their estimated frequency). It is reasonable to integrate out τ because this parameter essentially has *no impact* on the performances of our algorithms. Indeed, the accuracy and recall guarantees offered by TipTap do not depend on τ, i.e., they hold for any value of τ. The sample size does not depend on this parameter either. The runtime is also essentially independent from τ, because this parameter is used only when *querying* TipTap for the collection of approximate frequent patterns, which is computed very quickly using the (already computed) sample frequencies of the patterns. These queries can be submitted at any time step, possibly with different values of τ for different queries, giving yet another reason to integrate out this parameter in the evaluation. The only aspect that is impacted by the choice of τ is the *precision*, but the number of "close" false positives (patterns that are included in the output of TipTap but would not be frequent in the whole graph) depends more on the spectrum of the exact pattern frequencies around τ than on the workings of TipTap.

Both the ground truth $\mathcal{F}_k(\tau)$ and the estimate $\widetilde{\mathcal{F}}_k(\tau, \varepsilon)$ depend on the threshold τ we are integrating upon. The value of the function $\Phi(\cdot, \cdot)$ for different inputs only changes when τ crosses the frequency of one of the elements in either ranking (i.e., the function $\Phi(\cdot, \cdot)$ is piecewise-constant in τ). Therefore, we can compute the integral as a finite sum:

$$\sum_{\tau} \Phi(\mathcal{F}_k(\tau), \widetilde{\mathcal{F}}_k(\tau, \varepsilon))\Delta\tau,$$

where $\Delta\tau$ is the difference between one value of τ and the previous one in the integration.

One of the selling points of TipTap is that, with probability at least $1 - \delta$, the approximate collection of patterns that TipTap extracts from the sample has perfect recall. There exists applications that do not require perfect recall and would benefit from higher precisions. In these cases, some recall can be traded off to obtain a higher precision: one can use the point estimate of the frequency of a pattern and compare it directly to the desired frequency threshold τ, rather than to $\tau - \varepsilon/2$ in order to include the pattern in the output. Hence, in addition to the approximate collection of patterns that can be extracted from the sample as per Equation (2), i.e., any pattern P_i such that $\tilde{f}(P_i) \geq \tau - \frac{\varepsilon}{2}$, we also evaluate the quality of the collection of patterns that are deemed as frequent if their estimated frequency $\tilde{f}(P_i)$ is greater than τ. We distinguish the results of each of these variants by indicating in parenthesis the lower bound used for defining frequent patterns, i.e., $(\tau - \frac{\varepsilon}{2})$ for perfect recall case, and (τ) for recall-precision trade off case. See an example in Table 4.

Table 4. Evaluation Metrics for $k = 3$, Incremental Scenario
(All Numbers in Percentage)

Dataset	PT		YT	
Algorithm	ER	TipTap-I	ER	TipTap-I
MAE	0.164	**0.024**	0.028	**0.004**
Kendall τ	85.385	**98.792**	68.232	**89.834**
Spearman ρ	94.057	99.911	79.280	97.354
Precision $(\tau - \frac{\varepsilon}{2})$	95.165	**96.332**	40.561	**55.698**
Recall $(\tau - \frac{\varepsilon}{2})$	97.203	**100.000**	**100.000**	**100.000**
F1 score $(\tau - \frac{\varepsilon}{2})$	95.314	**97.914**	56.037	**67.358**
AUPR $(\tau - \frac{\varepsilon}{2})$	99.766	**99.971**	73.122	**99.960**
Precision (τ)	98.902	**99.738**	67.990	**98.638**
Recall (τ)	95.597	**99.557**	81.707	**97.040**
F1 score (τ)	96.637	**99.599**	73.218	**97.512**
AUPR (τ)	99.766	**99.971**	73.122	**99.960**

See the text for a description of the notation and a discussion of the results.
Best results on each dataset in **bold**.

We also evaluate the time efficiency of the algorithms by using appropriate metrics described in Section 5.4.

For ease of presentation, we report the values of the metrics at the end of the stream (if not mentioned otherwise). However, TipTap can return an ε-approximation to the collection of frequent patterns at any given point in the stream.

We report the results of experiments averaged over five runs. We do not report the variance or any quantile information about the distribution of the results over these runs because the results are extremely consistent, and presenting additional results would add negligible information while costing in clarity.

5.2 Incremental Case

We first report the result of our evaluation of TipTap-I, our proposed algorithm for incremental streams with skip-optimization (see Table 2). Starting from an empty graph, we add one edge per time step, for both the PT and YT datasets.

Table 4 shows the evaluation metrics for $k = 3$ on both datasets. For the PT dataset, both TipTap-I and ER behave similarly in terms of classification accuracy, as reflected by the precision and recall values obtained, with TipTap-I slightly outperforming ER in the aforementioned metrics. On the other hand, in terms of point-wise frequency estimation, TipTap-I significantly outperforms ER, with a MAE less than 85% of the one obtained by ER on the PT dataset. For the YT dataset, TipTap-I provides superior performance both in terms of classification accuracy and frequency estimation, significantly outperforming ER in all the evaluation metrics. We report in Figure 4 the pattern-by-pattern comparison of true frequencies versus estimates along with the confidence region w.r.t. the accuracy parameter ε in the YT dataset.

We now report the results for $k = 4$. Table 5 shows that in the PT dataset, TipTap-I provides superior performance both in terms of classification accuracy and frequency estimation, significantly outperforming ER in all the evaluation metrics. We also report in Figure 5 the pattern-by-pattern comparison of true frequencies versus estimates along with the confidence region in the PT dataset. ER is often unable to provide good estimates that fall within the confidence interval of the true frequencies, especially for the most frequent patterns, which are of the most interest. Conversely, TipTap-I provides excellent estimation accuracy for all the patterns. ER's inferior

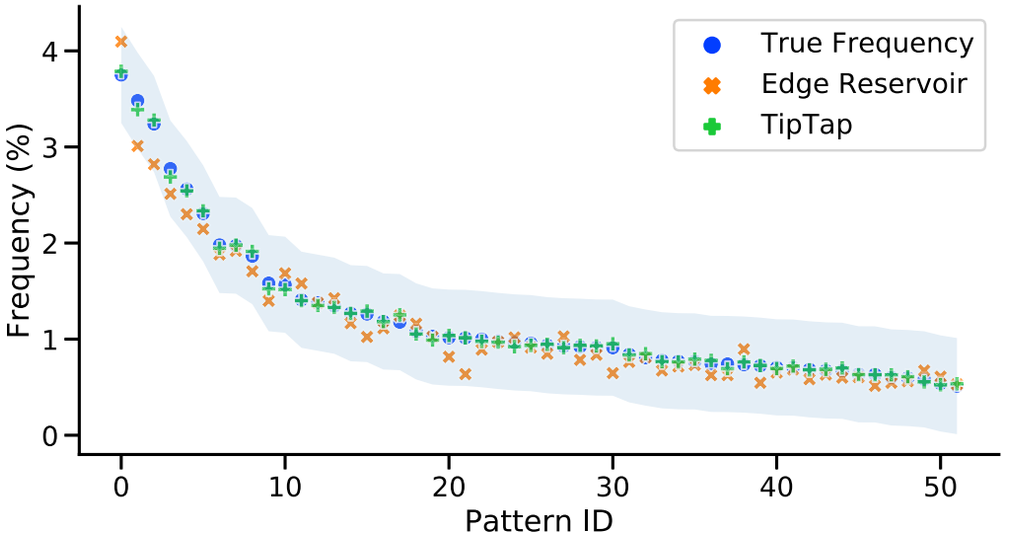


Fig. 4. True frequencies and estimates on the 3-patterns of YT with frequency larger than $\frac{\epsilon}{2}$ in the incremental scenario. The shaded area represents the confidence region $[f(P_i) - \frac{\epsilon}{2}, f(P_i) + \frac{\epsilon}{2}]$.

Table 5. Evaluation Metrics for $k = 4$, Incremental Scenario (All Numbers in Percentage)

Dataset	PT	
Algorithm	ER	TipTap-I
MAE	0.033	**0.006**
Kendall τ	80.240	**93.936**
Spearman ρ	90.932	**99.053**
Precision $(\tau - \frac{\epsilon}{2})$	71.230	**84.834**
Recall $(\tau - \frac{\epsilon}{2})$	98.793	**100.000**
F1 score $(\tau - \frac{\epsilon}{2})$	79.175	**89.399**
AUPR $(\tau - \frac{\epsilon}{2})$	91.117	**99.995**
Precision (τ)	84.947	**99.439**
Recall (τ)	94.353	**98.639**
F1 score (τ)	87.030	**98.888**
AUPR (τ)	91.117	**99.995**

See the text for a description of the notation and a discussion of the results. Best results on in **bold**.

estimation accuracy translates to lower precision, recall, and MAE values than of TipTap-I's as shown in Table 5. For the YT dataset, the exact computation could not scale to counting 4-subgraphs, hence, we are not able to report precision, recall, and MAE evaluation metrics on this dataset, as they required the availability of exact results. Thus, we only report in Figure 6 the pattern-by-pattern comparison of estimated frequencies by TipTap-I and ER, and observe the performance of ER using the confidence region computed w.r.t. the estimates obtained by TipTap-I as a proxy to the true confidence region.

Overall, we can conclude that TipTap-I provides extremely close estimation of frequencies compared to the exhaustive counting, with very low average relative error. These results then lead to similar conclusions in terms of the high precision and recall, as compared to the estimators. They corroborate our hypothesis that maintaining a reservoir of subgraphs results in higher-quality frequency estimations.

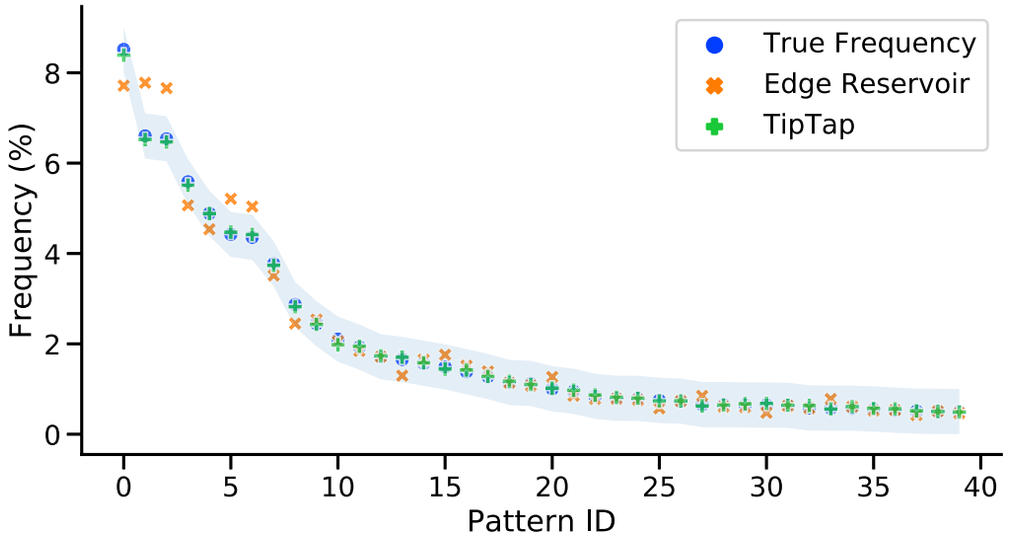


Fig. 5. True frequencies and estimates on the 4-patterns of PT with frequency larger than $\frac{\varepsilon}{2}$ in the incremental scenario. The shaded area represents the confidence region $[f(P_i) - \frac{\varepsilon}{2}, f(P_i) + \frac{\varepsilon}{2}]$.

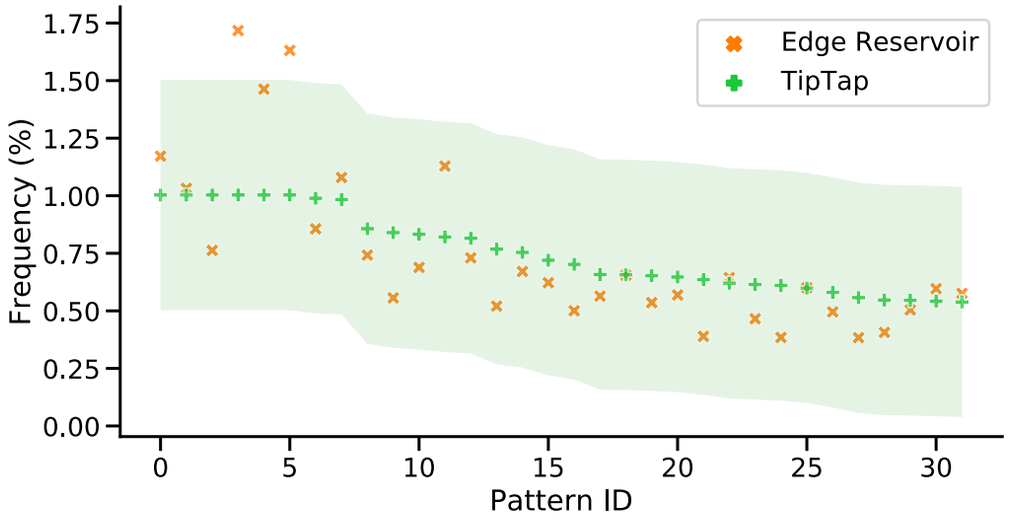


Fig. 6. Estimated frequencies on the 4-patterns of YT with estimated frequency larger than $\frac{\varepsilon}{2}$ in the incremental scenario. The shaded area represents the confidence region for TipTap-I. We are not able to compute the true frequencies given the size of the problem.

5.3 Fully-dynamic Case

We now report the results of the evaluation of the algorithms for fully-dynamic streams, i.e., TipTap-D (see Table 2) and and the fully-dynamic version of ER. To produce edge deletions, we use a sliding window model. This model is of practical interest as it allows to observe recent trends in the stream. For each dataset, we choose a sliding window large enough so that the number of edges (subgraphs) do not fit in the edge (subgraph) reservoir, otherwise ER is equivalent to EC: the sliding window has size 5M for the PT dataset, 10M for YT, and 1M for DB.

Table 6 reports the evaluation metrics on all three datasets for $k = 3$. Similarly to the incremental stream case, ER produces a higher relative error than TipTap-D, which directly translates to poor precision and recall. Figure 7 shows the pattern-by-pattern comparison of true frequencies versus estimates along with the confidence region w.r.t. the accuracy parameter ε on the DB dataset (results for other datasets are qualitatively similar). As expected, TipTap-D provides extremely close

Table 6. Evaluation Metrics for $k = 3$, Fully-dynamic Scenario (All Numbers in Percentage)

Dataset	PT		YT		DB	
Algorithm	ER	TipTap-d	ER	TipTap-d	ER	TipTap-d
MAE	0.115	**0.066**	0.014	**0.004**	0.004	**0.003**
Kendall τ	**98.338**	96.103	72.751	**88.461**	51.923	**57.725**
Spearman ρ	**99.559**	98.676	84.213	**96.324**	69.142	**73.710**
Precision $(\tau - \frac{\varepsilon}{2})$	**99.097**	99.075	62.785	**66.661**	2.701	**2.764**
Recall $(\tau - \frac{\varepsilon}{2})$	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**	**100.000**
F1 score $(\tau - \frac{\varepsilon}{2})$	**99.480**	99.461	72.929	**75.396**	3.765	**3.852**
AUPR $(\tau - \frac{\varepsilon}{2})$	**100.000**	**100.000**	99.406	**99.924**	97.421	**99.030**
Precision (τ)	99.476	**99.634**	95.813	**98.751**	94.092	**95.256**
Recall (τ)	99.493	**99.976**	93.639	**98.584**	91.818	**94.033**
F1 score (τ)	99.372	**99.763**	94.083	**98.542**	91.879	**94.011**
AUPR (τ)	**100.000**	**100.000**	99.406	**99.924**	97.421	**99.030**

See the text for a description of the notation and a discussion of the results. Best results on each dataset in **bold**.

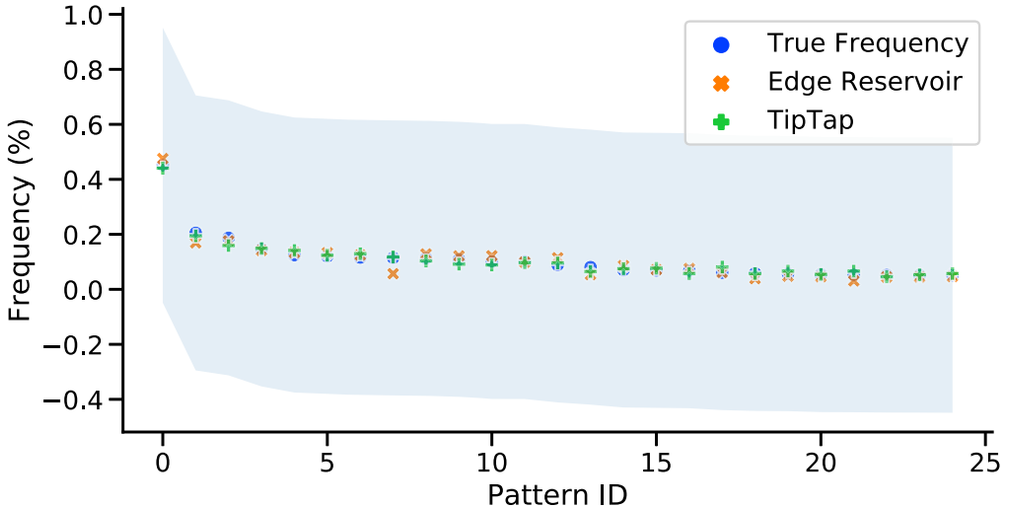


Fig. 7. True frequencies and estimates on the 3-patterns of DB with frequency larger than $\frac{\varepsilon}{20}$ in the fully-dynamic scenario. The shaded area represents the confidence region $[f(P_i) - \frac{\varepsilon}{2}, f(P_i) + \frac{\varepsilon}{2}]$.

estimation of frequencies compared to the exhaustive counting, with very low average relative error. We do not report comparison results in the fully-dynamic settings for $k = 4$ as it would require to design complex unbiased frequency estimators for ER, which have not been studied in the literature.

5.4 Processing Time

Further, we evaluate the algorithms in terms of the time to process edge insertions. In Figure 8, we show the cumulative execution time of insertion-only algorithms for the PT dataset, $k = 4$ (results are qualitatively similar for other settings). TipTap-i clearly provides significant gains in terms of the processing time compared to the EC. In particular, the total running time for EC to process the approximately 14 million edge insertions is greater than 3 days, compared to 3 hours using TipTap-i. This behavior is due to the fact that exhaustive counting requires exploration of all the subgraphs in the neighborhood of the newly-inserted edge.

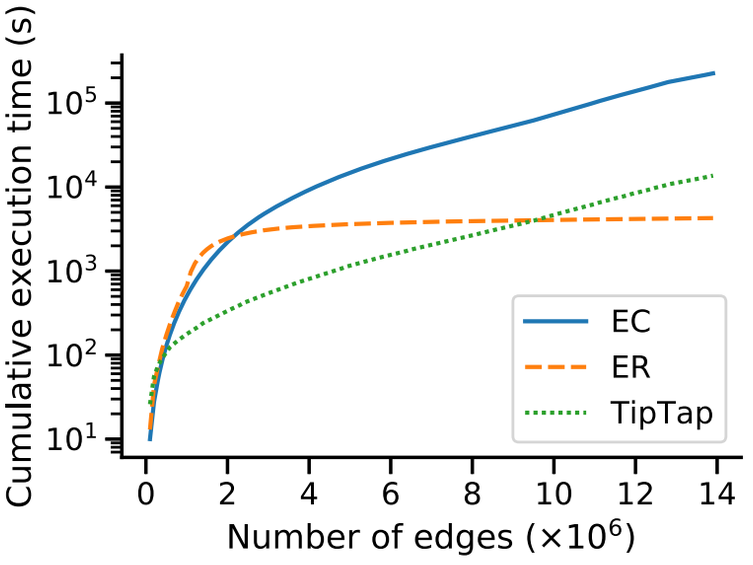
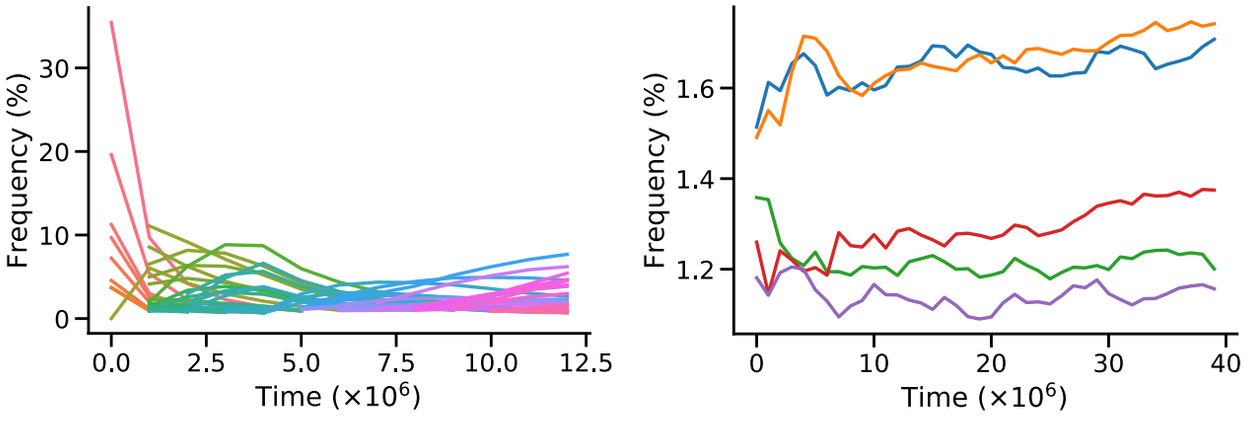


Fig. 8. Cumulative execution time on incremental streams (PT $k = 4$).



(a) PT dataset. **(b) YT dataset.**

Fig. 9. Temporal analysis of the most frequent patterns for $k = 4$ in the incremental setting. The horizontal axis represents the time stamps in the stream, which indicate the arrival of new edges. The vertical axis represents the frequency of the patterns at any given point in the stream. Different patterns are represented by different colors.

As expected, ER provides the best performance in terms of the update time once the sample fills up, as this algorithm only operates on the edges that reside in the reservoir. TipTap-i provides a desirable middle ground: it enables accurate and efficient estimation of the pattern frequencies and computes high-quality approximations of the collection of frequent patterns by smartly exploring the neighborhood around the newly-added edge, while storing more information thanks to the use of a subgraph reservoir sample.

5.5 Temporal Evolution of Patterns

Finally, we show a possible application of TipTap to study the temporal evolution of subgraph patterns. We report a visualization of the most frequent patterns for different snapshots of time in Figure 9. This figure shows the type of analysis that TipTap enables. For both PT and YT datasets, we report the frequency of the top-h most frequent patterns every 10^6 edges ($h = 30$ for PT, $h = 5$ for YT). The PT dataset presents an interesting dynamic with multiple burn-in periods: some

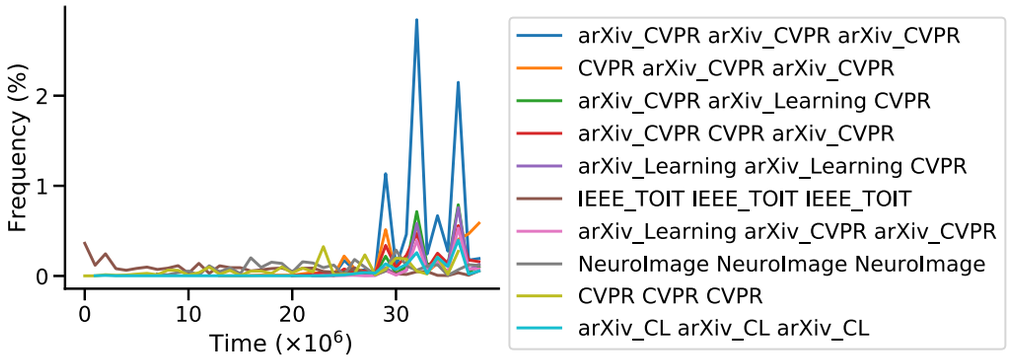


Fig. 10. Evolution in frequencies for the top-10 global patterns in DB.

patterns are more frequent at the beginning of the stream, while others take over in the middle, and a third set are more frequent at the end. Conversely, the most frequent patterns in YT are very stable across the stream, with minor variations in the top positions. This analysis, which enables the study of the patterns across time and allows comparing different datasets, is made possible by the online, anytime nature of TipTap.

Lastly, we execute TipTap-D with the DB dataset to observe how citation patterns among publication venues evolve over the years. To perform this experiment, we execute the algorithm in a sliding window model (window size of 1M edges). Moreover, we order the stream by the publication year and extract the three-node patterns ($k = 3$). Figure 10 reports the dynamic evolution top-10 global patterns. Firstly, we observe that all of the most frequent patterns are wedges. Secondly, we discover a prominence of CVPR, and a tendency in recent years to publish and cite arXiv papers, which is typical of fast-moving fields (compare the blue and yellow lines). Thirdly, we note an interesting downfall of TOIT, which was prominent in earlier years. This experiment highlights the potential of TipTap, and shows an immediate applicability to a real-world use case.

6 CONCLUSION

We present TipTap, a collection of sampling-based approximation algorithms for the collection of frequent k-vertex induced connected subgraph patterns in fully-dynamic labeled graphs represented as a stream of edge updates (insertions and deletions). TipTap extracts high-quality approximations of the collection of frequent k-vertex subgraph patterns, for a given frequency threshold, at any given time instance, with high probability. It maintains a uniform random sample of k-vertex subgraphs, using novel variants of reservoir sampling and RP. We derive bounds to the sample size sufficient to obtain an approximation of the desired quality with the desired confidence. Our bounds are based on VC-dimension, a key concept from statistical learning theory, which allows us to derive sample sizes that are completely independent from the size of the graph. This feature makes TipTap extremely attractive for analyzing ever-growing graphs. The results of the experimental evaluation show that TipTap generates high-quality results compared to natural baselines.

Promising directions for future work include exploring ways to limit, as much as possible, the exponential worst-case runtime dependency on the size of the graph, although that seems necessary due to the need of performing an exploration of the neighborhood of the added/removed edge. Additionally, it would be interesting to derive a systematic way to identify the structural equivalence classes used by TipTap for any k, rather than a customized derivation for each value of k.

ACKNOWLEDGMENTS

The authors are grateful to Aristides Gionis for the guidance and support during the preliminary phase of this work. We are also thankful to the anonymous reviewers for their valuable feedback.

Gianmarco De Francisci Morales acknowledges the support from Intesa Sanpaolo Innovation Center. The funder had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

REFERENCES

[1] Ehab Abdelhamid, Mustafa Canim, Mohammad Sadoghi, Bishwaranjan Bhattacharjee, Yuan-Chi Chang, and Panos Kalnis. 2017. Incremental frequent subgraph mining on large evolving graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2710–2723.

[2] Çigdem Aslay, Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, and Aristides Gionis. 2018. Mining frequent patterns in evolving graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*. 923–932. DOI: https://doi.org/10.1145/3269206.3271772

[3] László Babai. 2016. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. ACM, 684–697.

[4] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. 2012. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*. 33–42.

[5] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. 2011. Mining frequent closed graphs on evolving data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*. 591–599.

[6] Ilaria Bordino, Debora Donato, Aristides Gionis, and Stefano Leonardi. 2008. Mining large networks with subgraph counting. In *Proceedings of the International Conference on Data Mining (ICDM'08)*. IEEE, 737–742.

[7] Karsten M. Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. 2006. Pattern mining in frequent dynamic subgraphs. In *Proceedings of the International Conference on Data Mining (ICDM'06)*. 818–822.

[8] Peter Braun, Juan J. Cameron, Alfredo Cuzzocrea, Fan Jiang, and Carson K. Leung. 2014. Effectively and efficiently mining frequent patterns from dense graph streams on disk. *Procedia Computer Science* 35 (2014), 338–347. https://www.sciencedirect.com/science/article/pii/S1877050914010795.

[9] Björn Bringmann and Siegfried Nijssen. 2008. What is frequent in a single graph? In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 858–863.

[10] Chen Chen, Xifeng Yan, Feida Zhu, and Jiawei Han. 2007. gapprox: Mining frequent approximate patterns from a massive network. In *Proceedings of the International Conference on Data Mining (ICDM'07)*.

[11] Xiaowei Chen and John Lui. 2017. A unified framework to estimate global and local graphlet counts for streaming graphs. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'17)*. 131–138.

[12] Xu Cheng, Cameron Dale, and Jiangchuan Liu. 2008. Statistics and social network of Youtube videos. In *Proceedings of the 2008 16th Interntional Workshop on Quality of Service*. 229–238.

[13] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Transactions on Knowledge Discovery from Data* 11, 4 (2017), 43.

[14] Lorenzo De Stefani, Erisa Terolli, and Eli Upfal. 2017. Tiered sampling: An efficient method for approximate counting sparse motifs in massive graph streams. In *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data'17)*. IEEE, 776–786.

[15] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment* 7, 7 (2014), 517–528.

[16] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. 2006. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 595–606.

[17] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. 2008. Maintaining bounded-size sample synopses of evolving datasets. *The VLDB Journal* 17, 2 (2008), 173–201.

[18] Aristides Gionis and Charalampos E. Tsourakakis. 2015. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2313–2314.

[19] Bronwyn H. Hall, Adam B. Jaffe, and Manuel Trajtenberg. 2001. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. Technical Report. National Bureau of Economic Research.

[20] Sariel Har-Peled and Micha Sharir. 2011. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry* 45, 3 (2011), 462–496.

[21] Sebastian Hellmann, Claus Stadler, Jens Lehmann, and Sören Auer. 2009. DBpedia live extraction. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Springer, 1209–1223.

[22] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. 2000. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*.

[23] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Transactions on Knowledge Discovery from Data* 9, 3, Article 15 (2015), 1–21.

[24] Chuntao Jiang, Frans Coenen, and Michele Zito. 2013. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review* 28, 1 (2013), 75–105.

[25] Arijit Khan, Xifeng Yan, and Kun-Lung Wu. 2010. Towards proximity pattern mining in large graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 867–878.

[26] Michihiro Kuramochi and George Karypis. 2001. Frequent subgraph discovery. In *Proceedings of the International Conference on Data Mining (ICDM'01)*. 313–320.

[27] Michihiro Kuramochi and George Karypis. 2005. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery* 11, 3 (2005), 243–271.

[28] Yi Li, Philip M. Long, and Aravind Srinivasan. 2001. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences* 62, 3 (2001), 516–527. DOI: https://doi.org/10.1006/jcss.2000.1741

[29] Yongsub Lim and U. Kang. 2015. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 685–694.

[30] Maarten Löffler and Jeff M. Phillips. 2009. Shape fitting on point sets with probability distributions. In *Proceedings of the European Symposium on Algorithms*, Amos Fiat and Peter Sanders (Eds.). Lecture Notes in Computer Science, Vol. 5757. Springer, Berlin, 313–324. DOI: https://doi.org/10.1007/978-3-642-04128-0_29

[31] Brendan D. McKay. 1981. *Practical Graph Isomorphism*. Technical Report. Department of Computer Science, Vanderbilt University Tennessee, 45–87.

[32] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1870–1881.

[33] Abhik Ray, Larry Holder, and Sutanay Choudhury. 2014. Frequent subgraph discovery in large attributed streaming graphs. In *Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. 166–181.

[34] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2018. Estimation of graphlet counts in massive networks. *IEEE Transactions on Neural Networks and Learning Systems* 30, 1 (2018), 44–57.

[35] Antony Rowstron, Dushyanth Narayanan, Austin Donnelly, Greg O'Shea, and Andrew Douglas. 2012. Nobody ever got fired for using Hadoop on a cluster. In *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing (HotCDP'18)*. ACM, 2.

[36] Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. 2011. Triangle sparsifiers. *Journal of Graph Algorithms and Applications* 15, 6 (2011), 703–726.

[37] Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. Wiley. Retrieved from http://www.worldcat.org/isbn/0471030031.

[38] Vladimir N. Vapnik and Alexey J. Chervonenkis. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications* 16, 2 (1971), 264–280. DOI: https://doi.org/10.1137/1116025

[39] Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software* 11, 1 (1985), 37–57.

[40] Bianca Wackersreuther, Peter Wackersreuther, Annahita Oswald, Christian Böhm, and Karsten M. Borgwardt. 2010. Frequent subgraph discovery in dynamic networks. In *Proceedings of the 8th Workshop on Mining and Learning with Graphs*. ACM, 155–162.

[41] Pinghui Wang, John C. S. Lui, Don Towsley, and Junzhou Zhao. 2016. Minfer: A method of inferring motif statistics from sampled edges. In *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE'16)*.

[42] Fabrice Wendling, Karim Ansari-Asl, Fabrice Bartolomei, and Lotfi Senhadji. 2009. From EEG signals to brain connectivity: A model-based evaluation of interdependence measures. *Journal of Neuroscience Methods* 183, 1 (2009), 9–18.

[43] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining (ICDM'02)*. 721–724.

Received March 2020; revised October 2020; accepted December 2020

CASPITA: Mining Statistically Significant Paths in Time Series Data from an Unknown Network

Andrea Tonon
Department of Information Engineering
University of Padova
Padova, Italy
andrea.tonon@dei.unipd.it

Fabio Vandin
Department of Information Engineering
University of Padova
Padova, Italy
fabio.vandin@unipd.it

2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00075

Caspita: italian exclamation indicating surprise,
e.g., "Caspita! Such significant paths are really surprising."

***Abstract*—The mining of time series data has applications in several domains, and in many cases the data are generated by networks, with time series representing paths on such networks. In this work, we consider the scenario in which the dataset, i.e., a collection of time series, is generated by an *unknown* underlying network, and we study the problem of *mining statistically significant paths*, which are paths whose number of observed occurrences in the dataset is unexpected given the distribution defined by some features of the underlying network. A major challenge in such a problem is that the underlying network is unknown, and, thus, one cannot directly identify such paths. We then propose CASPITA, an algorithm to mine statistically significant paths in time series data generated by an unknown and underlying network that considers a generative null model based on meaningful characteristics of the observed dataset, while providing guarantees in terms of false discoveries. Our extensive evaluation on pseudo-artificial and real data shows that CASPITA is able to efficiently mine large sets of significant paths, while providing guarantees on the false positives.**

***Index Terms*—Pattern Mining, Statistically-Sound Pattern Mining, Time Series, Graph Mining**

I. INTRODUCTION

Time series data mining [1]–[3] is a fundamental data mining task that covers a wide range of real-life problems in various fields of research. Even if the common purpose is to extract meaningful knowledge from the data, many different problems and approaches have been proposed over the years, ranging from anomaly detection [4], [5] to motif discovery [6], from clustering [7] to classification [8]. However, in many real life scenarios, time series data are generated by networks, and thus represent paths constrained by the structures and the distributions that define such networks. Very often, one has access to a collection of time series but does not know the distribution on the network that generated them, or the structure of such network. As an example, consider a survey on the paths traveled by people with the underground service of a given city. In such a scenario, one has a dataset that represents a limited number of paths from a network, defined by the underground structure, but does not know the distribution defined by the entire population that uses such service.

In this work, we study the problem of mining statistically significant paths from an unknown network. We assume to have a time series dataset, defined as a collection of time series, and that such time series are paths generated from an unknown network. In such a scenario, we are interested in mining unexpected paths from the dataset. Standard techniques usually use the frequency or the number of occurrences as extraction criteria, with the aim of finding interesting paths, but, in many real applications, such metrics are not enough to find paths that provide useful knowledge. For example, paths that appear only few times in a dataset may be over represented if we consider the distribution of the network underlying the data, or vice-versa, paths that appear a lot of times may be under represented. Thus, techniques based on such metrics may led to several spurious discoveries. In addition, since we do not know the network underlying the data, we can not directly find over or under represented paths.

We then introduce CASPITA, an algorithm to find statistically significant paths over (or under) represented from time series data considering a generative null model based on meaningful characteristics of the observed dataset, while providing guarantees in terms of the false positives employing the Westfall-Young (WY) method. Our generative null model is based on the observed number of occurrences of paths of a given length, with the idea that such paths represent well-known substructures of the underlying network. In the simplest case, they are paths of length one, that are edges of the underlying network. Then, such null model is used to test the significance of paths of a given, higher, length, which are the paths of interest mined from the observed dataset. The intuition is to create a generative null model that is able to explain the number of occurrences of shorter paths, and to check whether such generative null model is able to also explain the observed number of occurrences of the paths of interest. Otherwise, such paths can be considered significant, in the sense that they appear more (or less) times than expected under such generative null model. Let us consider, as example, a network composed by the web-pages of a website, and suppose that we want to find sequences of web-pages visited more (or less) than expected with respect to the underlying distribution of the network, defined by the navigation of the users on the website. Given the application or the structure

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00075

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

of the network, there may be some well-know substructures, defined as short sequences of web-pages, that are traversed by the users that visit such website with a particular distribution. Again, let us note that in the simplest case, the substructures are paths of length one, that represent a direct link between two web-pages. Thus, in such a scenario, one may be interested in finding if such substructures also explain the number of observed occurrences of longer paths, or if such longer paths are significant due to some external factors causing their number of occurrences.

A. Our Contributions

In this work, we introduce the problem of mining *statistically significant paths* in time series data from an unknown network. In this regard, our contributions are:

- We introduce the problem of mining statistically significant paths in time series data from an unknown network, defining a generative null model based on meaningful characteristics of the observed dataset;
- We introduce CASPITA, an algorithm to mine statistically significant paths (over or under represented) from a time series dataset, while providing guarantees on the probability of reporting at least one false positive;
- We introduce an alternative interesting scenario in which CASPITA can be applied, which consists in mining paths that are significant with respect to a null model based on data from a different dataset;
- We perform an extensive suite of experiments that demonstrates that CASPITA is able to efficiently mine statistically significant paths in real datasets while providing guarantees on the false positives.

Let us note that throughout the paper, we only describe the scenario in which one is interested in mining statistically significant paths that occur more times than expected under the null hypothesis (*over represented paths*), for clarity of presentation. However, all the reasoning are still valid to mine paths that occur less times than expected (*under represented paths*). In particular, our open source implementation of CASPITA mines over or under represented paths, and results for both scenarios are shown in the experimental evaluation.

B. Related Works

We now discuss the relation of our work to prior art on significant pattern mining, anomaly detection in sequential data, and temporal anomaly detection in graphs, which are the areas most related to our work. Since the nature of our work, we only consider unsupervised approaches.

In significant pattern mining, the dataset is seen as a collection of samples from an unknown distribution, and one is interested in finding patterns significantly deviating from an assumed *null hypothesis*, i.e., *distribution*. Many variants and algorithms have been proposed for the problem. We point interested reader to the survey [9], and recent works that employ permutation testing [10]–[12]. Even if our work falls within the framework of significant pattern mining, such approaches are orthogonal to our work, which focuses on finding significant paths, i.e., patterns, from time series that are constrained by a network structure.

Many works have been proposed to detect anomalies in sequential data [4], [13], employing several definition of anomalies, and considering different types of patterns. For example, [5] defines a pattern as *surprising* if its frequency differs substantially from that expected by chance, given some previously seen data. Lemmerich et al. [14], instead, considers the mining of subgroups, defined by subsets of attributes, that exhibit exceptional transition behavior, i.e., induce different transition models compared to the ones of the attributes that describe the entire dataset. Although our approach adopts a definition of significant pattern based on how the number of its occurrences differs from the one expected under an appropriate model, similarly to other works, we consider the setting in which the data represent paths from a weighted and directed graph, which results in a different problem. In fact, this aspect makes our work closer to the task of detecting anomalies in temporal graph [15], [16], i.e., graphs that evolve over time. However, even if our work considers data generated by a network and aims to find paths whose number of occurrences is significant with respect to the network's distribution, we consider the scenario in which we do not know the network, and we have only access to a sample.

The only work that considers the problem of finding anomalous paths in time series data from an unknown network is [17]. In this work, the authors propose an algorithm, HYPA, to find anomalous length k paths using a null model based on length $k-1$ paths. In particular, they aim to find length k paths whose number of occurrences in a dataset is anomalous with respect to a null model based on the number of occurrences of length $k-1$ paths in the same dataset. Reducing the difficult problem of detecting anomalous length k paths to the easier problem of detecting anomalous edges in a k-th order De Bruijn graph, they describe a strategy based on the hypergeometric distribution to compute a score for each length k path, where the score describes the level of anomaly of such a path. Even if our approach is inspired by [17], our work differs from it in many key aspects. First of all, we aim to find length k paths whose number of occurrences in a dataset is significant with respect to a null model based on the number of occurrences of length h paths, with $h \in \{1, \dots, k-1\}$ provided in input by the user, and not only with $h = k-1$ as in [17]. In such a direction, it is not clear if HYPA can be modified to consider a more general length $h \in \{1, \dots, k-1\}$. Finally, while our approach employs the WY method to correct for multiple hypothesis testing providing guarantees in terms of false positives, [17] uses fixed thresholds to define interesting patterns, which does not provide any guarantee.

To the best of our knowledge, our work is the first approach that employs the statistical hypothesis testing framework to mine paths, i.e., patterns, from time series constrained by the structure and the distribution of an unknown network, while providing rigorous guarantees on the probability of reporting at least one false positive, i.e., the FWER.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

II. Preliminaries

We now provide the definitions and concepts used in the paper. First, in Section II-A, we describe the task of mining paths in time series data from a network. Then, in Section II-B, we define, similarly to [18], the concept of k-th order De Bruijn graph used in the paper to define our generative null model. Finally, in Section II-C, we describe concepts of hypothesis and multiple hypothesis testing for paths.

A. Mining Paths in Time Series Data from a Network

Let us define a *network* $N = (G, \omega)$ as a *directed graph* $G = (V, E)$ and a *weight function* $\omega : E \to [0, 1]$. $V = \{v_1, v_2, \ldots, v_{|V|}\}$ is the *vertices set*, where each $v \in V$ is called *vertex*, and $E = \{(u, v) : u, v \in V\}$ is the *edges set*, where each (u, v) is an *ordered* pair of vertices, called *edge*. An edge (u, v) is an *incoming edge* of the vertex v and an *outgoing edge* of the vertex u. Denoting with $(u, :)$ an outgoing edge of u, for each vertex $u \in V$, we have $\sum_{(u,:)\in E} \omega((u, :)) = 1$, that is, the weights of the edges from u represent a probability distribution. Fig. 1 (left) shows an example of network.

A *path*, or *walk*, $w = \{v_{i_0}, v_{i_1}, \ldots, v_{i_{|w|}}\}$ of length $|w|$ on the network N is an ordered sequence of $|w| + 1$ vertices such that $(v_{i_j}, v_{i_{j+1}}) \in E \ \forall j \in \{0, \ldots, |w| - 1\}$. Note that a vertex $v \in V$ is a path $w = \{v\}$ of length $|w| = 0$. A path $w = \{w_0, w_1, \ldots, w_{|w|}\}$ *occurs* in a path $q = \{q_0, q_1, \ldots, q_{|q|}\}$ starting from position $s \in \{0, \ldots, |q| - |w|\}$, denoted by $w \subset q^{(s)}$, if and only if $w_0 = q_s, w_1 = q_{s+1}, \ldots, w_{|w|} = q_{s+|w|}$. We say that the path w is a *sub-path* of the path q. The number of *occurrences* $Occ_q(w)$ of w in q is the number of times that w occurs in q, that is, $Occ_q(w) = |\{s \in \{0, \ldots, |q| - |w|\} : w \subset q^{(s)}\}|$.

A *time series dataset* $\mathcal{D} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{D}|}\}$ from a network N is a bag of $|\mathcal{D}|$ *transactions*, which are paths on N. Given a path w on N, the number of occurrences $Occ_{\mathcal{D}}(w)$ of w in $\mathcal{D}$ is the sum of the number of occurrences $Occ_\tau(w)$ of w in τ, $\forall \tau \in \mathcal{D}$, that is, $Occ_{\mathcal{D}}(w) = \sum_{\tau\in\mathcal{D}} Occ_\tau(w)$.

Given a positive integer ℓ, the task of *mining paths* of length ℓ from a time series dataset $\mathcal{D}$ from a network N is the task of mining the set $\mathcal{W}_{\mathcal{D}}(\ell)$ of all paths of length ℓ that occur at least once in $\mathcal{D}$ and the number of their occurrences, that is,

$$\mathcal{W}_{\mathcal{D}}(\ell) = \{(w, Occ_{\mathcal{D}}(w)) : |w| = \ell \wedge Occ_{\mathcal{D}}(w) > 0\}.$$

With an abuse of notation, in the following we use $w \in \mathcal{W}_{\mathcal{D}}(\ell)$ to indicate that $\exists (w, Occ_{\mathcal{D}}(w)) \in \mathcal{W}_{\mathcal{D}}(\ell)$.

B. k-th Order De Bruijn Graph

Given a directed graph $G = (V, E)$ and an integer $k > 0$, the *k-th order De Bruijn graph* $G^k = (V^k, E^k)$ of G is a directed graph where each vertex $v^k \in V^k$ is a path of length $k - 1$ on G, i.e., $v^k = \{v_{i_0}, v_{i_1}, \ldots, v_{i_{k-1}}\}$, and an ordered pair (v^k, u^k), with $v^k = \{v_{i_0}, v_{i_1}, \ldots, v_{i_{k-1}}\}$, $u^k = \{u_{j_0}, u_{j_1}, \ldots, u_{j_{k-1}}\} \in V^k$, it is an edge of G^k if and only if $v_{i_t} = u_{j_{t-1}} \ \forall t \in \{1, \ldots, k - 1\}$. Thus, each edge $(v^k, u^k) \in E^k$ is a path of length k on G, since $(v^k, u^k) = \{v_{i_0}, v_{i_1} = u_{j_0}, v_{i_2} = u_{j_1}, \ldots, v_{i_{k-1}} = u_{j_{k-2}}, u_{j_{k-1}}\}$. Let us note that G itself is a 1-st order De Bruijn graph of G. Fig. 1 (right) shows an example of k-th order De Bruijn graph.

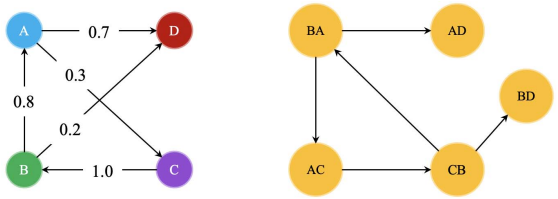


Fig. 1. Example of network and of De Bruijn graph. It shows the network $N = (G, \omega)$ (left) and the 2-nd order De Bruijn graph $G^2 = (V^2, E^2)$ of G (right). The network N is composed by the directed graph $G = (V, E)$, with $V = \{A, B, C, D\}$ and $E = \{(A, C), (A, D), (B, A), (B, D), (C, B)\}$, and by the weight function ω, such that $\omega((A, C)) = 0.3, \omega((A, D)) = 0.7$, $\omega((B, A)) = 0.8$, $\omega((B, D)) = 0.2$ and $\omega((C, B)) = 1.0$. The path $w = CBAC$ is an example of path of length $|w| = 3$ on N. The 2-nd order De Bruijn graph G^2 is composed by $V^2 = \{AC, AD, BA, BD, CB\}$, where each $v^2 \in V^2$ is a path of length 1 on G, and by $E^2 = \{(AC, CB), (BA, AC), (BA, AD), (CB, BA), (CB, BD)\}$, where each $(v^2, u^2) \in E^2$ is a path of length 2 on G.

C. Multiple Hypothesis Testing for Paths

The task of mining statistically significant paths is to identify paths whose number of occurrences in a dataset $\mathcal{D}$ is *significant*, or unexpected, with respect to the distribution of the weight function of the network that generated such data. To assess the significance of a path, we employ the framework of statistical hypothesis testing. For each path w, let H_w be the null hypothesis that the number of occurrences $Occ_{\mathcal{D}}(w)$ of w on $\mathcal{D}$ well conforms to the number of its occurrences in *random* time series data generated from the network $N = (G, \omega)$. That is, data generated from the graph G in accordance with the weight function ω while preserving the starting vertices observed in the original data and that have the same number of paths of interest of $\mathcal{D}$.

Under the null hypothesis, the number of occurrences of w is described by a random variable X_w, and in order to assess the significance of w, a p-value p_w is commonly computed. The p-value p_w of w is the probability of observing a number of occurrences, under the null hypothesis, at least as large as the number of occurrences $Occ_{\mathcal{D}}(w)$ of w in $\mathcal{D}$, that is,

$$p_w = \Pr\left[X_w \geq Occ_{\mathcal{D}}(w) | H_w\right].$$

For complex null hypotheses, the p-values can not be computed analytically, since there is not a closed form for X_w. However, when one can generate random data from the distribution described by the null hypothesis, the p-values can be estimated by a simple Monte Carlo (MC) procedure as follows: to generate M random time series datasets $\tilde{\mathcal{D}}_i$, with $i \in \{1, \ldots, M\}$, from the distribution described by the null hypothesis. Then, the p-value p_w is estimated as

$$p_w = \frac{1}{M+1}\left(1 + \sum_{i=1}^{M} \mathbb{1}\left[Occ_{\tilde{\mathcal{D}}_i}(w) \geq Occ_{\mathcal{D}}(w)\right]\right), \quad (1)$$

where $\mathbb{1}[\cdot]$ is the indicator function of value 1 if the argument is true, and 0 otherwise.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

The statistical hypothesis testing framework is commonly used to provide guarantees on the false discoveries, i.e., paths flagged as significant while they are not. When a single path w is tested for significance, flagging w as significant, i.e., rejecting the null hypothesis, when $p_w \leq \alpha$, where $\alpha \in [0,1]$ is the *significance threshold* fixed by the user, guarantees that the probability that w corresponds to a false discovery $\leq \alpha$.

The situation is completely different when several paths are tested simultaneously, as in the case of path mining. If d paths are tested with the approach used for a single path, i.e., each path is flagged as significant if its p-value is $\leq \alpha$, then the expected number of false discoveries can be as large as αd. To solve this issue, one identifies a *corrected significance threshold* $\delta \in [0,1]$ such that all paths with p-value $\leq \delta$ can be reported as significant while providing some guarantees on the number of false discoveries. A common approach is to identify δ that provides guarantees on the Family-Wise Error Rate (FWER), defined as the probability of reporting at least one false positive, that is, if FP is the number of false positives, then FWER $= \Pr[\text{FP} > 0]$. For a given value δ, let FWER(δ) be the FWER obtained when δ is used as corrected significance threshold, that is, by reporting as significant all paths with p-value $\leq \delta$. Often FWER(δ) can not be evaluated in closed form, and thus approaches, as the Bonferroni correction or based on permutation testing described below, must be employed.

The Westfall-Young (WY) method [19] is a multiple hypothesis testing procedure based on permutation testing that results in high statistical power and that has been successfully applied in other pattern mining scenarios [10]–[12]. The WY method directly estimates the joint distribution of null hypotheses using permuted datasets, i.e., datasets obtained from the distribution described by the null hypothesis. In detail, the WY method considers P random datasets $\tilde{\mathcal{D}}_i$, with $i \in \{1,\dots,P\}$, generated from the distribution described by the null hypothesis. Then, for every dataset $\tilde{\mathcal{D}}_i$, with $i \in \{1,\dots,P\}$, it computes the minimum p-value $p^{(i)}_{min}$ over all paths of interest in $\tilde{\mathcal{D}}_i$. The FWER FWER(δ) obtained using δ as corrected significance threshold can then be estimated as

$$\text{FWER}(\delta) = \frac{1}{P}\sum_{i=1}^{P} \mathbb{1}\left[p^{(i)}_{min} \leq \delta\right]. \tag{2}$$

Thus, given a *FWER threshold* $\alpha \in [0,1]$, the corrected significance threshold δ^* is obtained as

$$\delta^* = \max\{\delta : \text{FWER}(\delta) \leq \alpha\}. \tag{3}$$

III. caSPiTa: Mining Statistically Significant Paths

In this section, we describe our method caSPiTa, mining statistiCAlly Significant Paths In Time series dAta, to mine *statistically significant paths* in time series data generated by a network, while controlling the probability of having at least one false discovery, i.e., the FWER. Given a time series dataset $\mathcal{D}$ from an *unknown* network N, we aim to mine statistically significant paths, that are paths that have a number of occurrences on $\mathcal{D}$ that is surprising, i.e., higher than the expected number of their occurrences under the null hypothesis. In particular, given two natural values $k, h \in \mathbb{N}^+$, with $k > h$, we aim to mine length k paths from $\mathcal{D}$ whose number of occurrences are not due to the number of occurrences of length h paths observed in $\mathcal{D}$, with the idea that such paths of length h represent some well-know substructures in the underlying and unknown network.

The idea behind caSPiTa is the following. First, we mine all the paths $\mathcal{W}_{\mathcal{D}}(k)$ of length k from the time series dataset $\mathcal{D}$. Since we do not know the network N from which $\mathcal{D}$ has been generated, we can not directly infer the statistical significance of such paths with respect to N, and thus we need to construct a new generative null model from the dataset $\mathcal{D}$. Such generative null model is then used to estimate the p-values and to compute the corrected significance threshold δ^* using the WY method. We now describe the generative null model employed by caSPiTa.

A. Generative Null Model

In this work, we aim to find length k paths whose number of occurrences in $\mathcal{D}$ are not due to the number of occurrences of length h paths in $\mathcal{D}$. Thus, we create a generative null model in accordance with the number of occurrences of the paths of length h in $\mathcal{D}$, and then we test the significance of the paths of length k using such model. Given a time series dataset $\mathcal{D}$, generated by an unknown network $N = (G, \omega)$, and $h \in \mathbb{N}^+$, we define the *h-th order generative model* $N^h(\mathcal{D})$ of the time series dataset $\mathcal{D}$ as a network $N^h(\mathcal{D}) = (G^h, \omega^h)$, where G^h is the h-th order De Bruijn graph of G (based on $\mathcal{D}$, since the entire structure of G is unknown), and ω^h is a weight function. The h-th order De Bruijn graph $G^h = (V^h, E^h)$ is composed as follows: $V^h = \{w \in \mathcal{W}_{\mathcal{D}}(h-1)\}$, while E^h is constructed as defined in Definition II-B. Thus, each vertex $v^h \in V^h$ is a path of length $h-1$ in $\mathcal{D}$ (and thus on G), while each edge $(u^h, v^h) \in E^h$ represents a path of length h in $\mathcal{D}$ (and thus on G). With an abuse of notation, in the following we use (u^h, v^h) to indicate both the edge in G^h and the corresponding path on G. Finally, the weight function ω^h is defined as follows: $\forall (u^h, v^h) \in E^h$,

$$\omega^h\left((u^h, v^h)\right) = \frac{Occ_{\mathcal{D}}\left((u^h, v^h)\right)}{\sum_{(u^h,:)\in E^h} Occ_{\mathcal{D}}\left((u^h,:)\right)}.$$

Let us note that the weight function ω^h of the h-th order generative null model $N^h(\mathcal{D})$ is defined using the observed number of occurrences of length h paths, and that each edge of $N^h(\mathcal{D})$ represents a path of length h. Thus, $N^h(\mathcal{D})$ correctly represents the distribution of the number of occurrences of the paths of length h in the dataset $\mathcal{D}$. An example of generative null models is shown in Fig. 2.

As explained in Section II-C, to compute the p-values p_w of the paths $w \in \mathcal{W}_{\mathcal{D}}(k)$ and to estimate the corrected significance threshold δ^*, we require random data generated from the generative null model. In the following two sections, we introduce two different strategies to generate random datasets

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

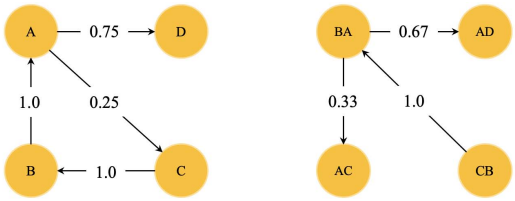


Fig. 2. Example of generative null models. It shows $N^1(\mathcal{D})$ (left) and $N^2(\mathcal{D})$ (right), respectively the 1-st and the 2-nd order generative null models of $\mathcal{D} = \{\tau_1 = BAD, \tau_2 = CBAC, \tau_3 = CBAD, \tau_4 = AD\}$, which is a possible time series dataset from the network N shown in Fig. 1. Let us note that the two generative models represent different probability distributions for the paths of length $> h$ and that they are based on the dataset $\mathcal{D}$ and not on the network N that generated $\mathcal{D}$. Indeed, they have some missing edges with respect to the network N (that is a 1-st order De Bruijn graph of itself) and its 2-nd order De Bruijn graph G^2, respectively, both shown in Fig. 1.

$\tilde{\mathcal{D}}$ that contain the same total number T of paths of length k of the original dataset $\mathcal{D}$, that is,

$$T = \sum_{w \in \mathcal{W}_{\mathcal{D}}(k)} Occ_{\mathcal{D}}(w) = \sum_{w \in \mathcal{W}_{\tilde{\mathcal{D}}(k)}} Occ_{\tilde{\mathcal{D}}}(w). \qquad (4)$$

First, we describe the *transactions oriented generation* (TOG) strategy, a natural way to generate random datasets performing a series of random walks that generate random transactions with characteristics similar to the ones of the transactions in $\mathcal{D}$. To overcome some issues of this strategy when it is applied to large generative null model, we then introduce the *paths oriented generation* (POG) strategy, an alternative approach that directly generates random length k paths. For this second strategy, we also introduce an approximation based on the binomial distribution that allows to estimate the p-values avoiding expensive MC procedures.

B. Transactions Oriented Generation (TOG) Strategy

In this section, we explain how to generate random datasets $\tilde{\mathcal{D}}$ from the generative null model $N^h(\mathcal{D})$ defined above using the TOG strategy. The idea is to perform a series of random walks that generate random transactions $\tilde{\tau}$ with characteristics similar to the ones of the transactions $\tau \in \mathcal{D}$. Characteristics that are natural to consider and that we want to preserve are: i) the dataset $\tilde{\mathcal{D}}$ has the same number of transactions of $\mathcal{D}$; ii) each transaction $\tilde{\tau} \in \tilde{\mathcal{D}}$ has the same length of the corresponding transaction $\tau \in \mathcal{D}$; iii) each transaction $\tilde{\tau} \in \tilde{\mathcal{D}}$ starts from the same vertex (of the generative null model) of the corresponding transaction $\tau \in \mathcal{D}$. Let us note that to preserve such characteristics guarantees to preserve also the total number T of length k paths in the dataset. As a motivation to consider such characteristics, let us consider the case in which the dataset $\mathcal{D}$ contains transactions that represent visits of some users in a website. In such a scenario, we are interested in preserving the web-pages from which the visits start, since they probably are homepages (or web-pages from which the users typically start their navigation). In addition, by preserving the length of the transactions, we preserve the number of web-pages that the users visit in a single navigation on the website.

Let s_τ be a path of length $|s_\tau| = h-1$ such that $s_\tau \subset \tau^{(0)}$, that is, s_τ is the vertex of $N^h(\mathcal{D})$ from which the transaction τ starts. The TOG strategy is the following. For each $\tau_i \in \mathcal{D}$, we perform a random walk on $N^h(\mathcal{D})$ of $|\tau_i| - (h-1)$ steps, starting from the vertex s_{τ_i}. At each step, the random walk moves from a vertex v^h to a vertex u^h with probability $\omega^h\left((v^h, u^h)\right)$. The path generated from such random walk is then the transaction $\tilde{\tau}_i \in \tilde{\mathcal{D}}$ that corresponds to the transaction $\tau_i \in \mathcal{D}$, with $|\tau_i| = |\tilde{\tau}_i|$. Performing all the $|\mathcal{D}|$ random walks, we generate the random dataset $\tilde{\mathcal{D}}$. Let us note that a random walk may reach a vertex without outgoing edges before performing the desired number of steps, generating a shorter transaction, and thus not preserving the second characteristic (and neither T). In such a case, we discard the transaction and repeat the random walk until we generate a transaction $\tilde{\tau}_i$ with $|\tilde{\tau}_i| = |\tau_i|$.

The TOG strategy is the most natural way to generate random data from the generative null model. However, when the generative null model is large, as happens in many real applications, the number of paths contained in a dataset is only a small fraction of the gargantuan number of paths that can be generated as sub-paths of such long transactions. Thus, the corrected significance threshold δ^* obtained with the WY method could be very small, resulting in few or even zero reported significant paths. In addition, depending on the structure of the generative null model, to generate such long transactions may be computationally expensive for the high number of transactions that we need to generate and discard before reaching the desired lengths.

C. Paths Oriented Generation (POG) Strategy

To overcome the issue of the TOG strategy, we now describe an alternative approach to generate random data. In the POG strategy, instead of generating long transactions, we generate single random paths w of length $|w| = k$. In particular, from the generative null model $N^h(\mathcal{D})$ defined above, we generate random datasets $\tilde{\mathcal{D}}$, which are bags of paths of length k, where the number of paths w of length $|w| = k$ that start in each vertex (of the generative null model) is the same in the two datasets $\mathcal{D}$ and $\tilde{\mathcal{D}}$. Let us note that to preserve such characteristic guarantees to also preserve the total number T of length k paths in the dataset.

Let us remember that s_w is a path of length $|s_w| = h-1$ such that $s_w \subset w^{(0)}$, that is, s_w is the vertex of $N^h(\mathcal{D})$ from which the path w starts, and let $\mathcal{S} = \{s_w : w \in \mathcal{W}_{\mathcal{D}}(k)\}$ be the set of vertices of $N^h(\mathcal{D})$ from which starts at least one path $w \in \mathcal{W}_{\mathcal{D}}(k)$. To generate random paths, for each vertex $s \in \mathcal{S}$, we perform a series of random walks of $k-(h-1)$ steps on $N^h(\mathcal{D})$, until we generate

$$n_s = \sum_{w \in \mathcal{W}_{\mathcal{D}}(k): s_w = s} Occ_{\mathcal{D}}(w) \qquad (5)$$

random paths w of length $|w| = k$ that start from such vertex s. Then, the bag of all the paths of length k generated from all the vertices $s \in \mathcal{S}$ is the random dataset $\tilde{\mathcal{D}}$. Let us note that $|\tilde{\mathcal{D}}| = \sum_{s \in \mathcal{S}} n_s = T$. As explained above, let us remember

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

that at each step the random walk moves from a vertex v^h to a vertex u^h with probability $\omega^h((v^h, u^h))$. Thus, each random walk generates a path of length k or a path of length $< k$ that ends in a vertex without outgoing edges. Since we are interested in paths of length k, we discard all generated paths of length shorter than k.

While the POG strategy overcomes the issue of the TOG strategy explained above reducing the space of paths that can be generated from the generative null model, it still requires expensive MC procedures to estimate the p-values, and such procedures could be computationally prohibitive for large datasets. In the following section, we introduce a method to approximate the p-values for the POG strategy avoiding the MC procedure.

1) Binomial Approximation for the p-values: In this section, we illustrate an approach to approximate the p-values p_w of paths w of length $|w| = k$ when the POG strategy is used to generate random data. First, we compute with which probabilities such paths are generated under the POG strategy. Let us consider a random walk that starts from a vertex $s \in \mathcal{S}$ and that performs $k - (h-1)$ steps on $N^h(\mathcal{D})$, and let $\mathcal{W}_s$ be the set of all paths that can be generated by such random walk. As explained above, the set $\mathcal{W}_s$ contains paths of length $|w| = k$ and, eventually, paths of length $|w| < k$ that end in a vertex without outgoing edges. Let $RW(w)$ be the set of edges of $N^h(\mathcal{D})$ that the random walk traverses to generate the path $w \in \mathcal{W}_s$. From the definition of random walk, the probability $\Pr(w)$ that the random walk generates $w \in \mathcal{W}_s$ starting from s is

$$\Pr(w) = \prod_{(u^h, v^h) \in RW(w)} \omega^h\left((u^h, v^h)\right).$$

Let us note that $\sum_{w \in \mathcal{W}_s} \Pr(w) = 1$. Let E_k be the event that the random walk generates a path of length exactly k and let $\mathcal{W}_s^k \subseteq \mathcal{W}_s$ be the set of paths $w \in \mathcal{W}_s$ with $|w| = k$. Since in the POG strategy we discard paths shorter than k which could be generated by the series of random walks, then the probability of generating the path $w \in \mathcal{W}_s^k$ is

$$\Pr(w \mid E_k) = \frac{\Pr(w \cap E_k)}{\Pr(E_k)}, \tag{6}$$

where $\Pr(w \cap E_k) = \Pr(w)$ for all $w \in \mathcal{W}_s^k$ and 0 otherwise, and $\Pr(E_k) = \sum_{w \in \mathcal{W}_s^k} \Pr(w)$. Again, let us note that $\sum_{w \in \mathcal{W}_s^k} \Pr(w \mid E_k) = 1$, and that if $\mathcal{W}_s \backslash \mathcal{W}_s^k = \emptyset$, then $\Pr(w \mid E_k) = \Pr(w)$. An example of these probabilities is shown in Fig 3.

Since from a given vertex $s \in \mathcal{S}$, we generate exactly n_s (see Equation 5) length k paths, then the number of occurrences $Occ_{\tilde{\mathcal{D}}}(w)$ of a path $w \in \mathcal{W}_s^k$ in the random dataset $\tilde{\mathcal{D}}$ follows a binomial distribution, that is, $Occ_{\tilde{\mathcal{D}}}(w) \sim \text{Bin}(n_s, \Pr(w \mid E_k))$. For a fixed vertex $s \in \mathcal{S}$, this is true for all the paths $w \in \mathcal{W}_s^k$, but the binomial distributions corresponding to these paths are not independent, and thus, the computation of the p-values p_w as

$$p_w = \Pr\left[\text{Bin}(n_s, \Pr(w \mid E_k)) \geq Occ_{\mathcal{D}}(w)\right] \tag{7}$$

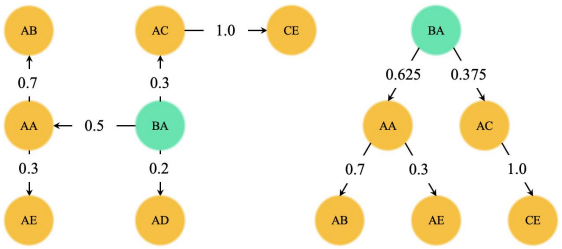


Fig. 3. Example of paths oriented generation. It shows the 2-nd order generative null model $N^2(\mathcal{D})$ and the starting vertex BA (left), and the probabilities of all paths of length 3 under the POG strategy (right). For $k = 3$, starting from the vertex BA and performing $k - (h-1) = 2$ steps, a random walk can generate the paths of length 3 $BAAE$, $BAAB$ and $BACE$, or can reach the vertex AD just after one step, generating the path of length 2 BAD. The probabilities of all these paths are: $\Pr(BAAE) = 0.15$, $\Pr(BAAB) = 0.35$, $\Pr(BACE) = 0.3$, and $\Pr(BAD) = 0.2$. Instead, their probabilities under the POG strategy are: $\Pr(BAAE \mid E_3) = 0.1875$, $\Pr(BAAB \mid E_3) = 0.4375$, and $\Pr(BACE \mid E_3) = 0.375$.

considers a number of paths that is in *expectation* n_s, and not exactly n_s as for the original POG strategy. (Note that, as a consequence, the *total* number of considered paths of length k is T *in expectation*.) However, in our experimental evaluation, we empirically show that the p-values for the binomial approximation are within one order of magnitude of the corresponding MC p-values, and, thus, that the binomial approximation is a valid approach to approximate the p-values for the POG strategy, avoiding expensive MC procedures.

Let us note that while this approximation does not require the generation of M random datasets $\tilde{\mathcal{D}}$ to estimate the p-values, CASPITA still requires the generation of P random datasets $\tilde{\mathcal{D}}$ for the WY method. However, the binomial approximation can also be used to approximate the minimum p-value in the P random datasets. Thus, given the observed dataset $\mathcal{D}$, we compute the p-values for all paths $w \in \mathcal{W}_{\mathcal{D}}(k)$ using Equation 7. Then, we generate a series of P random datasets $\tilde{\mathcal{D}}$ required by the WY method using the POG strategy. For all the P random datasets $\tilde{\mathcal{D}}$, we compute the minimum p-value over all the paths $w \in \mathcal{W}_{\tilde{\mathcal{D}}}(k)$, where the p-value p_w of w is computed with Equation 7 replacing $Occ_{\mathcal{D}}(w)$ with $Occ_{\tilde{\mathcal{D}}}(w)$.

D. Analysis

In this section, we describe in detail our algorithm CASPITA and formally prove its false positives guarantees. Algorithm 1 shows the pseudo-code of CASPITA. Its inputs are the time series dataset $\mathcal{D}$, the FWER threshold $\alpha \in [0, 1]$, the order $h > 0$ of the the generative null model, and the paths length $k > h$. For a given generation strategy, (i.e., TOG or POG), CASPITA first mines the set $\mathcal{W}_{\mathcal{D}}(k)$, of all paths w of length $|w| = k$ that occur at least once in $\mathcal{D}$. Then, it constructs the generative null model $N^h(\mathcal{D})$ as explained in Section III-A, and it uses $N^h(\mathcal{D})$ to compute the p-values of the paths $w \in \mathcal{W}_{\mathcal{D}}(k)$. The p-values can be computed with a MC procedure using Equation 1 (for both generation strategies), and thus generating M random datasets, where M is a parameter set by the user, or with the binomial approximation

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

using Equation 7 (for the POG strategy). To compute the corrected significance threshold δ^*, it then employs the WY method, which requires the generation of P random datasets, where P is a parameter set by the user. For each random dataset $\tilde{\mathcal{D}}_i$, with $i \in \{1, \ldots, P\}$, it mines the set $\mathcal{W}_{\tilde{\mathcal{D}}_i}(k)$ and then it computes the minimum p-value $p^{(i)}_{min}$ over all paths $\tilde{w} \in \mathcal{W}_{\tilde{\mathcal{D}}_i}(k)$. For the computation of such p-values, the considerations made above are still valid. The corrected significance threshold δ^* is then computed using Equation 3. If $\delta^* > \alpha$, then we set $\delta^* = \alpha$, corresponding to an uncorrected threshold. Finally, the output is the set of paths $w \in \mathcal{W}_{\mathcal{D}}(k)$ such that $p_w < \delta^*$. Theorem 1 proves that the output of CASPITA has FWER $\leq \alpha$.

Algorithm 1: CASPITA

Data: Time Series Dataset $\mathcal{D}$, FWER Threshold $\alpha \in [0, 1]$, Order $h > 0$ of the Generative Null Model, Paths Length $k > h$.
Result: Set $\mathcal{SW}$ with FWER $\leq \alpha$.

$\mathcal{W} \leftarrow$ `MinePaths`$(\mathcal{D}, k)$;
$N^h \leftarrow$ `GenerativeNullModel`$(\mathcal{D}, h)$;
foreach $w \in \mathcal{W}$ **do**
  $p_w \leftarrow$ `PValue`$\big(N^h, w, Occ_{\mathcal{D}}(w)\big)$;
for $i \leftarrow 1$ **to** P **do**
  $\tilde{\mathcal{D}}_i \leftarrow$ `RandomDataset`(N^h, k, h);
  $\mathcal{W}_i \leftarrow$ `MinePaths`$(\tilde{\mathcal{D}}_i, k)$;
  foreach $\tilde{w} \in \mathcal{W}_i$ **do**
    $p_{\tilde{w}} \leftarrow$ `PValue`$\big(N^h, \tilde{w}, Occ_{\tilde{\mathcal{D}}_i}(\tilde{w})\big)$;
  $p^{(i)}_{min} \leftarrow \min\{p_{\tilde{w}} : \tilde{w} \in \mathcal{W}_i\}$;
$\delta^* \leftarrow \max\left\{\delta : \sum_{i=1}^{P} \left(\mathbb{1}[p^{(i)}_{\min} \leq \delta]\right) \leq \alpha P\right\}$;
$\mathcal{SW} \leftarrow \{(w, Occ_{\mathcal{D}}(w), p_w) : w \in \mathcal{W} \wedge p_w < \delta^*\}$;
return $\mathcal{SW}$;

Theorem 1. *The output of* CASPITA *has FWER* $\leq \alpha$.

Proof. Let us consider the P random datasets $\tilde{\mathcal{D}}_i$, with $i \in \{1, \ldots, P\}$, generated by CASPITA for the WY method. Let us note that they do not contain any significant paths of length k, since they are generated from the generative null model N^h, and thus from the distribution described by the null hypothesis. Given $\delta \in [0, 1]$, the FWER FWER(δ) obtained using δ as significance threshold can be estimated using Equation 2. That is, estimated as the fraction, over P, of the number of datasets $\tilde{\mathcal{D}}_i$ that contain at least one path with p-value $\leq \delta$, and thus a path that would be reported as significant while it is not when δ is used as significance threshold. Since CASPITA uses the corrected significance threshold $\delta^* = \max\{\delta : \text{FWER}(\delta) \leq \alpha\}$, then its output has FWER $\leq \alpha$, which concludes our proof. □

We now provide a brief analysis of the time complexity of CASPITA. The time complexity t_W to mine $\mathcal{W}_{\mathcal{D}}(k)$ and t_N to construct $N^h(\mathcal{D})$ are $t_W = t_N = \mathcal{O}(D)$, with $D = \sum_{\tau \in \mathcal{D}} |\tau|$, since they can be done with a single scan of the entire dataset. The time complexity t_P^{MC} to estimate the p-values of all paths $w \in \mathcal{W}_{\mathcal{D}}(k)$ using MC procedures is $t_P^{MC} = \mathcal{O}(M \cdot t_{\tilde{\mathcal{D}}} + |\mathcal{W}(\mathcal{D}(k)|)$, where $t_{\tilde{\mathcal{D}}}$ is the time complexity to generate a random dataset $\tilde{\mathcal{D}}$ and M is the number of random datasets to create. Let us note that $t_{\tilde{\mathcal{D}}}$ depends on the generation strategy, and that it is not trivial to bound it since we do not know in advance the number of random walks that we need to generate $\tilde{\mathcal{D}}$. However, our experimental evaluation empirically proves that such random datasets can be generated with feasible computational time. In addition, the MC procedure is well-suited to parallelization: when C cores are used to compute the p-values considering M random datasets, each core computes the p-values on M/C random datasets, and the results are then aggregated at the end. The time complexity t_P^B to compute the p-values of all paths $w \in \mathcal{W}_{\mathcal{D}}(k)$ using the binomial approximation is instead $t_P^B = \mathcal{O}(|\mathcal{W}(\mathcal{D}(k)|)$, but it first requires the computation of the probabilities of Equation 6. Such computation has time complexity $\mathcal{O}(|\mathcal{S}| \cdot R^{MAX})$, where $|\mathcal{S}|$ is the number of starting nodes and R^{MAX} is the maximum, over all $s \in \mathcal{S}$, number of vertices that can be reached in $k - h$ steps on $N^h(\mathcal{D})$, starting from each vertex s. Finally, the time complexity of the WY method is $t_{WY} = \mathcal{O}(P \cdot (t_{\tilde{\mathcal{D}}} + t_p))$, with P the number of random datasets to generate and t_p one of the two time complexity described above to compute the p-values.

As previously stated, while here we consider the mining of over represented paths, all our reasoning can be easily adapted to the mining of under represented paths.

IV. Mining Statistically Significant Paths from Different Datasets

In this section, we illustrate another interesting scenario in which our algorithm CASPITA can be applied. Let us suppose to have two datasets, $\mathcal{D}_1$ and $\mathcal{D}_2$, and that such two datasets are taken from the same network N, but in different circumstances, e.g., in different temporal points, or maybe that they represent data generated from two different populations, e.g., men and women. In such a scenario, one may be interested in finding paths from one of the two datasets that are statistically significant considering the distribution represented by the other dataset. Thus, it is possible to use a slightly modified version of CASPITA, considering the dataset $\mathcal{D}_1$ to generate the h-th order generative null model $N^h(\mathcal{D}_1)$ and then, to consider the paths mined from the other dataset $\mathcal{W}_{\mathcal{D}_2}(k)$, and to compute their significance using $N^h(\mathcal{D}_1)$. Differently from the scenario described above, in this setting it is also possible to mine statistically significant paths of length k considering the h-th order generative null model with $k = h$.

V. Experimental Evaluation

In this section, we report the results of our experimental evaluation on multiple pseudo-artificial and real datasets to assess the performance of CASPITA for mining statistically significant paths from an unknown network.

The goals of the evaluation are the following: i) to prove that for small datasets, CASPITA is able to find statistically significant paths with both generation strategies, i.e., TOG and

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

POG, using the MC procedure, while for larger datasets the binomial approximation is necessary to provide useful results; ii) to prove that the binomial approximation is a valid approach to approximate the p-values for the POG strategy; iii) focusing on the POG strategy with the binomial approximation, to prove that CASPITA is able to find large sets of statistically significant paths in real large datasets, while avoiding false positives, and compare CASPITA with HYPA [17]; iv) to prove that CASPITA is able to find statistically significant paths in the scenario in which the generative null model is constructed considering data from an other dataset (see Section IV).

A. Environment and Datasets

We implemented CASPITA in Java. We performed all the experiments on the same machine with 512 GB of RAM and 2 Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.3GHz, using Java 1.8.0_201. To parallelize the MC procedures, we used Apache Spark Java API version 3.1.1. Our open-source implementation of CASPITA and the code developed for the tests and to generate the datasets are available in [20]. In all the experiments, we fixed the FWER threshold to the commonly used value $\alpha = 0.05$. To compare with HYPA [17], we used their implementation available online.[1]

In the following, we describe the datasets used in the evaluation, whose characteristics are shown in Table I (more details about their generation are available in [20]): i) BIKE: 2019 data on the bike sharing service of Los Angeles.[2] Each vertex is a bike station, each transaction is the sequence of bike stations visited by a given bike; ii) BIKE10 and BIKE20: smaller versions of the BIKE dataset. From BIKE, we only considered the 10 or 20 vertices, respectively, that occur more times, and collect all the transactions that only contain such vertices; iii) FLIGHT: 2019 data from commercial flights in the USA.[3] Each vertex is an airport, each transaction is the sequence of airports visited in a single itinerary by a passenger; iv) WIKI: human navigation paths[4] on Wikipedia, collected through the human-computation game Wikispeedia [21]. Each vertex is a Wikipedia web-page, while each transaction is a sequence of web-pages visited by a user during a game.

B. Generation Strategies Comparison

In this section, we compare the results obtained by CASPITA with the TOG or POG strategies that employ MC procedures, and the POG strategy that uses the binomial approximation, on BIKE10 and BIKE20.

We report all the results in [20], and describe here the main observations. The experiments have been performed with $P = 1000$, $M = 10^5$, $k \in \{2, \ldots, 5\}$, and $h \in \{1, \ldots, k-1\}$. For BIKE10, the smallest dataset, the number of significant paths obtained with the TOG and POG strategies with MC procedures differs from at most 1, for all combinations of parameters. The same is true when the POG strategy with the binomial approximation is used. In all the cases, CASPITA reported at most 3 statistically significant paths, which is not surprising since BIKE10 only contains few distinct paths. For BIKE20, the situation is different. For some combinations of parameters, CASPITA with the MC procedures did not report any significant (over represented) paths, while it reported some paths (from 1 to 8) when the binomial approximation is used. In all such cases, the MC estimates resulted in a corrected threshold $\delta^* = 1/(M+1)$, corresponding to the minimum achievable p-value considering M random datasets. Thus, to be able to mine paths, one has to consider a larger value of M, which is infeasible with larger datasets, or to resort to the binomial approximation. This phenomenon appeared with $k > h - 1$, that is, when a large number of distinct paths can be generated, even for a small dataset such as BIKE20. This emphasizes the issue of the TOG strategy described above, that is, the gargantuan number of paths that must be considered with the generation of long transactions.

TABLE I
DATASETS CHARACTERISTICS. $|\mathcal{D}|$: NUMBER OF TRANSACTIONS; AVG $|\tau|$: AVERAGE TRANSACTION LENGTH; MAX $|\tau|$: MAXIMUM TRANSACTION LENGTH; FOR THE 1-ST GENERATIVE NULL MODEL $N^1(\mathcal{D})$, $|V^1|$: NUMBER OF VERTICES; $|E^1|$: NUMBER OF EDGES.

Dataset $\mathcal{D}$	$\lvert\mathcal{D}\rvert$	Avg $\lvert\tau\rvert$	Max $\lvert\tau\rvert$	$N^1(\mathcal{D})$	
				$\lvert V^1\rvert$	$\lvert E^1\rvert$
BIKE10	3025	1.54	11	10	76
BIKE20	5080	1.90	21	20	279
BIKE	38651	7.51	232	237	10269
FLIGHT	17447803	1.63	15	455	69234
WIKI	51307	5.76	434	4169	59530

We then compared the p-values from the POG strategy obtained with the MC procedure and the binomial approximation. Note that while in the MC procedure the total number of length k paths starting from a vertex is fixed to the value observed in the data, using the binomial approximation such property holds only in expectation. Thus, the p-values from the two approaches will be different. However, by comparing the p-values[5] for all paths (over and under represented) of BIKE10 and BIKE20 with $k \in \{2, \ldots, 5\}$ and $h \in \{1, \ldots, k-1\}$, and considering $M \in \{10^4, 10^5, 10^6\}$ random datasets for the MC estimates, we observed that the p-values for the binomial approximation are within one order of magnitude of the corresponding MC p-values, and that the difference between binomial p-values and MC p-values is lower than the standard deviation of the MC estimates (obtained from 5 estimates of the MC p-values). Furthermore, the binomial approximation is several order of magnitude faster than the MC procedure (few milliseconds against over 40 seconds, considering the maximum execution time for both strategies and using 8 cores to parallelize the MC estimates).

[1] https://github.com/tlarock/hypa
[2] https://bikeshare.metro.net/about/data/
[3] https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FLM
[4] https://snap.stanford.edu/data/wikispeedia.html

[5] We only considered p-values $\geq 1/(M+1)$, since lower p-values require larger M to be correctly estimated with the MC procedure.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

C. Results for POG Strategy with Binomial Approximation

Since the results of the previous section demonstrated that the POG strategy with the binomial approximation is necessary to mine statistically significant paths from large datasets, and that the p-values for the binomial approximation are within one order of magnitude of the corresponding MC p-values, in this section we focus on such version of CASPITA.

First, we investigated the false positives guarantees of CASPITA on pseudo-artificial datasets. Starting from a real dataset, we created its h-th order generative null model, which we used to generate random datasets using the POG strategy. Thus, each random dataset is a bag of paths of a given length $k > h$ that does not contain any significant path of length k (since they have been generated in accordance with the generative null model). We then executed CASPITA on each random dataset, with parameters h and k corresponding to the ones used to generate the random dataset, and we checked whether CASPITA reported some paths, which would be false positives by construction. We considered BIKE10 and BIKE20 as starting real datasets, $k \in \{2, \ldots, 5\}$, and $h \in \{1, \ldots, k-1\}$, mining under and over represented paths. Given a real dataset, we generated 20 random datasets for each combinations of h and k, obtaining a total number of 400 runs for each real dataset. We then estimated the FWER as the fraction of runs with at least one false positive. For BIKE10, the estimated FWER is 0.75% with $P = 100$, 1.25% with $P = 1000$, and 0.5% with $P = 10000$. Instead, for BIKE20, the estimated FWER is 2.25% with $P = 100$, 1.75% with $P = 1000$, and 3.25% with $P = 10000$. These results show that the false positives guarantees of CASPITA are even better than the theoretical ones, which are $\leq 5\%$ using $\alpha = 0.05$, and that $P = 100$ is enough to obtain such guarantees. Using these random datasets, we also made a comparison with HYPA. Let us remember that HYPA employs a fixed threshold β to flag as anomalous a path, without any theoretical guarantees. We used $\beta \in \{0.00001, 0.001, 0.05\}$, which are the minimum, the most commonly used, and the maximum value used in [17], and $k \in \{2, \ldots, 5\}$. (For h, HYPA always considers $h = k-1$, thus we only used this value.) For BIKE10, it obtained an estimated FWER of 40.63% with $\beta = 0.05$, 15.63% with $\beta = 0.001$, and 0.0% with $\beta = 0.00001$. Instead, for BIKE20, it obtained an estimated FWER of 60.63% with $\beta = 0.05$, 32.50% with $\beta = 0.001$, and 1.25% with $\beta = 0.00001$. These results show that HYPA is able to return anomalous paths achieving low FWER with the correct threshold, but also emphasize the importance of having a strategy, as the one that we employ, to compute such threshold in an automatic way, since the usage of fixed thresholds may lead to many spurious discoveries, or to a low statistical power.

We then executed CASPITA on some real datasets, i.e., BIKE, FLIGHT, and WIKI. Table II reports the results obtained with $P = 100$, $k \in \{2, \ldots, 5\}$, and $h \in \{1, \ldots, k-1\}$. For all the datasets, and for almost all combinations of parameters, CASPITA reported some significant paths. It is interesting to notice that the number of over represented paths is almost always (some order of magnitude) greater than the number of under represented paths. In addition, for a fixed value of k, the number of over represented paths always decreases considering higher values of h. The number of under represented paths, instead, always decreases for BIKE, while increases and then decreases for FLIGHT and WIKI, highlighting different substructures in the three underlying networks. The computational time ranges from under 3 minutes (BIKE with $k = 2$, $h = 1$) to over 12 hours (FLIGHT with $k = 5$, $h = 1$). Let us note that FLIGHT has over $17M$ transactions, but we are still able to analyze it with a reasonable running time. The combination $k = 5$ and $h = 1$ is always the most expensive from a computational point of view, since it requires to generate longer paths and also to analyze a large number of vertices to compute the probabilities of Equation 7. Overall, these results show that CASPITA is able to efficiently mine significant paths from real datasets, with feasible computational time even in huge datasets.

D. Analysis of BIKE

In this section, we provide a brief analysis of some paths returned by CASPITA from BIKE. The over represented path of length 2 with the lowest p-value and highest number of occurrences is a path which starts and ends in "Ocean Front Walk & Navy" located in Venice Beach. The fact that this path is over represented indicates that people tend to leave and then come back to this place, instead of moving to other parts of the city. For example, such pattern may capture the fact that people leave the beach to buy some food and then immediately come back. Instead, the under represented path of length 2 with the lowest p-value is a path which starts from "Union Station West Portal", goes to "Main & 1st", and then comes back. "Main & 1st" is located near the Los Angeles City Hall, the center of the government of the city, while "Union Station West Portal" is near the Union Station, the main railway station of the city. The fact that this path is under represented is probably due to the fact that a lot of people move from the station to the city hall, and vice-versa, but in particular moments of the day, i.e., in the morning and in the evening. Thus, even if the two direct links are very popular, it is uncommon to see this entire path. These are only two example of paths mined by CASPITA, but they highlight its capability in detecting real life trends.

Finally, we investigated the capability of CASPITA in mining significant paths in the scenario in which the generative null model is created considering a different dataset (see Section IV). Using the procedure to generate BIKE (described in [20]), we generated a new dataset, NEWBIKE, considering the 2020 data from the same website. We then used the original BIKE dataset to create the h-th order generative null model and we tested on it the significance of length k paths mined from NEWBIKE. Given the pandemic situation that involved the world, one may be interested in finding changes in the habits of the people defining a model based on paths traveled in 2019 to test paths traveled in 2020. Table III reports the results obtained with $k = h \in \{1, \ldots, 5\}$ and $P = 100$. (For NEWBIKE, we only considered full transactions that can be

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

TABLE II

CASPITA RESULTS WITH REAL DATASETS. k: PATHS LENGTH; h: ORDER OF THE NULL MODEL; FOR EACH DATASET, BIKE, FLIGHT, AND WIKI, $|\mathcal{W}|$: NUMBER OF DISTINCT PATHS OF LENGTH k; T: NUMBER OF TOTAL PATHS OF LENGTH k; NUMBER OF SIGNIFICANT PATHS REPORTED, OVER $(+)$ AND UNDER $(-)$ REPRESENTED; TIME (S): EXECUTION TIME IN SECONDS.

k	h	BIKE					FLIGHT					WIKI				
		$\|\mathcal{W}\|$	T	$+$	$-$	Time (s)	$\|\mathcal{W}\|$	T	$+$	$-$	Time (s)	$\|\mathcal{W}\|$	T	$+$	$-$	Time (s)
2	1	90.4K	252K	197	28	150	574K	11.1M	96.5K	16.4k	5.73K	155K	244K	160	53	264
3	1	172K	221K	118	40	350	849K	5.16M	132K	36	10.0K	169K	194K	219	6	384
	2			1	8	375			128K	1.63K	13.3K			8	12	495
4	1	179K	195K	80	15	662	406K	530K	19.2K	0	2.85K	139K	147K	193	1	765
	2			10	7	639			10.4K	30	1.99K			16	6	747
	3			0	3	713			3.11K	19	1.52K			2	2	594
5	1	166K	174K	71	6	855	127K	155K	6.66K	0	43.9K	106K	108K	113	0	1.03K
	2			17	3	514			4.28K	0	1.31K			7	0	371
	3			0	1	458			1.80K	5	723			0	1	267
	4			0	1	365			884	2	638			0	0	211

TABLE III

CASPITA RESULTS ON BIKE CONSIDERING DIFFERENT DATASETS. SEE TABLE II FOR THE MEANING OF THE VALUES.

$k = h$	$\|\mathcal{W}\|$	T	$+$	$-$
1	5.79K	68.0K	256	120
2	5.51K	12.2K	30	9
3	1.19K	2.45K	14	1
4	436	786	4	0
5	124	203	0	0

generated by the generative null model obtained from BIKE.) Again, it is possible to notice that CASPITA returned paths for almost all combinations of parameters, and that the number of over represented paths is always higher than the number of under represented paths. Overall, these results demonstrate that CASPITA is able to mine paths also in such a scenario.

VI. CONCLUSIONS

In this work, we introduced the problem of mining *statistically significant paths* in time series data from an unknown network, which naturally arises in several applications. We described CASPITA, an algorithm to mine statistically significant paths (over and under represented) while controlling the probability of reporting at least one false positive, i.e., the FWER, employing the Westfall-Young method. We also introduced an alternative scenario which considers a different dataset to construct the generative null model. Our extensive experimental evaluation shows that CASPITA is able to efficiently mine large sets of significant paths from real datasets, while correctly controlling the FWER. Interesting future directions are the extension of CASPITA to mine statistically significant paths while controlling the *false discovery rate* (FDR) and to identify the k most significant paths.

ACKNOWLEDGMENT

Part of this work was supported by the MIUR, the Italian Ministry of Education, University and Research, under the initiative "Departments of Excellence" (Law 232/2016) and PRIN Project n. 20174LF3T8 "AHeAD", and by the University of Padova project "SID 2020: RATED-X".

REFERENCES

[1] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: a survey and empirical demonstration," *Data Min. Knowl. Discov.*, 2003.
[2] J. F. Roddick et al., "An updated bibliography of temporal, spatial, and spatio-temporal data mining research," *Int. Workshop on Temporal, Spatial, and Spatio-Temporal Data Min.*, 2000.
[3] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys*, 2012.
[4] G. M. Weiss, "Mining with rarity: a unifying framework," *ACM SIGKDD Explorations Newsletter*, 2004.
[5] E. Keogh et al., "Finding surprising patterns in a time series database in linear time and space," *ACM SIGKDD*, 2002.
[6] J. Lin et al., "Visually mining and monitoring massive time series," *ACM SIGKDD*, 2004.
[7] E. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: implications for previous and future research," *Knowl. Info. Syst.*, 2005.
[8] L. Wei and E. Keogh, "Semi-supervised time series classification," *ACM SIGKDD*, 2006.
[9] W. Hämäläinen and G. I. Webb, "A tutorial on statistically sound pattern discovery," *Data Min. Knowl. Discov.*, 2019.
[10] F. Llinares-López et al., "Fast and memory-efficient significant pattern mining via permutation testing," *ACM SIGKDD*, 2015.
[11] L. Pellegrina and F. Vandin, "Efficient mining of the most significant patterns with permutation testing," *Data Min. Knowl. Discov.*, 2020.
[12] A. Tonon and F. Vandin, "Permutation strategies for mining significant sequential patterns," *IEEE ICDM*, 2019.
[13] M. Gupta et al., "Outlier detection for temporal data: A survey," *Trans. Knowl. Data Eng.*, 2013.
[14] F. Lemmerich et al., "Mining subgroups with exceptional transition behavior," *ACM SIGKDD*, 2016.
[15] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," *ACM SIGKDD*, 2003.
[16] L. Akoglu et al., "Graph based anomaly detection and description: a survey," *Data Min. Knowl. Discov.*, 2015.
[17] T. LaRock et al., "Hypa: Efficient detection of path anomalies in time series data on networks," *SIAM SDM*, 2020.
[18] I. Scholtes, "When is a network a network? multi-order graphical model selection in pathways and temporal networks," *ACM SIGKDD*, 2017.
[19] P. H. Westfall and S. S. Young, "Resampling-based multiple testing: Examples and methods for p-value adjustment," *John Wiley & Sons*, 1993.
[20] https://github.com/VandinLab/CASPITA.
[21] R. West and J. Leskovec, "Human wayfinding in information networks," *WWW*, 2012.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:33:53 UTC from IEEE Xplore. Restrictions apply.

Efficient Detection of Network Motifs

Sebastian Wernicke

Abstract—Motifs in a given network are small connected subnetworks that occur in significantly higher frequencies than would be expected in random networks. They have recently gathered much attention as a concept to uncover structural design principles of complex networks. Kashtan et al. [*Bioinformatics*, 2004] proposed a sampling algorithm for performing the computationally challenging task of detecting network motifs. However, among other drawbacks, this algorithm suffers from a sampling bias and scales poorly with increasing subgraph size. Based on a detailed analysis of the previous algorithm, we present a new algorithm for network motif detection which overcomes these drawbacks. Furthermore, we present an efficient new approach for estimating the frequency of subgraphs in random networks that, in contrast to previous approaches, does not require the explicit generation of random networks. Experiments on a testbed of biological networks show our new algorithms to be orders of magnitude faster than previous approaches, allowing for the detection of larger motifs in bigger networks than previously possible and thus facilitating deeper insight into the field.

Index terms—Network motif detection algorithm, subgraph enumeration, subgraph sampling, subgraph concentration in random graphs.

1 Introduction

Many biological networks[1] appear to contain certain small subnetworks in significantly higher frequencies than random networks. For example, protein-protein interaction networks contain some three- and four-node substructures far more often than one would expect from a random network with similar mathematical properties [19], [36]. Based on the idea that "evolution preserves modules that define specific [...] functions" [31], Milo et al. [23], [24] proposed using such overabundant "topological modules" [31] to uncover the structural design principles of biological networks, thereby coining the term *network motifs* for them.[2]

Some excitement has surrounded the network motif approach with the original paper by Milo et al. [24] being cited well over 60 times in some major scientific journals as of March 2006. The analysis of network motifs has led to interesting results (of which we name only a few here), e.g., in the areas of protein-protein interaction prediction [1], hierarchical network decomposition [12], and the analysis of temporal gene expression patterns [14], [27], [28]. The transcriptional network of *Escherichia coli* displays motifs to which specific functionalities such as the generation of temporal expression programs or the response to fluctuating external signals can be attributed [24], [28], suggesting that network motifs play key information processing roles in this type of network [15]. The same motifs as in the transcriptional interaction network of *E. coli* were also identified for the yeast *Saccharomyces cerevisiae*, possibly hinting that common network function implies the sharing of common motifs [18]. Recently, the converse assertion that common motifs imply a common network function has also been made [23].

To put research on network motifs into proper perspective, it should be noted that it has also been met with some criticism. Vázquez et al. [30] demonstrated that global network features such as the clustering coefficient also influence local features such as the abundance of certain subgraphs. Artzy-Randrup et al. [4] found that certain random network models (such as "preferential attachment" as introduced by Barabási and Albert [5]) lead to a display of motifs although there is no explicit selection mechanism for local structures. Milo et al. answer this criticism in [22] by suggesting not only to look at the overabundance of *individual* subgraphs but rather at a broader picture in the form of so-called "subgraph significance profiles."

Focusing on the algorithmic aspects of network motif detection, this work does not intend to position itself in the discussion about the concept of network motifs. Rather, we present new algorithmic approaches which enable the analysis of larger networks and more complex motifs than previously possible, thus facilitating deeper insight into the field.

1.1 Previous Work

Much work related to network motifs has been spent on interpreting and applying the general concept, but considerably less on the involved algorithmics. Finding network motifs consists of three subtasks:

1. Find which subgraphs occur in the input graph and in which number.
2. Determine which of these subgraphs are topologically equivalent (that is, isomorphic) and group them into subgraph classes accordingly.

1. We use the terms "network" and "node" for fields outside mathematics and computer science. The terms "graph" and "vertex" are used for discussing algorithmic aspects.

2. Note that the term "network motif" has been used in other contexts as well and, e.g., may also refer to a common subnetwork in a set of given networks [26] or to any small labeled subnetwork (without considering connectivity or isomorphy) [8].

• *The author is with the Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany. E-mail: wernicke@minet.uni-jena.de.*

Manuscript received 31 Oct. 2005; revised 24 Mar. 2006; accepted 14 Apr. 2006; published online 31 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0127-1005.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.
1545-5963/06/$20.00 © 2006 IEEE Published by the IEEE CS, CI, and EMB Societies & the ACM

3. Determine which subgraph classes are displayed at a much higher frequency than in random graphs under a specified random graph model.

Performing the first subtask by explicitly enumerating all subgraphs of a certain size can be time consuming due to their potentially large number even in small, sparse networks. While some work has been spent on enumerating certain subgraph classes (such as cycles [2]), estimating the frequency of general subgraphs has seemingly attracted less consideration.[3] For this reason, Kashtan et al. [15] proposed an algorithm that estimates subgraph occurrences from a randomly sampled set of subgraphs. We discuss this algorithm in full detail in Section 2, mentioning only in passing here that it has a sampling bias which in turn leads to considerable drawbacks such as an inconsistent sampling quality and the need for a computationally expensive bias correction. Besides [15], we are only aware of the work, by Duke et al. [10], on approximating the number of size-k subgraphs in a given graph. However, their algorithm—which is based on Szemerédi's regularity lemma [29]—has no relevance in practice: In order to ensure a reasonable quality of approximation, the input graph has to be astronomically large and contain far more than $e^{k^{65}}$ vertices.

Much work has already been done concerning the second subtask and we rely on McKay's *nauty* algorithm [20], [21] for performing it in practice.

As to the third subtask, the standard approach for determining subgraph significance so far has been to explicitly generate an ensemble of random graphs (typically at least a thousand) under a given random graph model. One advantage of this approach is clearly that we can rely on a huge selection of existing random graph models which are able to account for, e.g., specific growth mechanisms [3] or built-in topologies [32]. However, independently of the random graph model, the approach of explicit generation is extremely time-consuming since we need to determine subgraph concentrations for each of these graphs just as in the original network. Here, we focus on the popular model of random graphs which preserve the degree sequence of the original network.[4] While there has been some research concerning the properties of graphs with prescribed degree sequence (such as the average path length [25]), the problem of subgraph distributions within such graphs has only been studied for size-3 subgraphs in directed sparse random graphs with *expected* degree sequences [13]. Section 3 introduces a new approach to estimate the concentration of any given size-k subgraph in random graphs with a given degree sequence without requiring their explicit generation.

1.2 Contribution and Structure of this Work

We provide significant improvements for the first and third subtask of motif detection. Based on a comprehensive analysis of the drawbacks encountered when using the subgraph sampling approach proposed by Kashtan et al. [15], Section 2 presents a new algorithm for subgraph sampling which does not suffer from these drawbacks. While this comes at the price of only being able to control the *expected* number of samples, our proposed algorithm is faster, much easier to implement, and shows some additional useful features, such as being able to quickly estimate the total number of size-k subgraphs in a given graph.

As to the task of determining subgraph significance, Section 3 proposes a new approach that does not require the explicit generation of random graphs with a prescribed degree sequence. This approach leads to a faster algorithm that is, moreover, able to focus on determining the significance of specific subgraphs (which is not possible with previous approaches).

Our algorithms have been implemented in C++; the source code and a user-friendly motif detection tool [33] based on our subgraph sampling algorithm are freely available online at http://theinf1.informatik.uni-jena.de/motifs/. For a testbed of biological networks, Section 4 shows that our algorithms detect network motifs by orders of magnitude faster than the implementation of Kashtan et al. This enables the analysis of larger networks and more complex motifs than previously possible.

2 A Faster Algorithm for Subgraph Sampling

The algorithm for subgraph sampling suggested by Kashtan et al. [15] is based on the idea that we start by selecting a random edge in the input graph and then randomly extend this edge until we obtain a connected subgraph with the desired number of vertices. Section 2.2 discusses this approach and its main drawbacks. We present a new approach to subgraph sampling based on randomized enumeration in Section 2.3.

2.1 Notation

Basic familiarity with graph-theoretic terminology is assumed. For a given graph $G = (V, E)$, we let $n \stackrel{\text{def}}{=} |V|$ and assume that all vertices in V are uniquely labeled by the integers $1, \ldots, n$. To abbreviate that the label of a vertex u is larger than that of a vertex v, we write "$u > v$." In order to simplify the presentation, edges in $G = (V, E)$ are always identified using set notation, regardless of whether the graph is directed or undirected. In the case that G is directed and contains the edges (u, v) and (v, u) for two vertices u and v, these edges become a single *bidirectional* edge $\{u, v\}$ in our notation.

For a set $V' \subseteq V$ of vertices, its *open neighborhood* $N(V')$ is the set of all vertices from $V \setminus V'$ which are adjacent to at least one vertex in V'. For a vertex $v \in V \setminus V'$, its *exclusive neighborhood* with respect to V', denoted $N_{excl}(v, V')$, consists of all vertices neighboring v that do not belong to $V' \cup N(V')$.

A connected subgraph that is induced by a vertex set of cardinality k is called *size-k subgraph*. For a given integer k, the set of all size-k subgraphs in G can be partitioned into sets $\mathcal{S}_k^i(G)$ called *subgraph classes*, where two size-k subgraphs belong to the same subgraph class if and only if they are isomorphic (that is, if and only if they are topologically

3. Note that the term "frequent subgraphs" is also used in the literature to denote common subgraphs in a set of given graphs (see, e.g., [17]).

4. Other models are also considered in the literature, e.g., additionally preserve the number of bidirectional edges in directed random networks.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

Algorithm: EDGE SAMPLING(G, k) (ESA)
Input: A graph $G = (V, E)$ and an integer $2 \leq k \leq |V|$.
Output: Vertices of a randomly chosen size-k subgraph in G.

```
01  {u, v} ← random edge from E
02  V' ← {u, v}
03  while |V'| ≠ k do
04      {u, v} ← random edge between V' and N(V')
05      V' ← V' ∪ {u, v}
06  return V'
```

Fig. 1. Pseudocode for the algorithm ESA which samples a random size-k subgraph in a given graph G.

equivalent). The *concentration* $\mathcal{C}_k^i(G)$ of a subgraph class $\mathcal{S}_k^i(G)$ is defined as

$$\mathcal{C}_k^i(G) \stackrel{\text{def}}{=} |\mathcal{S}_k^i(G)| \cdot \left(\sum_j |\mathcal{S}_k^j(G)| \right)^{-1}.$$

For a graph G, an integer k, and a set $\mathcal{R}$ of size-k subgraphs that were randomly sampled in G by some algorithm $\mathcal{A}$, a mapping $\hat{\mathcal{C}}_k^i : (\mathcal{R}, G) \to [0, 1]$ is called an *estimator* for $\mathcal{C}_k^i(G)$. We say that $\hat{\mathcal{C}}_k^i(\mathcal{R}, G)$ is *unbiased* (with respect to $\mathcal{A}$) if the expected value of $\hat{\mathcal{C}}_k^i(\mathcal{R}, G)$ equals $\mathcal{C}_k^i(G)$ and *biased* otherwise.[5]

2.2 The Previous Approach: Edge Sampling

For a given graph $G = (V, E)$ and an integer $k \geq 2$, Kashtan et al. [15] suggest sampling a random subgraph by starting with a randomly chosen edge and then adding neighboring vertices until a subgraph of the desired size k is obtained. A pseudocode description of this algorithm, which we will call ESA, is given in Fig. 1.

As already noted in [15], ESA has a bias for sampling certain subgraphs more often than others. Fig. 2 shows a concrete example we have constructed to illustrate this. The total number of connected size-3 subgraphs both in G_1 and G_2 is 28. Since the subgraph  occurs exactly once each in G_1 and G_2, we should expect that ESA samples with probability 1/28 within both graphs. However, we have

$$\Pr[\text{ESA samples } \triangle \text{ in } G_1] = \frac{1}{9} \cdot 1 + \frac{2}{9} \cdot \frac{2}{8} = \frac{1}{6}$$

and

$$\Pr[\text{ESA samples } \triangle \text{ in } G_2] = \frac{3}{12} \cdot \frac{2}{8} = \frac{1}{16}.$$

This illustrates some crucial problems of ESA: The subgraph is oversampled and—as a direct consequence—the only other occurring size-3 subgraph  is undersampled. The oversampling of is worse for G_1 than it is for G_2 and it is possible to show (using an adaptation of the above example) that the amount of bias cannot be estimated simply from the number of edges neighboring the oversampled subgraph.

5. Sidestepping a formal definition, by "expected value" we mean the average estimated subgraph concentration over a large number of runs of the sampling algorithm $\mathcal{A}$.

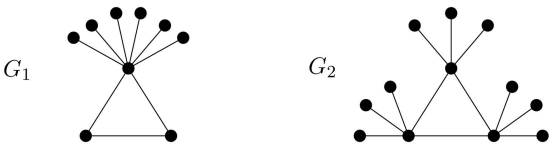


Fig. 2. The graphs G_1 and G_2 have an equal number of (connected) size-3 subgraphs. Furthermore, each of the two graphs has exactly one triangle among its size-3 subgraphs. As outlined in the text, ESA oversamples this subgraph in both G_1 and G_2, the oversampling being worse for G_1.

For a given set $\mathcal{R}$ of size-k subgraphs that were randomly sampled using ESA, the sampling bias can be circumvented by using the following unbiased estimator [15]:

$$\hat{\mathcal{C}}_k^i(\mathcal{R}, G) \stackrel{\text{def}}{=} \frac{\sum_{G' \in (\mathcal{R} \cap \mathcal{S}_k^i(G))} (\Pr[G' \text{ is sampled by ESA}])^{-1}}{\sum_{G' \in \mathcal{R}} (\Pr[G' \text{ is sampled by ESA}])^{-1}}. \quad (1)$$

The main idea here is that each subgraph is (ex post facto) scored inversely proportional to the probability that ESA samples it. While it is possible to correctly estimate $\mathcal{C}_k^i(G)$ in this way, several disadvantages remain:

- The bias itself remains as certain subgraphs are still (much) more likely to be sampled than others. This is especially problematic for subgraphs which appear in low concentration and are, at the same time, undersampled by ESA; they are hardly ever found.[6]
- Computing (1) is expensive since the calculation of *each single* probability can require as much as $\mathcal{O}(k^k)$ time [15]. It is also rather complicated to implement as, e.g., care needs to be taken to avoid numerical errors.
- We have no estimate as to what *fraction* of subgraphs has been sampled (which is of interest, e.g., to make statistical estimates about the sampling quality and to determine when enough subgraphs have been sampled).
- ESA can sample the same subgraph multiple times, spending time without gathering new information.

In the next subsection, we suggest a new approach to subgraph sampling that overcomes these problems.

2.3 The New Approach: Randomized Enumeration

The idea here is to start with an algorithm that efficiently enumerates all size-k subgraphs. This algorithm is then modified to randomly skip over some of these subgraphs during its execution, yielding an unbiased subgraph sampling algorithm.

2.3.1 Enumerating All $Size$-k Subgraphs

Given a graph $G = (V, E)$, the algorithm ESU shown in Fig. 3 enumerates all of its size-k subgraphs. The basic idea of this algorithm is that—starting with a vertex v from the input graph—we add only those vertices to the $V_{Extension}$ set that have two properties: Their label must be larger than

6. Kashtan et al. [15] observe that ESA can accurately estimate the concentration of $\mathcal{S}_k^i(G)$ with less than $(\mathcal{C}_k^i(G))^{-1}$ samples for subgraphs which are oversampled. In return, however, other subgraphs might be missed completely for far more than $(\mathcal{C}_k^i(G))^{-1}$ samples and would consistently be overlooked as motif candidates.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

Algorithm: ENUMERATESUBGRAPHS(G,k) (ESU)
Input: A graph $G=(V,E)$ and an integer $1 \le k \le |V|$.
Output: All size-k subgraphs in G.

01 **for** each vertex $v \in V$ **do**
02 $V_{Extension} \leftarrow \{u \in N(\{v\}) : u > v\}$
03 **call** EXTENDSUBGRAPH$(\{v\}, V_{Extension}, v)$
04 **return**

EXTENDSUBGRAPH$(V_{Subgraph}, V_{Extension}, v)$
E1 **if** $|V_{Subgraph}| = k$ **then output** $G[V_{Subgraph}]$ and **return**
E2 **while** $V_{Extension} \neq \emptyset$ **do**
E3 Remove an arbitrarily chosen vertex w from $V_{Extension}$
E4 $V'_{Extension} \leftarrow V_{Extension} \cup \{u \in N_{excl}(w, V_{Subgraph}) : u > v\}$
E5 **call** EXTENDSUBGRAPH$(V_{Subgraph} \cup \{w\}, V'_{Extension}, v)$
E6 **return**

Fig. 3. Pseudocode for the algorithm ESU which enumerates all size-k subgraphs in a given graph G. (The definition of the exclusive neighborhood $N_{excl}(v, V')$ is given in Section 2.1.)

that of v and they may only be neighbored to the newly added vertex w but not to a vertex already in $V_{Subgraph}$, that is, they must be in the exclusive neighborhood of w with respect to $V_{Subgraph}$. Some more insight into the structure of ESU can be gained by the following tree structure. (Note that we refer to the vertices of this structure as "nodes" in order to avoid confusion with the vertices of the input graph.)

Definition 1. *With a call to* EnumerateSubgraphs(G,k)*, we associate a tree of recursive function calls called* ESU-tree*. The root located at depth zero represents the function* EnumerateSubgraphs(G,k)*. Each call of* ExtendSubgraph$(V_{Subgraph}, V_{Extension}, v)$ *is represented by an edge from the node representing the caller function to a node representing the callee. The callee node is labeled* $(V_{Subgraph}, V_{Extension})$ *and located at depth* $|V_{Subgraph}|$.

The structure of the ESU-tree is illustrated in an example in Fig. 4. It is the basis to establish the correctness of ESU in Theorem 2. For a node w in the tree, we use $\mathrm{SUB}(w)$ and $\mathrm{EXT}(w)$ to denote the sets $V_{Subgraph}$ and $V_{Extension}$ of its label, respectively. Furthermore, it is assumed that the nodes of the ESU-tree are ordered according to the order in which the subroutines they represent are called. If a node w_1 precedes a node w_2 in this order, we designate this by writing $w_1 \prec w_2$. Given a set of tree nodes, a node is called *minimal* in this set if it precedes all other nodes in the set. The next lemma states some properties of the ESU-tree that are used for proving the correctness of ESU in Theorem 2.

Lemma 1. *The ESU-tree has the following properties:*

1. *Let w_1 be a node distinct from the root. For every vertex $u \in \mathrm{EXT}(w_1)$, the node w_1 has a child node w_2 such that $u \in \mathrm{SUB}(w_2)$.*
2. *For each node w in the ESU-tree distinct from the root and for each vertex $u \in \mathrm{EXT}(w)$, we have $u > v$, where v is the smallest-label vertex in $\mathrm{SUB}(w)$.*
3. *Let w_1 and w_2 be two nodes in the tree with a common parent node and $w_1 \prec w_2$. Then, $\mathrm{SUB}(w_1)$ contains exactly one vertex u_1 which is not contained in $\mathrm{SUB}(w_2)$ and vice versa. For every node w' whose path to the root contains w_2, we have $u_1 \notin \mathrm{SUB}(w')$.*

Proof. Property 1 follows from the fact that lines *E3* to *E5* are carried out for every vertex u in the original $V_{Extension}$ set (basically, $V_{Extension}$ can be viewed as a stack from which we pop the vertices u until it is empty).

Property 2 follows directly from lines *02* and *E4* of the algorithm, where one condition for a vertex to be added to $V_{Subgraph}$ is that its label is larger than the label of the vertex v. Hence, v, the first vertex added to $V_{Subgraph}$, is the smallest-label vertex in $\mathrm{SUB}(w)$, as claimed.

For Property 3, two cases need to be considered. In the first case, let the common parent of w_1 and w_2 be the root of the ESU-tree. Then, Property 3 holds because line *03* of ESU is executed exactly once for each vertex v of the input graph and Property 2 ensures that every node w' that is a descendant of w_2 satisfies $\mathrm{SUB}(w') \cap \mathrm{SUB}(w_1) = \emptyset$. Assume now as the second case that the common parent of w_1 and w_2 is distinct from the root. Then, the existence of the vertex u_1 as claimed becomes clear from the fact that once u_1 has been considered for addition to the $V_{Subgraph}$ set, it is removed from $V_{Extension}$ by line *E3* of the ESU algorithm. The claim that once the call to ExtendSubgraph$(V_{Subgraph} \cup \{u_1\}, V_{Extension})$ in line *E5* has been completed, no

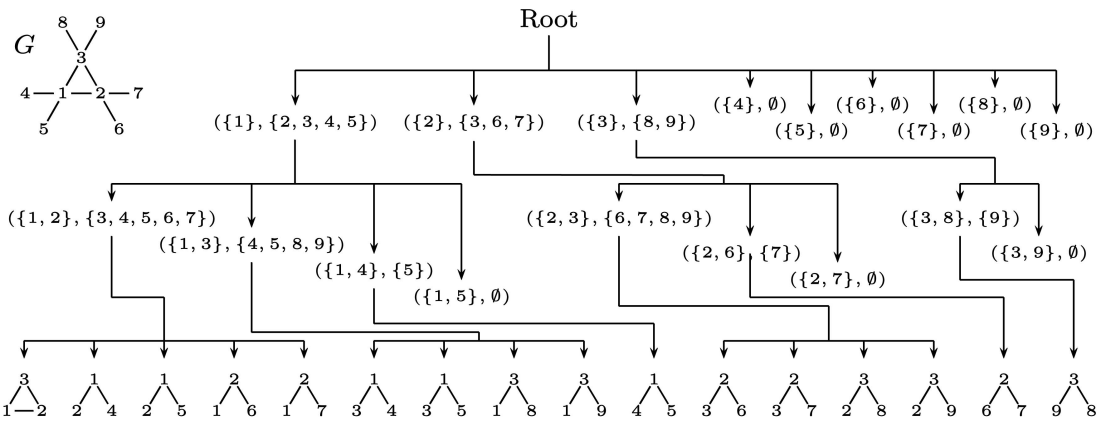


Fig. 4. Given the labeled graph in the left upperhand corner, the above ESU-tree corresponds to calling ENUMERATESUBGRAPHS$(G,3)$. The tree has 16 leaves which correspond to the 16 size-3 subgraphs of G.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

subgraph containing u_1 is output until we reach line *E6* again is proved by observing that u_1 must be neighbor to some vertex in $V_{Subgraph}$ since it is in $V_{Extension}$. Then, however, there exists no vertex $u' \in V$ for which $u_1 \in N_{excl}(u', V_{Subgraph})$ and, hence, once u_1 is removed from $V_{Extension}$, it is not added to this set by any recursive call of EXTENDSUBGRAPH until we reach line *E6* again. □

Theorem 2. *Given a graph G and $k \geq 2$,* ESU *enumerates all size-k subgraphs in G. Each size-k subgraph is output exactly once.*

Proof. Given a graph G and an integer $k \geq 2$, we show that every size-k subgraph in G is output at least once and at most once.

"At least once." Calling $\mathrm{EnumerateSubgraphs}\ (G, k)$, let T be the corresponding ESU-tree. Assume for the purpose of contradiction that there exists a size-k subgraph G' with vertex set $\{v_1, \ldots, v_k\}$ in G that is not output by ESU. Without loss of generality, we assume v_1 to be the smallest-label vertex in G'. Because line *03* of the ESU algorithm is called for every vertex $v \in V(G)$ including v_1, the root of T has exactly one child node w_1 with $\mathrm{SUB}(w_1) = \{v_1\}$. All neighbors of v_1 in G' are in $\mathrm{EXT}(w_1)$ (by line *02* of ESU considering the assumption that v_1 is the smallest-label vertex). Then, by Property 1 of the ESU-tree in Lemma 1, w_1 has a child node w_2 with $\mathrm{SUB}(w_2) = \{v_1, v'\}$ for each neighbor v' of v_1 in G'. Let w_2' be the minimal of these child nodes and assume without loss of generality that the respective neighbor of v_1 added to $\mathrm{SUB}(w_1)$ is v_2, that is, $\mathrm{SUB}(w_2') = \{v_1, v_2\}$. We now claim that $\mathrm{EXT}(w_2')$ contains all neighbors that v_1 and v_2 have in G': Assume for the purpose of contradiction that there exists a neighbor which is not contained in $\mathrm{EXT}(w_2')$. This could only be for three reasons, all of which we can rule out, yielding the desired contradiction:

1. Its label could be smaller than that of v_1 (which can be ruled out because v_1 is the smallest-label vertex in G').
2. It could be neither a neighbor of v_1 nor in the exclusive neighborhood of v_2 (which can be ruled out because, in G', it is a neighbor of either v_1, v_2, or both).
3. It could already have been taken from $\mathrm{EXT}(w_2')$ (which can be ruled out because we assumed w_2' to be minimal).

Inductively carrying out the above argument for the vertices $v_3, \ldots, v_k$ leads to a leaf node w_k in the ESU-tree for which $\mathrm{SUB}(w_k) = V(G')$, a contradiction to our assumption that G' is not output by the algorithm.

"At most once." Assume, for the purpose of contradiction, that a subgraph G' is enumerated twice. This means that there are two leaves, w_1 and w_2, in the corresponding ESU-tree for which $\mathrm{SUB}(w_1) = \mathrm{SUB}(w_2)$. The path p_1 from w_1 to the root must differ at least partly from the path p_2 from w_2 to the root. Call the greatest-depth node in the tree that p_1 and p_2 share the *split node*. Due to Property 3 of Lemma 1, the existence of the split node implies that $\mathrm{SUB}(w_1)$ and $\mathrm{SUB}(w_2)$ differ by at least one element, a contradiction. □

Besides being useful for the above correctness proof, the ESU-tree exposes some additional useful properties. For example, we can quickly estimate the total number of size-k subgraphs in the input graph using a technique by Knuth [16] that randomly explores paths in the ESU-tree. With this estimate at hand, it is, e.g., possible to see if a total enumeration of subgraphs is expected to be feasible, to estimate the running time of the ESU algorithm (for example, in order to implement a progress indicator [9]) and to make statistical error estimates about the sampling error if no exact enumeration seems feasible. Probably the most important feature of the ESU-tree, however, is that we can use it to efficiently sample subgraphs uniformly at random (that is, without bias). This is further explored in the next subsection.

2.3.2 Uniformly Sampling $Size$-k Subgraphs.

The ESU algorithm completely traverses its corresponding ESU-tree. Where complete traversal is too time-expensive, we can explore only parts of the ESU-tree such that each leaf is reached with equal probability. For this purpose, a probability $0 < p_d \leq 1$ is introduced for each depth $1 \leq d \leq k$ in the tree. Using p_d, we determine for each child vertex at depth d whether we traverse the subtree rooted at it. This is implemented by replacing lines *03* and *E5* of the ESU algorithm with

"With probability p_d, call EXTENDSUBGRAPH(...),"

where $d \stackrel{\mathrm{def}}{=} 1$ in line *03* and $d \stackrel{\mathrm{def}}{=} |V_{Subgraph}| + 1$ in line *E5*. We call this new algorithm RAND-ESU. To simplify the discussion, we will also use this name when all p_d are set to 1, in which case, RAND-ESU is equivalent to ESU. As we now show, RAND-ESU visits each leaf of the ESU-tree with equal probability and, hence, estimating subgraph concentrations from its output is straightforward.

Lemma 3. RAND-ESU *visits each leaf in the* ESU-*tree with probability $\prod_d p_d$.*

Proof. Let w_k be a leaf node in the ESU-tree and $w_{k-1}, \ldots, w_1$ the nodes that lie on the path from w_k to the root. Then, we have

$$\begin{aligned}\Pr[w_k \text{ is reached}] &= \Pr[w_k \text{ reached} \mid w_{k-1} \text{ reached}] \\ &\quad \cdot \Pr[w_{k-1} \text{ reached}] \\ &= p_k \cdot \Pr[w_{k-1} \text{ reached}] \\ &= p_k \cdot p_{k-1} \cdot \Pr[w_{k-2} \text{ reached}] \\ &= \cdots = p_k \cdot p_{k-1} \cdot \ldots \cdot p_1 = \prod_{1 \leq d \leq k} p_d.\end{aligned}$$

□

Proposition 4. *Given a graph G, an integer k, and $0 < p_d \leq 1$ for $1 \leq d \leq k$, let $\mathcal{R}$ be a set of size-k subgraphs obtained by running* RAND-ESU *on G using the probabilities p_d. Then,*

$$\hat{C}_k^i(\mathcal{R}, G) \stackrel{\mathrm{def}}{=} \frac{|\{G' \in \mathcal{R} \,:\, G' \in \mathcal{S}_k^i(G)\}|}{|\mathcal{R}|}$$

is an unbiased estimator for $C_k^i(G)$.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

Proof. The proof follows directly from Lemma 3: If the input graph contains exactly N subgraphs and N' of these are representatives of a subgraph class $\mathcal{S}_i^k$, the fraction of leaves in the ESU-tree that correspond to representatives of $\mathcal{S}_i^k$ equals N'/N. Since each leaf in the ESU-tree is reached with equal probability, the expected fraction of subgraphs in $\mathcal{R}$ that correspond to representatives of $\mathcal{S}_i^k$ is precisely $N'/N = \mathcal{C}_k^i(G)$. □

It remains to discuss how the values p_d should be chosen. If we wish to sample an expected fraction $0 < q < 1$ of all size-k subgraphs using RAND-ESU, we clearly have to ensure that $\prod_{1\le d\le k} p_d = q$. However, this still leaves us to choose the individual values, that is, do we uniformly set every p_d equal to $q^{1/k}$ or are there better choices? Some general observations are:

- Choosing whether or not to explore a subtree whose root is close to the root of the ESU-tree generally has a higher influence on the total number of explored leaves than for a subtree whose root is farther from it.
- The parameters p_d influence the distribution of the sampling, that is, if p_d is small for small d, some local neighborhoods in the input graph are likely not to be explored at all while others will be explored extensively.
- The running time is influenced from an amortized point of view: The larger the p_d values are for small values of d, the more of the ESU-tree is explored in order to sample a certain expected number of leaves.

Further analysis concerning the choice of sampling parameters p_d can be made based on generating functions [11], [34]: Assume we have two independent random variables X and Y and determine a random variable Z by first choosing a nonnegative integer n according to the distribution of X and then building the sum of n independent random variables chosen according to the distribution of Y. Then, as shown in [11, p. 577], the expected value of Z is

$$\mathbb{E}(Z) = \mathbb{E}(X)\mathbb{E}(Y) \tag{2}$$

with a variance of

$$\mathrm{Var}(Z) = \mathrm{Var}(X)(\mathbb{E}(Y))^2 + \mathbb{E}(X)\mathrm{Var}(Y). \tag{3}$$

Consider a random tree T_k of height k where independently for each node at depth d its number of children is a random variable X_d with expected value $\mathbb{E}_d := \mathbb{E}(X_d)$ and variance $\mathrm{Var}_d := \mathrm{Var}(X_d)$.[7] For a random traversal of T_k with given parameters p_d, iterative application of (2) and (3) shows that the expected number of visited leaves in T_k is $\mathbb{E}_{T_k} = \prod_{1\le d\le k}(p_d \cdot \mathbb{E}_d)$ (as was to be expected from Lemma 3) with a variance of

$$\mathrm{Var}_{T_k} = \sum_{1\le d\le k}\left(\left(\prod_{1\le i<d} p_j\cdot\mathbb{E}_j\right)\cdot\left(p_j\cdot\mathrm{Var}_j + (p_j - p_j^2)\cdot\mathbb{E}_j\right)\cdot\left(\prod_{d<i\le k}(p_j\cdot\mathbb{E}_j)^2\right)\right). \tag{4}$$

Note how, in accordance with our discussion above, Var_{T_k} is reduced if p_d is small for larger values of d. The term "$(p_j - p_j^2)\cdot\mathbb{E}_j$" found in the middle of (4) is the variance for the number of child nodes that we choose to explore further, its origin lies in lines *03* and *E5* of RAND-ESU, which give rise to a binomial distribution for the number of children that are explored. We can reduce this variance as follows: Let w_d be a node in the tree at depth d with x children. Then, instead of deciding independently with probability p_d for each of the x children whether it is to be explored further, we randomly choose x' of the x children, where

$$x' = \begin{cases} \lceil x\cdot p_d\rceil & \text{with probability } (x\cdot p_d - \lfloor x\cdot p_d\rfloor)\\ \lfloor x\cdot p_d\rfloor & \text{with probability } (1-(x\cdot p_d - \lfloor x\cdot p_d\rfloor))\end{cases}$$

and explore exactly these.[8] This new procedure has the advantage that it does not change the probability of an individual child being explored ($\Pr[w_{d+1}$ is reached $\mid w_d$ is reached$] = p_d$ also in this model) while at the same time the variance is reduced from $\mathbb{E}_j\cdot(p_j - p_j^2)$ to at most

$$\max\{(x\cdot p_d - \lfloor x\cdot p_d\rfloor)^2, (\lceil x\cdot p_d\rceil - x\cdot p_d)^2\} < 1.$$

Further analysis will be required to see if any other improvements can be derived for the algorithm. For example, considering Property 2 of Lemma 1, one might consider reducing Var_j in (4) for the ESU algorithm by a certain labeling of the vertices in the input graph.

As a general rule from our analysis in this section, the parameters p_d should be larger for small d and become smaller as d increases—as long as the sacrifice made with respect to the amortized running time per sample is acceptable. This ensures a lower variance for the number of samples and the exploration of many different regions in the input graph.

Concluding this section, while RAND-ESU—as compared to ESA—requires a choice of sampling parameters and only allows for controlling the expected number of samples, it has a lot to offer in return. Most importantly, it is unbiased, which rules out the respective disadvantages of ESA. Also, it is much faster (see the experiments in Section 4) and easier to implement since we do not require any bias-correcting parts. Contrary to ESA, our new algorithm never samples more subgraphs than the input graph contains and its results become exact as the number of samples reaches the total number of size-k subgraphs in the input graph.

7. We are aware that this is only a very coarse model for the ESU-tree, but it allows us to emphasize the main points we wish to make here while it appears that a more precise model does not generate significant additional insight.

8. The idea is to always explore at least $\lfloor x\cdot p_d\rfloor$ children. If $x\cdot p_d$ is not an integer, one more child than $\lfloor x\cdot p_d\rfloor$ is explored with a certain probability, ensuring that exactly $x\cdot p_d$ children are reached on average.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

3 Direct Calculation of Motif Significance

As already mentioned in the introduction, network motif detection spends considerable amounts of time for the subtask of determining subgraph significance. Traditionally, this task involves explicitly generating an ensemble of random graphs under a certain random graph model and then determining their subgraph concentrations. Here, we propose a new approach to determining subgraph significance that does not need an explicit random graph generation and offers some additional advantages, such as being able to focus the estimation of significance on specific subgraphs.

3.1 A New Approach to Calculating Subgraph Concentrations

We consider the case where the significance of a subgraph is determined by comparing its concentration in the given graph G to its mean concentration $\langle \mathcal{C}_k^i(G) \rangle$ in random graphs with the same degree sequence [15], [24]. It is suggested in [15], [24] to estimate $\langle \mathcal{C}_k^i(G) \rangle$ by generating a large ensemble of random graphs (typically at least 1,000) with the same degree sequence as the original graph and then sampling subgraphs in these random graphs. We will call this approach EXPLICIT.

The random graphs in EXPLICIT are generated from the original graph by randomly switching endpoints between graph edges. This requires a lot of switching operations while, at the same time, it is never certain when proper randomization has been reached. Also, with this method, we are likely to spend lots of computational effort in estimating the concentrations of subgraph classes we are not interested in.[9] In this subsection, we propose a new algorithm DIRECT for determining subgraph significance without the need to explicitly generate random graphs. This approach is, moreover, able to focus the estimation of concentrations on specific subgraphs.

Explicitly generating a random network and then estimating its subgraph concentrations can be seen as a random experiment where we first choose a random graph with the same degree sequence as the original graph and then randomly choose subsets of k vertices that induce a connected subgraph. Milo et al. observe that the total number of size-k subgraphs within an ensemble of large graphs with the same degree sequence does not vary much (see supplementary online material to [23] for details). Hence, we could estimate $\langle \mathcal{C}_k^i(G) \rangle$ by a differently ordered random experiment where we first select a subset of k vertices and then determine the ratio of graphs with the same degree sequence where these vertices induce a subgraph from $\mathcal{S}_k^i(G)$. More precisely, we have

$$\langle \mathcal{C}_k^i(G) \rangle \approx \langle \hat{\mathcal{C}}_k^i(G) \rangle \stackrel{\text{def}}{=} \frac{\sum_{G' \in \mathrm{SEQ}(G)} |\mathcal{S}_k^i(G')|}{\sum_{G' \in \mathrm{SEQ}(G)} \sum_i |\mathcal{S}_k^i(G')|}, \qquad (5)$$

where $\mathrm{SEQ}(G)$ is the set of all graphs G' that have the same degree sequence as G. Since all graphs G' can be viewed as graphs over the same set of vertices (because they differ only in their edge sets), the right part of (5) can also be written as

$$\langle \hat{\mathcal{C}}_k^i(G) \rangle = \frac{\sum_{\{v_1,\ldots,v_k\} \subseteq V} |\{G' \in \mathrm{SEQ}(G) : G'[\{v_1,\ldots,v_k\}] \in \mathcal{S}_k^i\}|}{\sum_{\{v_1,\ldots,v_k\} \subseteq V} |\{G' \in \mathrm{SEQ}(G) : G'[\{v_1,\ldots,v_k\}] \text{ is connected}\}|}. \qquad (6)$$

Both the nominator and denominator of this equation can be estimated in a Monte Carlo approach—that is, by randomly sampling size-k subsets of the vertices in the input graph—as long as we are able to perform the following calculation: Given G and $\{v_1, \ldots, v_k\}$, find $|\{G' \in \mathrm{SEQ}(G) : G'[\{v_1, \ldots, v_k\}] \in \mathcal{S}_k^i\}|$. As will be discussed in the next subsection, this number can indeed be calculated. We refer to the resulting algorithm as DIRECT throughout the remainder of this work.

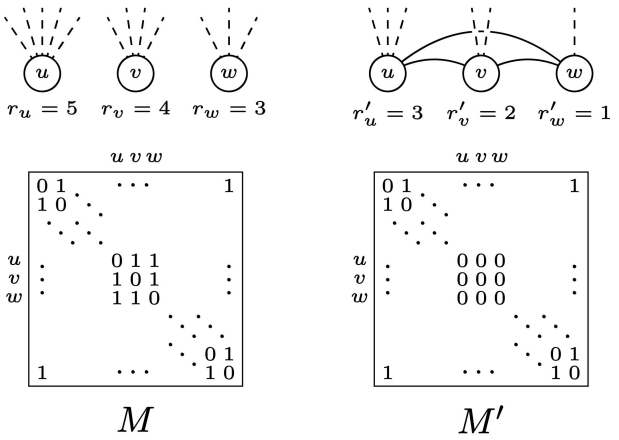


Fig. 5. Theorem 5 allows us to count the number of graphs with a given degree sequence under the constraint that certain edges must be present. On the left, the bitmask matrix M causes every graph with a given degree sequence to be counted by the equation in Theorem 5 since no edge is forbidden by a zero-entry. On the right, we "force" the subgraph to be induced by u, v, and w in all counted graphs by decreasing the degrees of these vertices by two each (e.g., $r_u = 5$ becomes $r'_u = 3$) and using the bitmask matrix M' to avoid the forced edges to be counted.

3.2 The Number of Graphs that Induce a Fixed Subgraph

In 1974, Bender [6] proved a powerful theorem that allows for an asymptotic estimation of the number of directed graphs with a prescribed degree sequence. This was later extended to a theorem about undirected graphs together with Canfield [7]. The power behind both theorems lies in the fact that we can not only estimate the number of graphs with a prescribed degree sequence, but also, using a *bitmask-matrix* M, specify pairs of vertices that are not to be connected in the graphs we are counting. We describe this in more detail following a formal (and alas quite technical) recapitulation of Bender and Canfield's theorems.

Definition 2. *Given functions $f, g, h : \Omega \to \mathbb{R}$, we say "$g \sim h$ uniformly as $f \to \infty$" if*

$$\lim_{k \to \infty} \sup_{\{\omega \in \Omega \,:\, f(\omega) = k\}} \left| \frac{g(\omega)}{h(\omega)} - 1 \right| = 0.$$

9. This is especially important for sparse networks where a randomly sampled subgraph is likely to be a tree. Trees, however, are often considered to be uninteresting motifs [8].

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Number of Size-k Subgraphs and the Number of Respective Subgraph Classes in Our Test Instances for $3 \leq k \leq 6$

		COLI	YEAST	ELEGANS	YTHAN
number of nodes		423	688	306	135
number of edges		519	1 079	2 345	597
average node degree		1.2	1.6	7.7	4.4
subgraphs	size-3	5 206	13 150	47 322	9 487
	size-4	83 893	183 174	1 394 259	169 733
	size-5	1 433 502	2 508 149	43 256 069	2 908 118
	size-6	22 532 584	32 883 898	1 309 307 357	45 889 039
subgraph classes	size-3	4	7	13	8
	size-4	17	33	197	57
	size-5	83	173	7 071	629
	size-6	390	888	286 375	9 339

All instances are directed graphs.

Theorem 5 (For undirected graphs, adapted from [7]). *Let $M = (m_{ij})$ be a binary symmetric $n \times n$-matrix, where $m_{ii} = 0$ for all i and the number of zeros per row is bounded by a constant. Given a length-n vector $r = (r_1, \ldots r_n)$ over $\{0, 1, \ldots, d\}$, let $G(M, r)$ be the number of binary symmetric $n \times n$-matrices (g_{ij}) that satisfy $m_{ij} = 0 \Rightarrow g_{ij} = 0$ and $\sum_j g_{ij} = r_i$. Then,*

$$G(M, r) \sim \frac{\sqrt{2}(f/e)^{(f/2)}}{\exp(a^2 + a + b) \cdot \prod_i (r_i!)}$$

uniformly as $f \to \infty$, where $a = \sum_i \frac{r_i^2 - r_i}{2f}$, $b = \sum_{m_{ij}=0, i<j} \frac{r_i r_j}{f}$, and $f = \sum_i r_i$. □

Theorem 6 (For directed graphs, main theorem in [6]). *Let $M = (m_{ij})$ be a binary symmetric $n \times n$-matrix, where $m_{ii} = 0$ for all i and the number of zeros per row is bounded by a constant. Given two length-n vectors $r = (r_1, \ldots r_n)$ and $c = (c_1, \ldots c_n)$ over $\{0, 1, \ldots, d\}$, let $G(M, r, c)$ be the number of binary symmetric $n \times n$-matrices (g_{ij}) that satisfy $m_{ij} = 0 \Rightarrow g_{ij} = 0$, $\sum_j g_{ij} = r_i$, and $\sum_i g_{ij} = c_i$. Then,*

$$G(M, r, c) \sim \frac{f!}{\exp(a + b) \cdot \prod_i (r_i!) \cdot \prod_j (c_j!)}$$

uniformly as $f \to \infty$ where $a = \frac{(\sum_i r_i^2 - r_i) \cdot (\sum_j c_j^2 - c_j)}{2f^2}$, $b = \sum_{m_{ij}=0} \frac{r_i c_j}{f}$, and $f = \sum_i r_i = \sum_j c_j$. □

Theorems 5 and 6 are both concerned with binary matrices; it is easy to see that they directly relate to the number of adjacency matrices of graphs with a certain given degree sequence. Furthermore, as we now show in more detail, these theorems can also be used to count the total number of graphs with a given degree sequence under the condition that a certain subgraph is fixed. To see this for Theorem 5, observe that we can "forbid" an edge between two vertices v_i and v_j in the counted graphs by setting the corresponding entries m_{ij} and m_{ji} in the bitmask matrix M to zero. We can also "force" an edge between two vertices v_i and v_j by forbidding it via the bitmask matrix M and additionally decreasing the values of r_i and r_j by one each. In this way, Theorem 5 allows us to calculate, for a given degree sequence, how many graphs there are which realize exactly this degree sequence under the constraint that a certain subgraph is fixed. This is illustrated in Fig. 5 and works similarly for directed graphs using Theorem 6. Given a subgraph class $\mathcal{S}_k^i$ and k vertices $\{v_1, \ldots, v_k\}$, we can thus consider all at most $k!$ ways in which these vertices can induce a subgraph from $\mathcal{S}_k^i$ in order to calculate the nominator in (6).

To estimate the denominator in (6), there exist several possibilities. On the one hand, if, for a fixed k, we have calculated the nominator for all subgraph classes $\mathcal{S}_k^i$, then the denominator is simply the sum of all these values. On the other hand, if we have calculated the nominator of (6) for only a few subgraph classes, then a different approach is possible, the key idea of which is that we do not have to *explicitly* fix the subgraph between the vertices $\{v_1, \ldots, v_k\}$, but only ensure that they are connected. This can be achieved by forcing a spanning tree between the vertices and ensuring—through forbidding certain edges—that no connected subgraph between $\{v_1, \ldots, v_k\}$ is considered

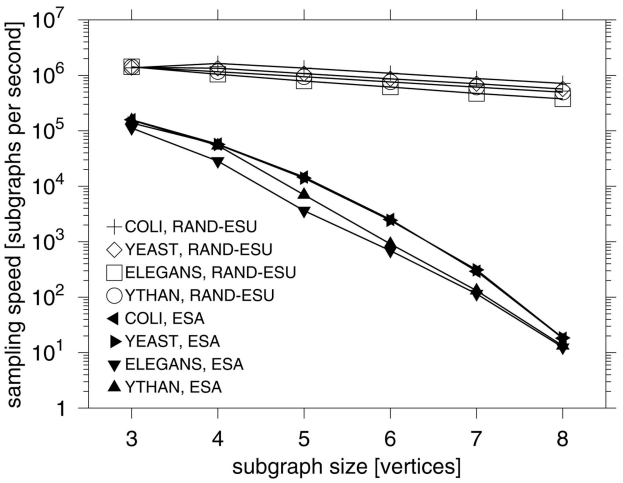


Fig. 6. Sampling speed for different subgraph sizes on a semi-log scale. Independently of the network, ESU (represented by the top four curves) is much faster than ESA and scales much better as the subgraph size increases. Details as to the exact experimental setting are given in the text.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

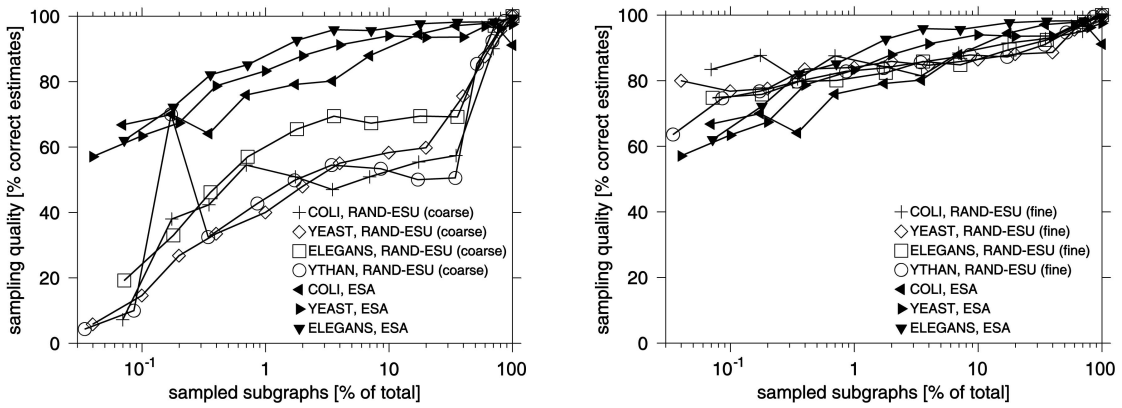


Fig. 7. Sampling quality for size-5 subgraphs (size-4 for YTHAN) versus the percentage of sampled subgraphs (semi-log scale). For our YTHAN instance, the *mfinder* tool reproducibly failed to report results for more than 100 samples, hence this curve is not shown. (a) shows the results for ESA versus RAND-ESU using the sampling parameter setting "coarse" and (b) shows the results using the sampling parameter setting "fine." Further details as to the experimental setting are given in the text.

twice. To avoid double-counting of certain subgraphs, we make use of the following straightforward observation:

Observation 7. *Every graph with pairwise distinct edge weights has a unique minimum spanning tree.*

Assume that we assign each edge $\{v_i, v_j\}$ (where, without loss of generality, $i < j$) a weight of $i + k \cdot j$. Then, every potential edge in the graph has a unique weight. Hence, we require the enumeration of k^{k-2} trees to estimate the denominator of (6) for undirected graphs, assuming each of these trees to be the minimum spanning tree of the subgraph that is induced by the vertices $\{v_1, \ldots, v_k\}$ and forbidding exactly those edges that would contradict this minimality.

For the directed case, the argument is similar to the above, except that, for each spanning tree, we have to consider all ways of coloring the $k-1$ edges with two colors, one color meaning the "directed edge is forced from the vertex with the lower label to the vertex with the higher label" and the other one meaning "the directed edge is forced from the vertex with the higher label to the vertex with the lower label *and* the directed edge is forbidden from the vertex with the lower label to the vertex with the higher label." All in all, this requires the consideration of $2^{k-1} \cdot k^{k-2} = 2 \cdot (2k)^{k-2}$ such trees.

At first glance, it might seem as if our new approach to estimating subgraph significance is prohibitively expensive to calculate, but it actually promises a gain in efficiency for a number of reasons:

- If a certain subgraph seldom appears and in relatively few graphs that realize a given degree sequence, a huge number of random graphs have to be generated in order to obtain a sufficient number of samples. In contrast to this, DIRECT always yields a subgraph concentration for any set of vertices where a given subgraph can potentially be induced.
- For small k, we can store all nonautomorph isomorphisms of the respective subgraph (typically far less than the worst-case $k!$) in an index structure and do not have to recalculate them every time.
- The denominator in (6) is the same for all subgraph classes and, hence, has to be calculated only once. Moreover, it is conceivable that deeper mathematical analysis can estimate this number simply from the degree sequence of the input graph. Even if the denominator is not calculated, we have a *ratio* of subgraph concentrations by the various nominators. Comparing this ratio to the ratio in the original network might already suffice to show the significance of a subgraph as a motif.
- The number of occurring subgraph classes is often far less than the total number of subgraphs (see Table 1) and we can focus our analysis directly on them.

Experiments which are discussed in the next section confirm this expected performance gain.

4 Experimental Studies

We have implemented our algorithms from Sections 2 and 3 in C++. The source code is freely obtainable online at http://theinf1.informatik.uni-jena.de/motifs/.[10] As a comparison, we used the *mfinder* 1.1 tool[11] by Kashtan et al., which implements the ESA algorithm.

4.1 Method and Results

All tests were performed on an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory running under the Debian GNU/Linux 3.1 operating system. Sources were compiled with the GNU gcc/g++ 3.3.4 compiler using the option "-O3."

The network instances for testing the algorithms were up-to-date versions of the motif detection testbed used by Kashtan et al. [15]. The testbed consists of the instances

10. On the same Web site, we also offer a user-friendly RAND-ESU-based motif detection tool, see [33] for details.

11. Source at http://www.weizmann.ac.il/mcb/UriAlon/groupNetworkMotifSW.html.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Approximate Size-3 Subgraph Concentrations in Random Graphs
Based on the Methods Discussed in Section 3

	COLI		YEAST		ELEGANS		YTHAN	
	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$	$\langle \mathcal{C}_k^i(G) \rangle$	$\langle \hat{\mathcal{C}}_k^i(G) \rangle$
	9.12 E−1	9.07 E−1	9.00 E−1	8.99 E−1	2.00 E−1	2.00 E−1	4.15 E−1	3.72 E−1
	4.95 E−2	5.11 E−2	7.13 E−2	7.07 E−2	3.65 E−1	3.74 E−1	2.16 E−1	2.28 E−1
	3.65 E−2	4.05 E−2	2.79 E−2	2.93 E−2	3.17 E−1	3.23 E−1	2.17 E−1	2.42 E−1
	2.83 E−4	2.22 E−4	1.43 E−4	1.03 E−4	3.44 E−2	2.74 E−2	4.02 E−2	3.65 E−2
	4.36 E−5	3.20 E−5	1.68 E−5	1.20 E−5	3.69 E−2	2.94 E−2	4.35 E−2	4.66 E−2
	1.44 E−7	7.38 E−8	3.31 E−8	1.72 E−8	2.49 E−3	1.58 E−3	4.05 E−3	3.67 E−3
	1.44 E−3	1.33 E−3	1.09 E−3	1.06 E−3	3.23 E−2	3.45 E−2	4.79 E−2	5.29 E−2
	2.90 E−6	3.33 E−6	9.46 E−7	9.94 E−7	4.08 E−3	4.45 E−3	1.71 E−3	2.59 E−3
	2.11 E−6	1.60 E−6	1.34 E−6	9.52 E−7	2.13 E−3	1.79 E−3	3.47 E−3	3.36 E−3
	7.60 E−7	4.59 E−7	1.95 E−7	1.34 E−7	1.65 E−3	1.43 E−3	6.04 E−3	6.73 E−3
	1.98 E−7	1.50 E−7	5.38 E−8	3.71 E−8	2.41 E−3	2.07 E−3	3.21 E−3	3.94 E−3
	3.82 E−9	1.72 E−9	6.80 E−10	3.26 E−10	5.78 E−4	4.12 E−4	1.59 E−3	1.58 E−3
	—	1.73 E−12	—	1.96 E−13	2.87 E−5	1.68 E−5	9.76 E−5	7.74 E−5

The concentrations $\langle C_i^k(G) \rangle$ were determined with EXPLICIT by generating 10 million random graphs with the same degree sequence as the original network (observe that, even with this large number of random graphs, the subgraph shown in the bottom-most row of the table was never found in COLI and YEAST although the degree sequences of both networks theoretically allow for its presence). The concentrations $\langle C_i^k(G) \rangle$ were calculated by DIRECT, evaluating (6) from Section 3.2 for 100 million random size-3 vertex subsets.

COLI (transcriptional network of *Escherichia coli* [28]), YEAST (transcriptional network of *Saccharomyces cerevisiae* [24]), ELEGANS (neuronal network of *Caenorhabditis elegans* [15]), and YTHAN (food web of the Ythan estuary [35]). Some properties of these networks are summarized in Table 1.

In order to compare RAND-ESU with ESA for speed, we measured the sampling speed of both algorithms on the testbed instances for $3 \leq k \leq 8$. Since the relative sampling speed of RAND-ESU depends on the sampling parameters (recall the discussion in Section 2.3.2), we determined the speed of RAND-ESU to be its mean speed for three different settings of $(p_1, \ldots, p_k)$ that lead to sampling of an expected 10 percent of all subgraphs, namely, $(1, \ldots, 1, .316, .316)$, $(1, \ldots, 1, .5, .2)$, and $(1, \ldots, 1, .1)$. The results are shown in Fig. 6. The speed of the deterministic ESU algorithm is not shown in the figure; it proved to be slightly faster than that of RAND-ESU. Note that, in order to get comparable results, our time measurements do not include the grouping of sampled subgraphs into nonisomorphic classes because *mfinder* uses a much less efficient canonical labeling routine than the *nauty* algorithm employed in our implementation.

To compare RAND-ESU with ESA for sampling quality, we formally define the sampling quality to be the percentage of subgraph classes $\mathcal{S}_k^i$ for which $\mathcal{C}_k^i$ is estimated with at most 20 percent relative error. In doing so, for a given number of subgraph samples, we consider only those subgraph classes for the quality measurement that we would expect to sample at least 10 times. The reason for this is that, in a real-case scenario, one would not want to rely on too few sampled subgraphs to estimate the overall concentration of the respective class. As in the speed measurements, RAND-ESU was run with different settings of the sampling parameters $(p_1, \ldots, p_k)$ in order to sample an expected percentage p of subgraphs in the input network. We refer to these settings as "coarse" $(1, \ldots, 1, \sqrt{p}, \sqrt{p})$ and "fine" $(1, \ldots, 1, p)$.[12] The obtained results are shown in Fig. 7.

As to the direct calculation of subgraph significance introduced in Section 3, two sets of experiments were carried out for the four testbed instances. The first set measures how close the subgraph concentrations determined by EXPLICIT are to those calculated by DIRECT. For this, we calculated the subgraph concentration of all 13 nonisomorphic directed size-3 subgraphs both by explicitly generating 10 million random graphs as well as by sampling 100 million size-3 subsets of vertices. (We used the rather small value of $k = 3$ in order to make such a large number of samples feasible and because this enables us to gain an overview over *all* subgraph classes.) The results are given in Table 2.

To compare the speed of DIRECT with EXPLICIT, we measured the standard deviation of the output subgraph concentrations with respect to time.[13] Since both algorithms converge to slightly different values—as shown in Table 2—the standard deviations have been made comparable by normalizing them to $\langle \mathcal{C}_k^i(G) \rangle$ and $\langle \hat{\mathcal{C}}_k^i(G) \rangle$ for DIRECT and EXPLICIT, respectively. A representative subset of the obtained results is given in Fig. 8; they are further discussed in the next section.

12. Note how we also used these settings in the speed measurements.

13. Measuring the standard deviation with respect to some machine-independent variable such as "number of samples" does not seem feasible due to the very different nature of the two algorithms.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

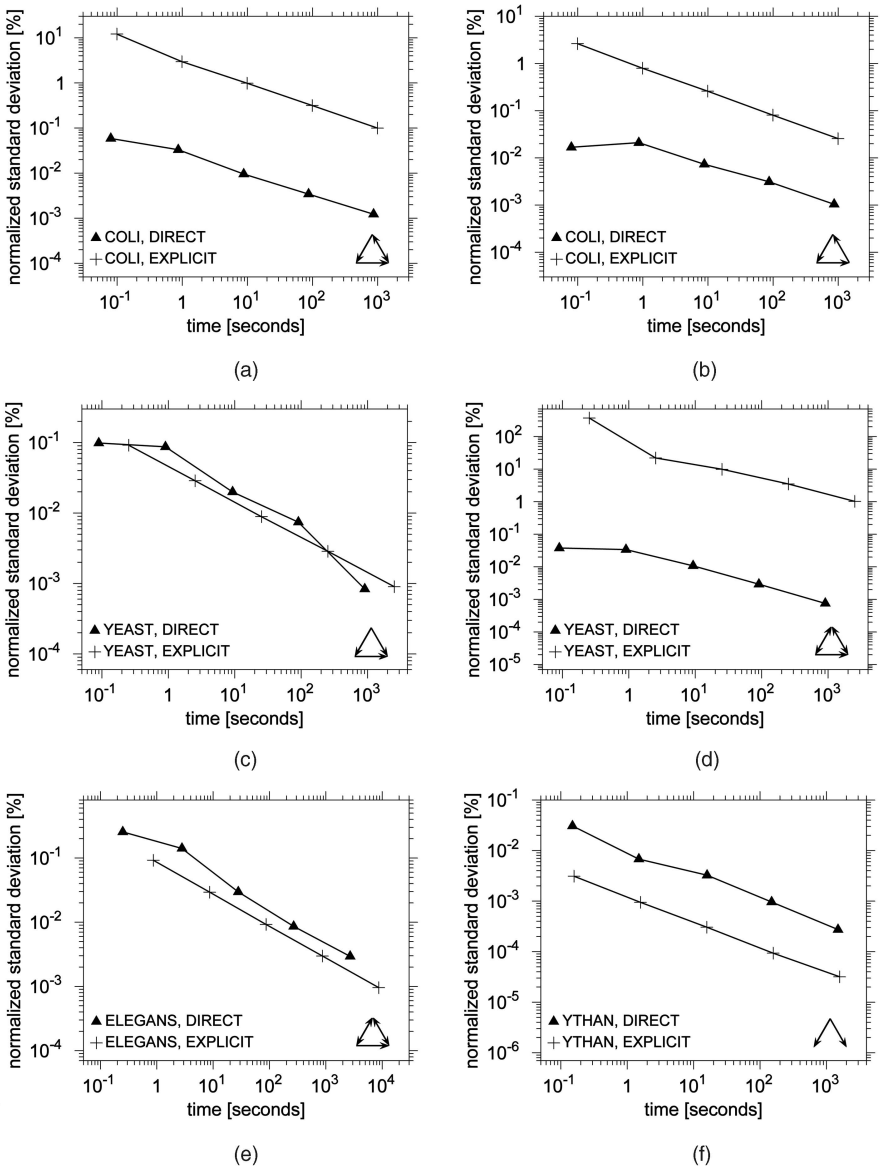


Fig. 8. Representative examples for the speed comparison of DIRECT with EXPLICIT. The name of the network is given in the lower lefthand corner, the subgraph class in the lower righthand corner of each graph. Details as to the experimental setting are given in the text.

4.2 Discussion

Most notable in Fig. 6, RAND-ESU is much faster than the ESA sampling in *mfinder*. This amounts to several orders of magnitude for larger subgraphs ($k \geq 5$). For small sampling quantities, the "coarse" variant of RAND-ESU proved to be faster than the "fine" variant (not explicitly shown in Fig. 6). However, Fig. 7a shows that the resulting sampling quality from using "coarse" settings for the p_d values is relatively low when compared to that of ESA. The qualities are roughly equal for the "fine" variant (see Fig. 7b) with ESA having a slight advantage for sampling sizes above 1 percent and close to 100 percent. (Note that, for 100 percent, RAND-ESU is equivalent to ESU and the results are exact.) Two things are to be noted in this respect, though: First, RAND-ESU is much faster and can, e.g., fully enumerate all size-5 subgraphs in roughly the same time that ESA needs to sample 1 percent of them. Second, the sampling quality of the "fine" variant appears to be more consistent for different networks, e.g., in some percentage ranges, ESA has a very good sampling quality for ELEGANS and a comparably fair one for COLI, whereas the "fine" RAND-ESU remains much more consistent here. Also note that—contrary to ESA—statistical estimates about the achieved sampling quality can be made with RAND-ESU because of its unbiasedness (especially with the "fine" variant where individual samples are fully independent of each other) and the ability to estimate the total number of subgraphs. Contrary to ESA, the sampling quality of RAND-ESU becomes perfect as the percentage of sampled subgraphs reaches 100 percent.

As to the estimation of subgraph significance, Table 2 shows that DIRECT yields subgraph concentrations that are mostly very close to those output by EXPLICIT. These concentrations generally appear to be closer for the denser

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

instances ELEGANS and YTHAN than for the sparse instances COLI and YEAST with most of the significant deviations occurring for subgraph classes that appear in very low concentrations of $\langle C_k^i(G)\rangle < 10^{-6}$ (observe from Table 1 that this is far less than one over the total number of size-3 subgraphs in these networks). Overall, the direct calculation of subgraph significance thus does not seem to have an impact on which subgraphs in the given network would be classified as motifs, making DIRECT a valid tool for determining subgraph significance.

As to the convergence speed of EXPLICIT and DIRECT, our experiments show, on the one hand, that EXPLICIT is generally faster than DIRECT for subgraphs which occur in high concentrations. Examples for this are given in Fig. 8c, Fig. 8e, and Fig. 8f. This particular speed advantage of EXPLICIT does not appear to be relevant for practical purposes, however, because both algorithms quickly reach a very low normalized standard deviation ($< 1\%$). On the other hand, as is exemplarily illustrated in Fig. 8a, Fig. 8b, and Fig. 8d, DIRECT is considerably more efficient than EXPLICIT for subgraphs which have a very low concentration in random graphs. Here, an acceptable standard deviation is achieved many orders of magnitude faster. With our new approach, it is even possible to estimate the frequency of some subgraphs for which EXPLICIT does not give *any* results due to an extremely low average concentration in the explicitly generated random graphs (two examples for this are given in the bottommost row of Table 2). Summarizing, it seems justified to say that DIRECT provides an accurate and much faster alternative to EXPLICIT for determining subgraph significance if the random graph model is that of preserved degree sequence.

5 Conclusion

Based on a detailed analysis of previous approaches, we have presented new algorithmic techniques which allow for a faster detection of network motifs and offer useful additional features such as unbiased subgraph sampling and a specifically targeted detection of subgraph significance. This enables motif detection for larger networks and more complex motifs than was previously possible, hopefully facilitating future research on network motifs and its adjoining fields in systems biology.

Further research could improve the presented sampling technique, e.g., by examining how the labeling of the vertices in the input graph affects the sampling quality or seeing if RAND-ESU can be tweaked to selectively sample "interesting" parts of the input graph. For subgraph significance, we have shown that a direct calculation scheme may serve as a fast and accurate alternative to the explicit generation of random networks. It would be interesting to further explore this path by extending the scheme to classes of random background models other than those that solely preserve the degree sequence.

Acknowledgments

This research was supported by the Deutsche Telekom Stiftung and the Studienstiftung des deutschen Volkes. The author is grateful to Rolf Niedermeier (Jena), Jens Gramm (Tübingen), and Falk Hüffner (Jena) for helpful discussions and comments. An anonymous referee of WABI 2005 also made some insightful and useful remarks on an extended abstract of this work which appears under the title "A Faster Algorithm for Detecting Network Motifs" in the *Proceedings of the Fifth Workshop on Algorithms in Bioinformatics* (WABI '05), pp. 165-177, October 2005. Funded by the Deutsche Forschungsgemeinschaft project PEAL (Parameterized Complexity and Exact Algorithms), NI 369/1, Florian Rasche (Jena) greatly helped in extending the functionality and usability of the ESA implementation in order to make it a user-friendly motif detection tool.

References

[1] I. Albert and R. Albert, "Conserved Network Motifs Allow Protein-Protein Interaction Prediction," *Bioinformatics,* vol. 20, no. 18, pp. 3346-3352, 2004.
[2] N. Alon, R. Yuster, and U. Zwick, "Finding and Counting Given Length Cycles," *Algorithmica,* vol. 17, no. 3, pp. 209-223, 1997.
[3] L.A.N. Amaral, A. Scala, M. Barthélémy, and H.E. Stanley, "Classes of Small-World Networks," *Proc. Nat'l Academy of Sciences,* vol. 97, no. 21, pp. 11149-11152, 2000.
[4] Y. Artzy-Randrup, S.J. Fleishman, N. Ben-Tal, and L. Stone, "Comment on 'Network Motifs: Simple Building Blocks of Complex Networks' and 'Superfamilies of Designed and Evolved Networks'," *Science,* vol. 305, no. 5687, p. 1007c, 2004.
[5] A.L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science,* vol. 286, no. 5439, pp. 509-512, 1999.
[6] E.A. Bender, "The Asymptotic Number of Non-Negative Matrices with Given Row and Column Sums," *Discrete Math.,* vol. 10, no. 2, pp. 217-223, 1974.
[7] E.A. Bender and E.R. Canfield, "The Asymptotic Number of Labeled Graphs with Given Degree Sequences," *J. Combinatorial Theory Series A,* vol. 24, no. 3, pp. 296-307, 1978.
[8] J. Berg and M. Lässig, "Local Graph Alignment and Motif Search in Biological Networks," *Proc. Nat'l Academy of Sciences,* vol. 101, no. 41, pp. 14689-14694, 2004.
[9] D.A. Berque, M.K. Goldberg, and J.A. Edmonds, "Implementing Progress Indicators for Recursive Algorithms," *Proc. Fifth ACM-SIGAPP Symp. Applied Computing (SAC '93),* pp. 533-538, 1993.
[10] R.A. Duke, H. Lefmann, and V. Rödl, "A Fast Approximation Algorithm for Computing the Frequencies of Subgraphs in a Given Graph," *SIAM J. Computing,* vol. 24, no. 3, pp. 598-620, 1995.
[11] R.L. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science,* second ed. Addison-Wesley, 1994.
[12] S. Itzkovitz et al., "Coarse-Graining and Self-Dissimilarity of Complex Networks," *Physical Rev. E,* vol. 71, p. 016127, 2005.
[13] S. Itzkovitz et al., "Subgraphs in Random Networks," *Physical Rev. E,* vol. 68, p. 026127, 2003.
[14] S. Kalir et al., "Ordering Genes in a Flagella Pathway by Analysis of Expression Kinetics from Living Bacteria," *Science,* vol. 292, no. 5524, pp. 2080-2083, 2001.
[15] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient Sampling Algorithm for Estimating Subgraph Concentrations and Detecting Network Motifs," *Bioinformatics,* vol. 20, no. 11, pp. 1746-1758, 2004.
[16] D.E. Knuth, "Estimating the Efficiency of Backtrack Programs," *Selected Papers on Analysis of Algorithms,* Stanford Junior Univ., Palo Alto, Calif., 2000.
[17] M. Koyutürk, A. Grama, and , and W. Szpankowski, "An Efficient Algorithm for Detecting Frequent Subgraphs in Biological Networks," *Bioinformatics,* vol. 20, no. supplement 1, pp. i200-i207, 2004.
[18] T.I. Lee et al., "Transcriptional Regulatory Networks in Saccharomyces Cerevisiae," *Science,* vol. 298, no. 5594, pp. 799-804, 2002.
[19] S. Li et al., "A Map of the Interactome Network of the Metazoan C. Elegans," *Science,* vol. 303, no. 5657, pp. 540-543, 2004.
[20] B.D. McKay, "Practical Graph Isomorphism," *Congressus Numerantium,* vol. 30, pp. 45-87, 1981.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

[21] B.D. McKay, "Nauty User's Guide (Version 1.5)," Technical Report TR-CS-90-02, Dept. of Computer Science, Australian Nat'l Univ., 1990.
[22] R. Milo et al., "Response to Comment on 'Network Motifs: Simple Building Blocks of Complex Networks' and 'Superfamilies of Designed and Evolved Networks'," *Science,* vol. 305, no. 5687, p. 1007d, 2004.
[23] R. Milo et al., "Superfamilies of Designed and Evolved Networks," *Science,* vol. 303, no. 5663, pp. 1538-1542, 2004.
[24] R. Milo et al., "Network Motifs: Simple Building Blocks of Complex Networks," *Science,* vol. 298, no. 5594, pp. 824-827, 2002.
[25] M.E.J. Newman, S.H. Strogatz, and D.J. Watts, "Random Graphs with Arbitrary Degree Distributions and Their Applications," *Physical Rev. E,* vol. 64, p. 026118, 2001.
[26] S. Ott, A. Hansen, S. Kim, and S. Miyano, "Superiority of Network Motifs over Optimal Networks and an Application to the Revelation of Gene Network Evolution," *Bioinformatics,* vol. 21, no. 2, pp. 227-238, 2005.
[27] M. Ronen, R. Rosenberg, B.I. Shraiman, and U. Alon, "Assigning Numbers to the Arrows: Parameterizing a Gene Regulation Network by Using Accurate Expression Kinetics," *Proc. Nat'l Academy of Sciences,* vol. 99, no. 16, pp. 10555-10560, 2002.
[28] S.S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network Motifs in the Transcriptional Regulation Network of Escherichia Coli," *Nature Genetics,* vol. 31, no. 1, pp. 64-68, 2002.
[29] E. Szemerédi, "Regular Partitions of Graphs," *Problèmes Combinatoires et Théorie des Graphes,* no. 260 in Colloques internationaux CNRS, pp. 399-401, 1976.
[30] A. Vázquez et al., "The Topological Relationship between the Large-Scale Attributes and Local Interaction Patterns of Complex Networks," *Proc. Nat'l Academy of Sciences,* vol. 101, no. 52, pp. 17940-17945, 2004.
[31] A. Vespignani, "Evolution Thinks Modular," *Nature Genetics,* vol. 35, no. 2, pp. 118-119, 2003.
[32] D.J. Watts, *Small Worlds: The Dynamics of Networks between Order and Randomness.* Princeton Univ. Press, 1999.
[33] S. Wernicke and F. Rasche, "FANMOD: A Tool for Fast Network Motif Detection," *Bioinformatics,* vol. 22, no. 9, pp. 1152-1153, 2006.
[34] H.S. Wilf, *Generatingfunctionology,* second ed. Academic Press, 1994.
[35] R.J. Williams and N.D. Martinez, "Simple Rules Yield Complex Food Webs," *Nature,* vol. 404, no. 6774, pp. 180-183, 2000.
[36] S.H. Yook, Z.N. Oltvai, and A.L. Barabási, "Functional and Topological Characterization of Protein Interaction Networks," *Proteomics,* vol. 4, no. 4, pp. 928-942, 2004.

Sebastian Wernicke studied bioinformatics at the Eberhard-Karls-Universität in Tübingen, Germany, from which he graduated in 2003. Currently, he is a PhD student of Rolf Niedermeier, chair for theoretical computer science and computational complexity, at the Friedrich-Schiller-Universität in Jena, Germany. His research focuses on parameterized and exact algorithms for hard combinatorial problems in computational biology.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: POLO BIBLIOTECARIO DI INGEGNERIA. Downloaded on October 29,2025 at 22:34:14 UTC from IEEE Xplore. Restrictions apply.

Motif Prediction with Graph Neural Networks

Maciej Besta
Raphael Grob
ETH Zurich

Cesare Miglioli
Research Center for Statistics,
University of Geneva

Nicola Bernold
Grzegorz Kwasniewski
Gabriel Gjini
ETH Zurich

Raghavendra Kanakagiri
University of Illinois at
Urbana-Champaign

Saleh Ashkboos
Lukas Gianinazzi
ETH Zurich

Nikoli Dryden
Torsten Hoefler
ETH Zurich

ABSTRACT

Link prediction is one of the central problems in graph mining. However, recent studies highlight the importance of *higher-order network analysis*, where complex structures called motifs are the first-class citizens. We first show that existing link prediction schemes fail to effectively predict motifs. To alleviate this, we establish a general *motif prediction problem* and we propose several heuristics that assess the chances for a specified motif to appear. To make the scores realistic, our heuristics consider – among others – *correlations between links*, i.e., the potential impact of some arriving links on the appearance of other links in a given motif. Finally, for highest accuracy, we develop a graph neural network (GNN) architecture for motif prediction. Our architecture offers vertex features and sampling schemes that capture the rich structural properties of motifs. While our heuristics are fast and do not need any training, GNNs ensure highest accuracy of predicting motifs, both for dense (e.g., k-cliques) and for sparse ones (e.g., k-stars). We consistently outperform the best available competitor by more than 10% on average and up to 32% in area under the curve. Importantly, the advantages of our approach over schemes based on uncorrelated link prediction increase with the increasing motif size and complexity. We also successfully apply our architecture for predicting more arbitrary *clusters* and *communities*, illustrating its potential for graph mining beyond motif analysis.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

Motifs, Motif Prediction, Link Prediction, Graph Neural Networks

ACM Reference Format:
Maciej Besta et al.. 2022. Motif Prediction with Graph Neural Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539343

An extended version: https://arxiv.org/abs/2106.00761

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
KDD '22, August 14–18, 2022, Washington, DC, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539343

1 INTRODUCTION AND MOTIVATION

One of the central problems in graph mining and learning is link prediction [3, 4, 40, 51, 61, 63], in which one is interested in assessing the likelihood that a given *pair of vertices* is, or may become, connected. However, recent works argue the importance of *higher-order graph organization* [6], where one focuses on finding and analyzing small recurring *subgraphs* called *motifs* (sometimes referred to as *graphlets* or *graph patterns*) instead of individual links. Motifs are central to many graph mining problems in computational biology, chemistry, and a plethora of other fields [9, 12, 13, 18, 20, 27, 30]. Specifically, motifs are building blocks of different networks, including transcriptional regulation graphs, social networks, brain graphs, or air traffic patterns [6]. There exist many motifs, for example k-cliques, k-stars, k-clique-stars, k-cores, and others [8, 29, 36]. For example, cliques or quasi-cliques are crucial motifs in protein-protein interaction networks [15, 38]. A huge number of works are dedicated to motif *counting*, *listing* (also called *enumeration*), or *checking for the existence* of a given motif [12, 20]. However, while a few recent schemes focus on predicting *triangles* [7, 43, 44], no works target the problem of *general motif prediction*, i.e., analyzing whether specified complex structures may appear in the data. As with link prediction, it would enable predicting the evolution of data, but also finding missing structures in the available data. For example, one could use motif prediction to find probable missing clusters of interactions in biological (e.g., protein) networks, and use the outcomes to limit the number of expensive experiments conducted to find missing connections [40, 41].

In this paper, we first (Section 3) establish and formally describe a general motif prediction problem, going beyond link prediction and showing how to predict higher-order network patterns that will appear in the future (or which may be missing from the data). A key challenge is the appropriate *problem formulation*. Similarly to link prediction, one wants a *score function* that – for a given vertex set V_M – assesses the chances for a given motif to appear. Still, the function must consider the combinatorially increased complexity of the problem (compared to link prediction). In general, contrary to a single link, a motif may be formed by an *arbitrary* set V_M of vertices, and the number of potential edges between these vertices can be large, i.e., $O(|V_M|^2)$. For example, one may be interested in analyzing whether a *group* of entities V_M may become a k-clique in the future, or whether a specific vertex $v \in V_M$ will become a *hub* of a k-star, connecting v to $k-1$ other selected vertices from $V_M \setminus \{v\}$. This leads to novel issues, not present in link prediction. For example, what if *some* edges, belonging to the motif being predicted, already exist? How should they be treated by a score function? Or,

how to enable users to apply their domain knowledge? For example, when predicting whether the given vertices will form some chemical particle, a user may know that the presence of some link (e.g., some specific atomic bond) may increase (or decrease) the chances for forming another bond. Now, how could this knowledge be provided in the motif score function? We formally specify these and other aspects of the problem in a general theoretical framework, and we provide example motif score functions. We explicitly consider correlations between edges forming a motif, i.e., the fact that the appearance of some edges may increase or decrease the overall chances of a given motif to appear.

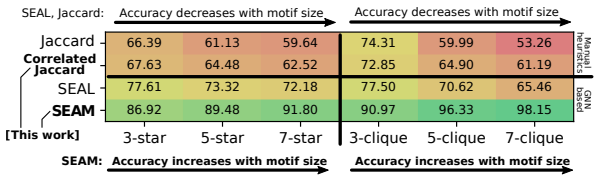


Figure 1: Motivating our work (SEAM): the accuracy (%) of predicting different motifs with SEAM compared to using a state-of-the-art SEAL link prediction scheme [61, 63] and a naive one that does not consider correlations between edges. The details of the experimental setup are in Section 5 (the dataset is USAir). **Importantly: (1)** SEAM outperforms all other methods, **(2)** the accuracy of SEAM *increases* with the size (k) of each motif, while in other methods it *decreases*.

Then, we develop a learning architecture based on graph neural networks (GNNs) to further enhance motif prediction accuracy (Section 4). For this, we extend the state-of-the-art SEAL link prediction framework [61] to support arbitrary motifs. For a given motif M, we train our architecture on what is the "right motif surroundings" (i.e., nearby vertices and edges) that could result in the appearance of M. Then, for a given set of vertices V_M, the architecture infers the chances for M to appear. The key challenge is to be able to capture the *richness* of different motifs and their surroundings. We tackle this with an appropriate selection of *negative* samples, i.e., subgraphs that resemble the searched motifs but that are not identical to them. Moreover, when selecting the size of the "motif surroundings" we rely on an assumption also used in link prediction, which states that only the "close surroundings" (i.e., nearby vertices and edges, 1–2 hops away) of a link to be predicted have a significant impact on whether or not this link would appear [61, 63]. We use this assumption for motifs: as our evaluation shows, it ensures high accuracy while significantly reducing runtimes of training and inference (as only a small subgraph is used, instead of the whole input graph). We call our GNN architecture **SEAM**: learning from Subgraphs, Embeddings and Attributes for **Motif** prediction[1]. Our evaluation (Section 5) illustrates the high accuracy of SEAM (often more than 90%), for a variety of graph datasets and motif sizes.

To motivate our work, we now compare SEAM and a proposed Jaccard-based heuristic that considers link correlations to two baselines that straightforwardly *use link prediction independently for each motif link*: a Jaccard-based score and the state-of-the-art SEAL scheme based on GNNs [61]. We show the results in Figure 1. The correlated Jaccard outperforms a simple Jaccard, while the proposed SEAM is better than SEAL. The benefits generalize to different graph datasets. Importantly, we observe that the larger the motif to predict becomes (larger k), *the more advantages our architecture delivers*. This is because larger motifs provide more room for *correlations between their associated edges*. Straightforward link prediction based schemes do not consider this effect, while our methods do, which is why we offer more benefits for more complex motifs. The advantages of SEAM over the correlated Jaccard show that GNNs more robustly capture correlations and the structural richness of motifs than simple manual heuristics. Simultaneously, heuristics do not need any training. Finally, SEAM also successfully predicts more arbitrary *communities* or *clusters* [12, 13, 25, 36]. They differ from motifs as they do not have a very specific fixed structure (such as a star) but simply have the edge density above a certain threshold. SEAM's high accuracy in predicting such structures illustrates its potential for broader graph mining beyond motif analysis.

Overall, the key contributions of our paper are (1) identifying and formulating the motif prediction problem and the associated score functions, (2) showing how to solve this problem with heuristics and graph neural networks, and (3) illustrating that graph neural networks can solve this problem more effectively than heuristics.

[1] In analogy to SEAL [61, 63], which stands for "learning from Subgraphs, Embeddings, and Attributes for Link prediction".

2 BACKGROUND AND NOTATION

We first describe the necessary background and notation.

Graph Model We model an undirected graph G as a tuple (V, E); V and $E \subseteq V \times V$ are sets of nodes (vertices) and links (edges); $|V| = n$, $|E| = m$. Vertices are modeled with integers $1, ..., n$; $V = \{1, ..., n\}$. N_v denotes the neighbors of $v \in V$; $d(v)$ denotes the degree of v.

Link Prediction We generalize the well-known link prediction problem. Consider two unconnected vertices u and v. We assign a *similarity score* $s_{u,v}$ to them. All pairs of vertices that are not edges receive such a score and are ranked according to it. The higher a similarity score is, the "more likely" a given edge is to be missing in the data or to be created in the future. We stress that the link prediction scores are usually not based on any probabilistic notion (in the formal sense) and are only used to make comparisons between pairs of vertices in the same input graph dataset.

There are numerous known similarity scores. First, a large number of scores are called *first order* because they only consider the neighbors of u and v when computing $s_{u,v}$. Examples are the **Common Neighbors** scheme $s^{CN}_{u,v} = |N_u \cap N_v|$ or the **Jaccard** scheme $s^{J}_{u,v} = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$ [10]. These schemes assume that two vertices are more likely to be linked if they have many common neighbors. There also exist similarity schemes that consider vertices not directly attached to u and v. All these schemes can be described using the same formalism of the *γ-decaying heuristic* proposed by [61]. Intuitively, for a given pair of vertices (u, v), the γ-decaying heuristic for (u, v) provides a sum of contributions into the link prediction score for (u, v) from all other vertices, weighted in such a way that nearby vertices have more impact on the score.

Graph Neural Networks Graph neural networks (GNNs) are a recent class of neural networks for learning over irregular data such as graphs [17, 19, 49, 50, 52, 56, 58, 59, 64, 65]. There exists a plethora of models and methods for GNNs; most of them consist of two fundamental parts: (1) an aggregation layer that combines the features of the neighbors of each node, for all the nodes in the input graph, and (2) combining the scores into a new score. The input to a GNN is a tuple $G = (A, X)$. The input graph G having n vertices is modeled with an adjacency matrix $A \in \mathbb{R}^{n \times n}$. The features of vertices (with dimension d) are modeled with a matrix $X \in \mathbb{R}^{n \times d}$.

Symbol	Description
E_M	All edges forming a motif in question; $E_M = E_{M,\mathcal{N}} \cup E_{M,\mathcal{E}}$
$E_{M,\mathcal{N}}$	Motif edges that do not yet exist
$E_{M,\mathcal{E}}$	Motif edges that already exist in the data
$\overline{E}_M$	Edges not in E_M, defined over vertex pairs in V_M; $\overline{E}_M = \overline{E}_{M,\mathcal{D}} \cup \overline{E}_{M,\mathcal{I}}$
E_{V_M}	All possible edges between motif vertices; $E_{V_M} = \overline{E}_M \cup E_M$
$\overline{E}_{M,\mathcal{D}}$	Deal-breaker edges; $\overline{E}_{M,\mathcal{D}} = \overline{E}_{M,\mathcal{D},\mathcal{N}} \cup \overline{E}_{M,\mathcal{D},\mathcal{E}}$
$\overline{E}_{M,\mathcal{D},\mathcal{N}}$	Deal-breaker edges that do not exist yet
$\overline{E}_{M,\mathcal{D},\mathcal{E}}$	Deal-breaker edges that already exist
$\overline{E}_{M,\mathcal{I}}$	Non deal-breaker edges in $\overline{E}_M$; "edges that do not matter"
E^*_M	"Edges that matter for the score": $E^*_M = E_M \cup \overline{E}_{M,\mathcal{D}}$
$E^*_{M,\mathcal{E}}$	All existing edges "that matter": $E^*_{M,\mathcal{E}} = E_{M,\mathcal{E}} \cup \overline{E}_{M,\mathcal{D},\mathcal{E}}$
$E^*_{M,\mathcal{N}}$	All non-existing edges "that matter": $E^*_{M,\mathcal{N}} = E_{M,\mathcal{N}} \cup \overline{E}_{M,\mathcal{D},\mathcal{N}}$

Table 1: Different types of edges used in this work.

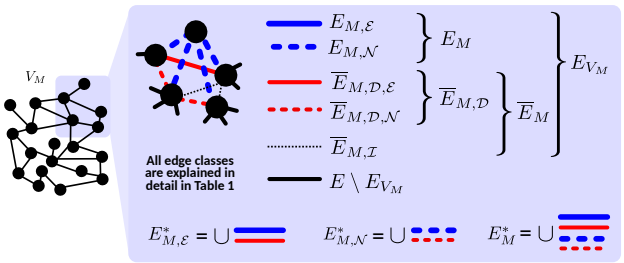


Figure 3: Illustration of edge types in motif prediction.

3 MOTIF PREDICTION: FORMAL STATEMENT AND SCORE FUNCTIONS

We now formally establish the motif prediction problem. We define a motif as a pair $M = (V_M, E_M)$. V_M is the set of *existing* vertices of G that form a given motif ($V_M \subseteq V$). E_M is the set of edges of G that form the motif being predicted; some of these edges may already exist ($E_M \subseteq V_M \times V_M$).

We make the problem formulation (in § 3.1–§ 3.3) *general*: it can be applied to any graph generation process. Using this formulation, one can then devise specific heuristics that may assume some details on how the links are created, similarly as is done in link prediction. Here, we propose example motif prediction heuristics that harness the Jaccard, Common Neighbors, and Adamic-Adar link scores.

We illustrate motif prediction and supported motifs in Figure 2.

3.1 Motif Prediction vs. Link Prediction

We illustrate the motif prediction problem by discussing the differences between link and motif prediction. We consider all these differences when proposing specific schemes for predicting motifs.

(M) There May Be Many Potential New Motifs For a Fixed Vertex Set Link prediction is a "binary" problem: for a given pair of unconnected vertices, there can only be one link appearing. In motif prediction, the situation is more complex. There are many possible motifs to appear between given vertices $v_1, ..., v_k$. We now state a precise count; the proof is in the appendix.

Observation 1. *Consider vertices $v_1, ..., v_k \in V$. Assuming no edges already connecting $v_1, ..., v_k$, there are $2^{\binom{k}{2}} - 1$ motifs (with between 1 and $\binom{k}{2}$ edges) that can appear to connect $v_1, ..., v_k$.*

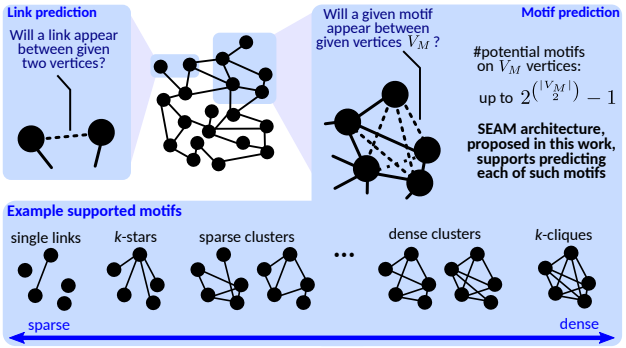


Figure 2: Illustration of the motif prediction problem and example supported motifs. We provide support for predicting arbitrary motifs.

Note that this is the largest possible number, which assumes no previously existing edges, and permutation dependence, i.e., two motifs that are isomorphic but have different vertex orderings, are treated as two different motifs. This enables, for example, the user to be able to distinguish between two stars rooted at different vertices. This is useful in, e.g., social network analysis, when stars rooted at different persons may well have different meaning.

(E) There May Be Existing Edges A link can only appear between *unconnected* vertices. Contrarily, a motif can appear and connect vertices *already* with some edges between them.

(D) There May Be "Deal-Breaker" Edges There may be some edges, the appearance of which would make the appearance of a given motif *unlikely* or even *impossible* (e.g., existing chemical bonds could prevent other bonds). We will refer to such edges as the "deal-breaker" edges.

(L) Motif Prediction Query May Depend on Vertex Labeling The query can depend on a specific vertex labeling. For example, when asking whether a 5-star will connect six given vertices $v_1, ..., v_6$, one may be interested in a 5-star connecting these vertices in a *specific way*, e.g., with its center being v_1. We enable the user to specify how edges in E_M should connect vertices in V_M.

3.2 Types of Edges in Motifs

We first describe different types of edges related to a motif. They are listed in Table 1 and shown in Figure 3. First, note that motif edges E_M are a union of two types of motif edges, i.e., $E_M = E_{M,\mathcal{N}} \cup E_{M,\mathcal{E}}$ where $E_{M,\mathcal{N}}$ are edges that do not exist in G at the moment of querying ($\forall_{e \in E_{M,\mathcal{N}}} e \notin E$; $\mathcal{N}$ indicates "$\mathcal{N}$on-existing") and $E_{M,\mathcal{E}}$ are edges that already exist, cf. (E) in § 3.1 ($\forall_{e \in E_{M,\mathcal{E}}} e \in E$; $\mathcal{E}$ indicates "$\mathcal{E}$xisting"). Moreover, there may be edges between vertices in V_M which do *not* belong to M (i.e., they belong to $E_{V_M} = \{\{i, j\} : i, j \in V_M \wedge i \neq j\}$ but *not* E_M). We refer to such edges as $\overline{E}_M$ since $E_{V_M} = \overline{E}_M \cup E_M$ (i.e., a union of disjoints sets). Some edges in $\overline{E}_M$ may be deal-breakers (cf. (D) in § 3.1), we denote them as $\overline{E}_{M,\mathcal{D}}$ ($\mathcal{D}$ indicates "$\mathcal{D}$eal-breaker"). Non deal-breakers that are in $\overline{E}_M$ are denoted with $\overline{E}_{M,\mathcal{I}}$ ($\mathcal{I}$ indicates "$\mathcal{I}$nert"). Note that $\overline{E}_M = \overline{E}_{M,\mathcal{D}} \cup \overline{E}_{M,\mathcal{I}}$ and $\overline{E}_M = E_{V_M} \setminus E_M$. To conclude, as previously done for the set E_M, we note that $\overline{E}_{M,\mathcal{D}} = \overline{E}_{M,\mathcal{D},\mathcal{N}} \cup \overline{E}_{M,\mathcal{D},\mathcal{E}}$ where $\overline{E}_{M,\mathcal{D},\mathcal{N}}$ are deal-breaker edges that do not exist in G at the moment of querying ($\forall_{e \in \overline{E}_{M,\mathcal{D},\mathcal{N}}} e \notin E$; $\mathcal{N}$ indicates "$\mathcal{N}$on-existing") and $\overline{E}_{M,\mathcal{D},\mathcal{E}}$ are deal-breaker edges that already exist, cf. (E) in § 3.1 ($\forall_{e \in \overline{E}_{M,\mathcal{D},\mathcal{E}}} e \in E$; $\mathcal{E}$ indicates "$\mathcal{E}$xisting"). We explicitly consider $\overline{E}_{M,\mathcal{D},\mathcal{N}}$ because – even if a given deal-breaker edge does not exist, but it *does* have a large chance of appearing – the motif score should become lower.

3.3 General Problem and Score Formulation

We now formulate a general *motif prediction score*. Analogously to link prediction, we assign scores to motifs, to be able to quantitatively assess which motifs are more likely to occur. Intuitively, we assume that a motif score should be high if the scores of participating edges are also high. This suggests one could reuse link prediction score functions. Full extensive details of score functions, as well as more examples, are in the appendix.

A specific motif score function $s(M)$ will heavily depend on a targeted problem. In general, we define $s(M)$ as a function of V_M and E^*_M; $s(M) = s(V_M, E^*_M)$. Here, $E^*_M = E_M \cup \overline{E}_{M,\mathcal{D}}$ are all the edges "that matter": both edges in a motif (E_M) and the deal-breaker edges ($\overline{E}_{M,\mathcal{D}}$). To obtain the exact form of $s(M)$, we harness existing link prediction scores for edges from E_M, when deriving $s(M)$ (details in § 3.4–§ 3.5). When using first-order link prediction methods (e.g., Jaccard), $s(M)$ depends on V_M and potential direct neighbors. With higher-order methods (e.g., Katz [32] or Adamic-Adar [2]), a larger part of the graph that is "around V_M" is considered for computing $s(M)$. Here, our evaluation (cf. Section 5) shows that, similarly to link prediction [61], it is enough to consider a small part of G (1-2 hops away from V_M) to achieve high prediction accuracy for motifs.

Still, simply extending link prediction fails to account for possible *correlations* between edges forming the motif (i.e., edges in E_M). Specifically, the appearance of some edges may impact (positively or negatively) the chances of one or more other edges in E_M. We provide score functions that consider such correlations in § 3.5.

3.4 Heuristics with No Link Correlations

There exist many score functions for link prediction [3, 4, 40, 51]. Similarly, one can develop motif prediction score functions with different applications in mind. As an example, we discuss score functions for a graph that models a set of people. An edge between two vertices indicates that two given persons know each other. For simplicity, let us first assume that there are no deal-breaker edges, thus $E^*_M = E_M$. For a set of people V_M, we set the score of a given specific motif $M = (V_M, E_M)$ to be the product of the scores of the associated edges: $s_\perp(M) = \prod_{e \in E_{M,\mathcal{N}}} s(e)$ where $\perp$ denotes the independent aggregation scheme. Here, $s(e)$ is any link prediction score which outputs into $[0, 1]$ (e.g., Jaccard). Thus, also $s_\perp(M) \in [0, 1]$ by construction. Moreover, this score implicitly states that $\forall e \in E_{M,\mathcal{E}}$ we set $s(e) = 1$. Clearly, this does not impact the motif score $s_\perp(M)$ as the edges are already $\mathcal{E}$xisting. Overall, we assume that a motif is more likely to appear if the edges that participate in that motif are also more likely. Now, when using the **Jaccard Score** for edges, the motif prediction score becomes $s_\perp(M)^J = \prod_{e_{u,v} \in E_{M,\mathcal{N}}} \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$.

To **incorporate deal-breaker edges**, we generalize the motif score defined previously as $s^*_\perp(M) = \prod_{e \in E_M} s(e) \cdot \prod_{e \in \overline{E}_{M,\mathcal{D}}} (1 - s(e))$, where the product over E_M includes partial scores from the edges that belong to the motif, while the product over $\overline{E}_{M,\mathcal{D}}$ includes the scores from deal-breaker edges. Here, the larger the chance for a e to appear, the higher its score $s(e)$ is. Thus, whenever e is a deal-breaker, using $1 - s(e)$ has the desired diminishing effect on the final motif score $s^*_\perp(M)$.

3.5 Heuristics for Link Correlations

The main challenge is how to aggregate the link predictions taking into account the rich structural properties of motifs. Intuitively, using a plain product of scores implicitly assumes the independence of participating scores. However, arriving links may increase the chances of other links' appearance in non-trivial ways. To capture such *positive correlations*, we propose heuristics based on the *convex linear combination of link scores*. To show that such schemes consider correlations, we first (Proposition 3.1) prove that the product P of any numbers in $[0, 1]$ is always bounded by the convex linear combination C of those numbers (the proof is in the appendix). Thus, our motif prediction scores based on the convex linear combination of link scores are always at least as large as the independent products of link scores (as we normalize them to be in $[0, 1]$, see § 3.6). The difference $(C - P)$ is due to link correlations. Details are in § 3.5.1.

Proposition 3.1. *Let $\{x_1, ..., x_n\}$ be any finite collection of elements from $U = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$. Then, $\forall n \in \mathbb{N}$ we have $\prod_{i=1}^n x_i \leq \sum_{i=1}^n w_i x_i$, where $w_i \geq 0$ $\forall i \in \{1, ..., n\}$ and subject to the constraint $\sum_{i=1}^n w_i = 1$.*

For *negative correlations* caused by deal-breaker edges, i.e., correlations that lower the overall chances of some motif to appear, we introduce appropriately normalized scores with a negative sign in the weighted score sum. The validity of this approach follows from Proposition 3.1 by noting that $\prod_{i=1}^n x_i \geq -\sum_{i=1}^n w_i x_i$ under the conditions specified in the proposition. This means that any combination of such negatives scores is always lower than the product of scores P; the difference $|C - P|$ again indicates effects between links not captured by P. Details are in § 3.5.2.

3.5.1 Capturing Positive Correlation. In order to introduce positive correlation, we set the score of a given specific motif $M = (V_M, E_M)$ to be the convex linear combination of the vector of scores of the associated edges:

$$s(M) = f(\mathbf{s}(\mathbf{e})) = \langle \mathbf{w}, \mathbf{s}(\mathbf{e}) \rangle \tag{1}$$

Here, $f(\mathbf{s}(\mathbf{e})) : [0, 1]^{|E_M|} \rightarrow [0, 1]$ with $|E_M| = |E_{V_M} \setminus \overline{E}_M|$ (i.e., not considering either $\mathcal{I}$nert or $\mathcal{D}$eal-breaker edges). In the weight vector $\mathbf{w} \in [0, 1]^{|E_M|}$, each component w_i is larger than zero, subject to the constraint $\sum_{i=1}^{|E_M|} w_i = 1$. Thus, $s(M)$ is a convex linear combination of the vector of link prediction scores $\mathbf{s}(\mathbf{e})$. Finally, we assign a unit score for each existing edge $e \in E_{M,\mathcal{E}}$.

Now, to obtain a **correlated Jaccard score for motifs**, we set a score for each $\mathcal{N}$on-existing edge $e_{(u,v)}$ as $\frac{|N_u \cap N_v|}{|N_u \cup N_v|}$. $\mathcal{E}$xisting edges each receive scores 1. Finally, we set the weights as $\mathbf{w} = \mathbf{1}\frac{1}{|E_M|}$, assigning the same importance to each link in the motif M. This gives $s(M)^J = \frac{1}{|E_M|} \left(\sum_{e_{u,v} \in E_{M,\mathcal{N}}} \frac{|N_u \cap N_v|}{|N_u \cup N_v|} + |E_{M,\mathcal{E}}| \right)$. Any choice of $w_i > \frac{1}{|E_M|}$ places a larger weight on the i-th edge (and lower for others due to the constraint $\sum_{i=1}^{|E_M|} w_i = 1$). In this way we can incorporate domain knowledge for the motif of interest. For example, in Figure 5, we set $\mathbf{w} = \mathbf{1}\frac{1}{|E_{M,\mathcal{N}}|}$ because of the relevant presence of $\mathcal{E}$xisting edges (each receiving a null score).

3.5.2 Capturing Negative Correlation. To capture *negative correlation* potentially coming from deal-breaker edges, we assign negative

signs to the respective link scores. Let $e \in E^*_M = E_M \cup \overline{E}_{M,\mathcal{D}}$. Then we set $s^*_i(e) = -s_i(e)$ if $e \in \overline{E}_{M,\mathcal{D},\mathcal{N}}$, $\forall i \in \{1, ..., |E^*_M|\}$. Moreover, if there is an edge $e \in \overline{E}_{M,\mathcal{D},\mathcal{E}}$, we have $\mathbf{s}^*(\mathbf{e}) = \mathbf{0}$. Assigning a negative link prediction score to a *potential* $\mathcal{D}$eal-breaker edge lowers the score of the motif. Setting $\mathbf{s}^*(\mathbf{e}) = \mathbf{0}$ when at least one $\mathcal{D}$eal-breaker edge exists, allows us to rule out motifs which cannot arise. We now state a final motif prediction score:

$$s^*(M) = f(\mathbf{s}^*(\mathbf{e})) = \max(0, \langle \mathbf{w}, \mathbf{s}^*(\mathbf{e}) \rangle) \tag{2}$$

Here $s^*(M) : [0,1]^{|E^*_M|} \to [0,1]$ with $|E^*_M| \leq \binom{|V_M|}{2}$. Furthermore, we apply a rectifier on the convex linear combination of the transformed scores vector (i.e., $\langle \mathbf{w}, \mathbf{s}^*(\mathbf{e}) \rangle$) with the rationale that any negative motif score implies the same impossibility of the motif to appear. All other score elements are identical to those in Eq. (1).

3.6 Normalization of Scores for Meaningful Comparisons and General Applicability

Motif scores defined so far consider only link prediction scores $s(e) \in [0,1]$. Thus, popular heuristics such as Common Neighbors, Preferential Attachment, or Adamic-Adar do not fit into this framework. For this, we introduce a *normalized* score $s(e)/c$ enforcing $c \geq \lceil \|\mathbf{s}(\mathbf{e})\|_\infty \rceil$ since the infinity norm of the vector of scores is the smallest value that ensures the desired mapping (the ceil function defines a proper generalization as $\lceil \|\mathbf{s}(\mathbf{e})\|_\infty \rceil = 1$ for, e.g., Jaccard [10]). Normalization allows to *meaningfully compare scores of different motifs that may differ in size or in their edge sets* E_M.

4 SEAM GNN ARCHITECTURE

We argue that one could *use neural networks to learn a heuristic for motif prediction.* Following recent work on link prediction [61, 63], we use a GNN for this; a GNN may be able to learn link correlations better than a simple hand-designed heuristic. Simultaneously, heuristics are still important as they do not require expensive training. We now describe a GNN architecture called SEAM (learning from Subgraphs, Embeddings and Attributes for Motif prediction). A high-level overview is in § 4.1 and in Figure 4.

4.1 Overview

Let $M = (V_M, E_M)$ be a motif to be predicted in G. First, we **extract the already existing instances of** M **in** G, denoted as $G_p = (V_p, E_p)$; $V_p \subseteq V$ and $E_p \subseteq E$. We use these instances G_p to **generate positive samples** for training and validation. To **generate negative samples** (details in § 4.3), we find subgraphs $G_n = (V_n, E_n)$ that do *not* form a motif M (i.e., $V_n = V_M$ and $E_M \nsubseteq E_n$ or $\overline{E}_{M,\mathcal{D}} \cap E_n \neq \emptyset$). Then, for each positive and negative sample, consisting of sets of vertices V_p and V_n, we **extract a "subgraph around this sample"**, $G_s = (V_s, E_s)$, with $V_p \subseteq V_s \subseteq V$ and $E_p \subseteq E_s \subseteq E$, or $V_n \subseteq V_s \subseteq V$ and $E_n \subseteq E_s \subseteq E$ (details in § 4.4). Here, we rely on the insights from SEAL [61] on their γ-decaying heuristic, i.e., it is G_s, the "surroundings" of a given sample (be it positive or negative), that are important in determining whether M appears or not. The nodes of these subgraphs are then **appropriately labeled** to encode the structural information (details in § 4.6). With these labeled subgraphs, we **train our GNN**, which classifies each subgraph depending on whether or not vertices V_p or V_n form the motif M. After training, we **evaluate the real world accuracy of our GNN** by using the validation dataset.

4.2 Specifying Motifs of Interest

The user specifies the motif to be predicted. SEAM provides an interface for selecting (1) vertices V_M of interest, (2) motif edges E_M, and (3) potential deal-breaker edges $\overline{E}_{M,\mathcal{D}}$. The user first picks V_M and then they can specify *any* of up to $2^{\binom{|V_M|}{2}} - 1$ potential motifs as a target of the prediction. The interface also enables specifying the vertex ordering, or motif's permutation invariance.

4.3 Positive and Negative Sampling

We need to provide a diverse set of samples to ensure that SEAM works reliably on a wide range of real data. For the positive samples, this is simple because the motif to be predicted (M) is specified. Negative samples are more challenging, because – for a given motif – there are many potential "false" motifs. In general, for each motif M, we generate negative samples using three strategies. (1) We first select positive samples and then remove a few vertices, replacing them with other nearby vertices (i.e., only a small number of motif edges are missing or only a small number of deal-breaker edges are added). Such negative samples closely resemble the positive ones. (2) We randomly sample V_M vertices from the graph; such negative samples are usually sparsely connected and do not resemble the positive ones. (3) We select a random vertex r into an empty set, and then we keep adding randomly selected vertices from the union over the neighborhoods of vertices already in the set, growing a subgraph until reaching the size of V_M; such negative samples may resemble the positive ones to a certain degree. The final set of negative samples usually contains about 80% samples generated by strategy (1) and 10% each of samples generated by (2) and (3). This distribution could be adjusted based on domain knowledge of the input graph (we also experiment with other ratios). Strategies (2) and (3) are primarily used to avoid overfitting of our model.

As an example, let our motif M be a 3-clique ($|V_M| = 3$ and $|E_M| = 3$). Consider a simple approach of generating negative samples, in which one randomly samples 3 vertex indices and verifies if there is a closed 3-clique between them. If we use these samples, in our evaluation for considered real world graphs, this leads to a distribution of 90% unconnected samples $|E_n| = 0$, 9% samples with $|E_n| = 1$ and only about 1% of samples with $|E_n| = 2$. Thus, if we train our GNN with this dataset, it would hardly learn the difference between open 3-cliques $|E_n| = 2$ and closed 3-cliques $|E_M| = 3$. Therefore, we provide our negative samples by ensuring that a third of samples are open 3-cliques $|E_n| = 2$ and another third of samples have one edge $|E_M| = 1$. For the remaining third of samples, we use the randomly generated vertex indices described above, which are mostly unconnected vertices $|E_M| = 0$.

Overall, we choose equally many positive and negative samples to ensure a balanced dataset. Furthermore, we limit the number of samples if there are too many, by taking a subset of the samples (selected uniformly at random). The positive and negative samples are split into a training dataset and a validation dataset. This split is typically done in a 9/1 ratio. To ensure an even distribution of all types of samples in these two datasets, we randomly permute the samples before splitting them.

4.4 Extracting Subgraphs Containing Samples

To reduce computational costs, we do not use the entire graph G as input in training or validation. Instead, we rely on recent insights on link prediction with GNNs [61, 63], which illustrate that

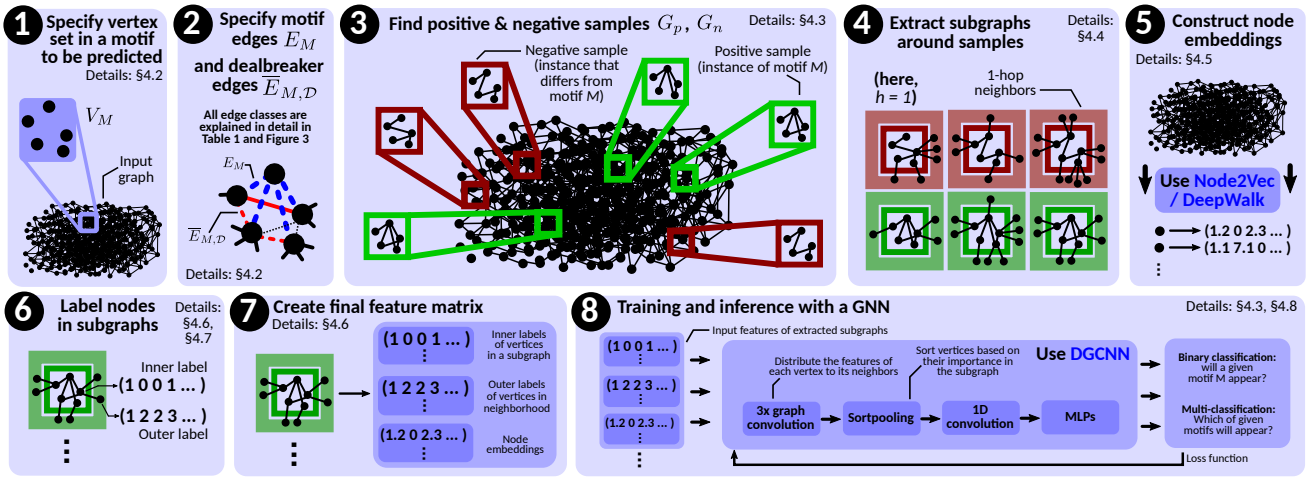


Figure 4: High-level overview of SEAM.

it suffices to provide a subgraph capturing the "close surroundings" (i.e., 1–2 hops away) of the vertices we want to predict a link between, cf. Section 2. We take an analogous assumption for motifs (our evaluation confirms the validity of the assumption). For $G = (V, E)$ and $V_M \subseteq V$, the *h-hop enclosing subgraph* $G^h_{V_M}$ (i.e., the "surroundings" of a motif M) is given by the set of nodes $\{i \in V \mid \exists x \in V_M : d(i,x) \leq h\}$. To actually extract the subgraph, we simply traverse G starting from vertices in V_M, for h hops.

4.5 Node Embeddings for More Accuracy

In certain cases, the h-hop enclosing subgraph might miss some information about the motif in question (the details of is missed depend on a specific input graph and selected motif). To alleviate this, while simultaneously avoiding sampling a subgraph with large h, we also generate a node embedding $X_E \in \mathbb{R}^{n \times f}$ which encodes the information about more distant graph regions using random walks. For this, we employ the established node2vec [26] with the parameters from DeepWalk [46]. f is the dimension of the low-dimensional vector representation of a node. We generate such a node embedding once and then only append the embedding vectors (corresponding to the nodes in the extracted subgraph) to the feature matrix of each extracted subgraph.

4.6 Node Labeling for Structural Features

In order to provide our GNN with as much structural information as possible, we introduce two node labeling schemes. These schemes serve as structural learning features, and we use them when constructing feature matrices of the extracted subgraphs, fed into a GNN. Let s be the total number of vertices in the extracted subgraph G_s and k be the number of vertices forming the motif. We call the vertices in the respective samples (V_p or V_n) the *inner* vertices since they form a motif sample. The rest of the nodes in the subgraph G_s are called *outer* vertices.

The first label is simply an enumeration of all the inner vertices. We call this label the *inner label*. It enables ordering each vertex according to its role in the motif. For example, to predict a k-star, we always assign the inner label 1 to the star central vertex. This inner node label gets translated into a one-hot matrix $H \in \mathbb{N}^{k \times k}$; $H_{ij} = 1$ means that the i-th vertex in V_M receives label j. In order to include H into the feature matrix of the subgraph, we concatenate H with a zero matrix $0_{(s-k)k} \in \mathbb{N}^{(s-k) \times k}$, obtaining $X_H = (H \quad 0_{(s-k)k})^T$.

The second label is called the *outer* label. The label assigns to each outer vertex its distances to each inner vertex. Thus, each of the $s-k$ outer vertices get k labels. The first of these k labels describes the distance to the vertex with inner label 1. All these outer labels form a labeling matrix $L \in \mathbb{N}^{(s-k) \times k}$, appended with a zero matrix 0_{kk}, becoming $X_L = (0_{kk} \quad L)^T \in \mathbb{N}^{s \times k}$. The final feature matrix X_s of the respective subgraph G_s consists of X_H, X_L, the subgraph node embedding matrix X_E and the subgraph input feature matrix $X_{s_i} \in \mathbb{R}^{s \times d}$; we have $X_s = \begin{pmatrix} X_{s_i} & X_E & X_H & X_L \end{pmatrix} \in \mathbb{R}^{s \times (d+f+2k)}$; d is the dimension of the input feature vectors and f is the dimension of the node embedding vectors.

4.7 Different Orderings of Motif Vertices

SEAM supports predicting both motifs where vertices have preassigned specific roles, i.e., where vertices are **permutation dependant**, and motifs with vertices that are **permutation invariant**. The former enables the user to assign vertices meaningful different structural roles (e.g., star roots). The latter enables predicting motifs where the vertex order does not matter. For example, in a clique, the structural roles of all involved vertices are equivalent (i.e., these motifs are vertex-transitive). This is achieved by permuting the inner labels according to the applied vertex permutation.

4.8 Used Graph Neural Network Model

For our GNN model, we use the graph classification neural network DGCNN [62], used in SEAL [61, 63]. We now summarize its architecture. The first stage of this GNN consist of three graph convolution layers (GConv). Each layer distributes the vertex features of each vertex to its neighbors. Then, we feed the output of each of these GConv layers into a layer called k-sortpooling where all vertices are sorted based on their importance in the subgraph. After that, we apply a standard 1D convolution layer followed by a dense layer, followed by a softmax layer to get the prediction probabilities.

The input for our GNN model is the adjacency matrix of the selected h-hop enclosing subgraph $G^h_{V_s}$ together with the feature matrix X_s. With these inputs, we train our GNN model for 100

epochs. After each epoch, to validate the accuracy, we simply generate $G^h_{V_p}$ and $G^h_{V_n}$ as well as their feature matrix X_p and X_n from our samples in the validation dataset. We know for each set of vertices V_p or V_n, if they form the motif M. Thus, we can analyse the accuracy of our model by comparing the predictions with the original information about the motifs. Ultimately, we expect our model to predict the set of vertices V_p to form the motif M and the set of vertices V_n not to form the motif M.

4.9 Computational Complexity of SEAM

We discuss the time complexity of different parts of SEAM, showing that motif prediction in SEAM is computationally feasible even for large graphs and motifs. Assume that k, t, and d are #vertices in a motif, the number of mined given motifs per vertex, and the maximum degree in a graph, respectively.

First, extracting samples depends on a motif of interest: **positive sampling** takes $O(nd^k)$ (k-cliques), $O(tm)$ (k-stars), $O(nd^k)$ (k-db-stars), and $O(ndk^3)$ (dense clusters). These complexities assume mining *all* instances of respective motifs; SEAM further enables fixing the number of samples to find upfront, which lowers the complexities. **Negative sampling** (of a single instance) takes $O(dk)$ (k-cliques), $O(d)$ (k-stars), $O(d+k^2)$ (k-db-stars), and $O(ndk^3)$ (dense clusters). The complexities may be reduced is the user chooses to fix sampling counts. The h-**hop subgraph extraction**, and inner and outer **labeling**, take – respectively – $O(kd^{2h})$ and $O(k^2d^h)$ time per sample. Finally, **finding node embeddings** (with Node2Vec) and **training as well as inference** (with DGCNN) have complexities as described in the associated papers [26, 62]; they were illustrated to be feasible even for large datasets.

5 EVALUATION

We now illustrate the advantages of our correlated heuristics and of our learning architecture SEAM. We feature a representative set of results, extended results are in the appendix.

As **comparison targets**, we use motif prediction based on three link prediction heuristics (Jaccard, Common Neighbors, Adamic-Adar), and on the GNN based state-of-the-art SEAL link prediction [61, 63]. Here, the motif score is derived using a product of link scores with no link correlation ("Mul"). We also consider our correlated heuristics, using link scores, where each score is assigned the same importance ("Avg", $\mathbf{w} = \mathbf{1}\frac{1}{|E_{M,\mathcal{N}}|}$), or the *smallest* link score is assigned the *highest* importance ("Min"). *This gives a total of 12 comparison targets.* We then consider different variants of SEAM (e.g., with and without embeddings described in § 4.5). More details on the evaluation setting are in Figure 5 (on the right).

To assess **accuracy**, we use AUC (Area Under the Curve), a standard metric to evaluate the accuracy of any classification model in machine learning. We also consider a plain fraction of all correct predictions; these results resemble the AUC ones, see the appendix.

Details of **parametrization** and **datasets** are included in the appendix. In general, we use the same datasets as in the SEAL paper [63] for consistent comparisons; these are, among others, Yeast (protein-protein interactions), USAir (airline connections), and Power (a power grid). Overall, our current selection of tested motifs covers the whole motif spectrum in terms of their density: stars (**very sparse**), communities (**moderately sparse** and **dense**, depending on the threshold), and cliques (**very dense**).

We ensure that the used graphs match our motivation, i.e., they are either evolving or miss higher order structures that are then predicted. For this, we prepare the data so that different edges are removed randomly, imitating noise.

5.1 SEAM GNN vs. SEAL GNN vs. Heuristics

We compare (1) our heuristics from Section 3, (2) a scheme using the SEAL link prediction, and (3) our proposed SEAM GNN architecture. The results for k-**stars**, k-**cliques**, and k-**db-stars** (for networks USAir and Yeast) are in Figure 5 while **clusters** and **communities** are analyzed in Figure 6 (**"k-dense"** indicates a cluster of k vertices, with at least 90% of all possible edges present).

Behavior and Advantages of SEAM First, in Figure 5, we observe that the improvement in accuracy in SEAM almost always scales with the size of the motif. This shows that SEAM captures correlation between different edges (in larger motifs, there is more potential correlation between links). Importantly, the advantages and the capacity of SEAM to capture correlations, also hold in the presence of *deal-breaker edges* ("k-db-star"). Here, we assign links connecting pairs of star outer vertices as deal-breakers (e.g., 7-db-star is a 7-star with 15 deal-breaker edges connecting its arms with one another). We observe that the accuracy for k-stars with deal-breaker edges is lower than that for standard k-*stars*. Yet, SEAM is still the best baseline since it appropriately learns such edges and their impact on the motif appearance. The results in Figure 6 follow similar trends to Figure 5; SEAM outperforms all other baselines. Its accuracy also increases with the increasing motif size. *Overall, SEAM significantly outperforms both SEAL and heuristics in accuracy, and is the only scheme that captures higher-order characteristics, i.e., its accuracy increases with the amount of link correlations.*

Behavior and Advantages of Heuristics While the core result of our work is the superiority of SEAM in accuracy, our correlated heuristics ("Avg", "Min") also to a certain degree improve the motif prediction accuracy over methods that assume link independence ("Mul"). This behavior holds often in a statistically significant way, cf. Jaccard results for 3-cliques, 5-cliques, and 7-cliques. In several cases, the differences are smaller and fall within the standard deviations of respective schemes. Overall, we observe that $AUC_{Mul} < AUC_{Min} < AUC_{Avg}$ (except for k-db-stars). This shows that different aggregation schemes have different capacity in capturing the rich correlation structure of motifs. In particular, notice that "Min" is by definition (cf. Proposition 3.1) a lower bound of the score $s(M)$ defined in § 3.5.1. This implies that it is the smallest form of correlation that we can include in our motif score given the convex linear combination function proposed in § 3.5.1.

The main advantage of heuristics over SEAM (or SEAL) is that *they do not require training, and are thus overall faster.* For example, to predict 100k motif samples, the heuristics take around 2.2 seconds with a standard deviation of 0.05 seconds, while SEAM has a mean execution time (including training) of 1280 seconds with a standard deviation of 30 seconds. Thus, we conclude that heuristics could be preferred over SEAM when training overheads are deemed too high, and/or when the sizes of considered motifs (and thus the amount of link correlations) are small.

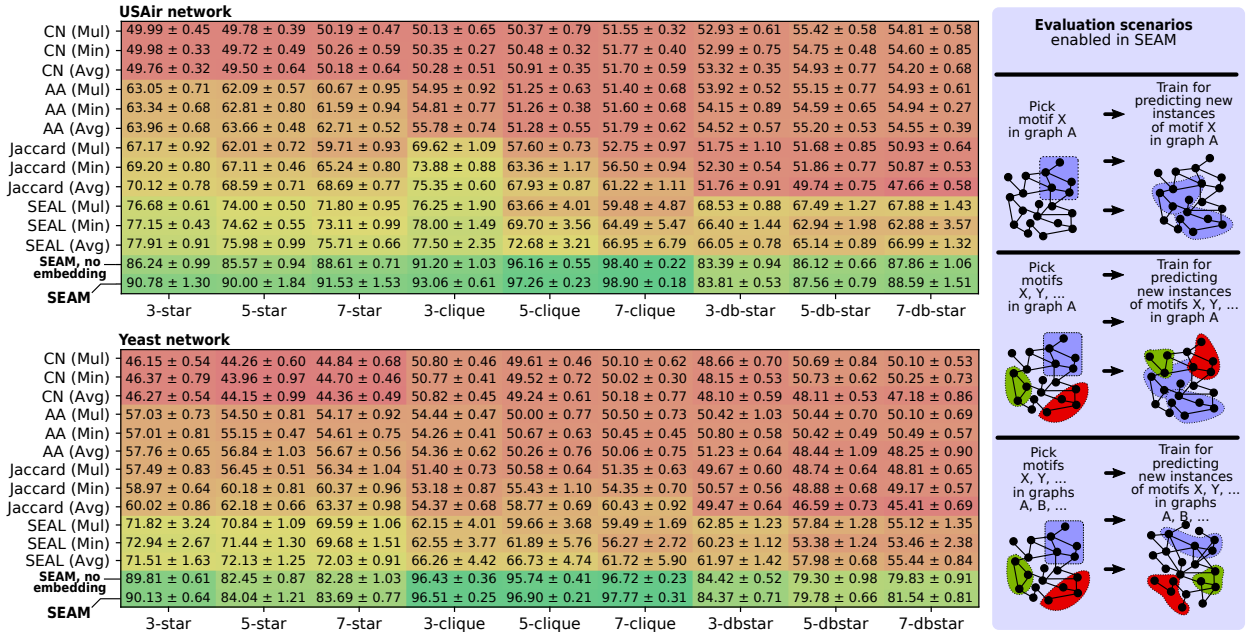


Figure 5: Comparison of different motif prediction schemes; SEAM is the proposed GNN based architecture. Other baselines use different link prediction schemes as building blocks; CN stands for Common Neighbors, AA stands for Adamic-Adar. We use graphs also used by the SEAL link prediction method [61, 63]. "k-db-star" indicate motifs with deal-breaker edges considered. In the presented data, we predict new instances of a given selected motif in a given graph dataset.

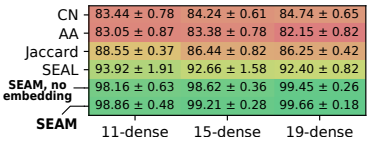


Figure 6: Comparison of prediction schemes as in Figure 5 for predicting dense subgraph motif described in § 4.3. All link prediction based schemes use the same motif score. We use the Yeast graph, also used by the SEAL link prediction method [61, 63].

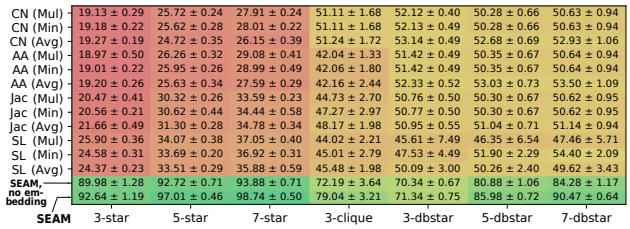


Figure 7: Comparison of different motif prediction schemes for a very sparse Power (power grid) graph. CN: Common Neighbors, AA: Adamic-Adar, Jac: Jaccard, SL: SEAL. "k-db-star": motifs with deal-breaker edges considered.

Interestingly, the Common Neighbors heuristic performs poorly. This is due to the similar neighborhoods of the edges that have to be predicted. The high similarity of these neighborhoods is caused by our subgraph extraction strategy discussed in Section 4.4, where we select the existing motif edges of the positive samples in such a way as to mimic the edge structure of the negative samples. These results show also that different heuristics do not perform equally with respect to the task of motif prediction and further studies are needed in this direction.

The accuracy benefits of SEAM over the best competitor (SEAL using the "Avg" way to compose link prediction scores into motif prediction scores) range from 12% to almost 32%. This difference is even larger for other methods; it is because there comparison targets cannot effectively capture link correlations in motifs. This result shows that the edge correlation in motifs is important to accurately predict a motif's appearance, and that it benefits greatly from being learned by a neural network.

5.2 Additional Analyses

We also consider a Power graph, see Figure 7. This graph dataset is very sparse with a very low average vertex degree of 2.7 (see the appendix for dataset details). SEAM again offers the best accuracy.

We also analyze the impact of additionally using Node2Vec node embeddings (cf. § 4.5). Interestingly, it consistently (by 0.2 – 4%) improves the accuracy while simultaneously *reducing the variance* in most cases by around 50% for cliques and dense clusters.

We also consider other aspects, for example, we vary the number of existing edges, and even eliminate all such edges; the results follow similar patterns to those observed above.

SEAM's running times heavily depend on the used model parameters. A full SEAM execution on the Yeast dataset with 40k training samples and 100 epochs typically takes 15–75 minutes (depending on the complexity of the motif, with stars and dense clusters being the fastest and slowest to process, respectively). The used hardware configuration includes an Intel 6130 @2.10GHz with 32 cores and an Nvidia V100 GPU; details are in the appendix.

Other analyses and ablation studies are in the appendix, they include varying the used labeling schemes, training dataset sizes, learning rates, epoch counts, or sizes of enclosing subgraphs.

6 RELATED WORK

Our work generalizes link prediction [4, 40] into arbitrary higher-order structures, considering inter-link correlations. Next, many works exist on listing, counting, or finding different patterns (also referred to as motifs, graphlets, or subgraphs) [12, 18, 30, 36, 47]. Our work enables predicting any of such patterns. Moreover, SEAM can use these schemes as subroutines when mining for specific samples. Third, different works analyze the temporal aspects of motifs [35, 53], for example by analyzing the temporal dynamics

of editor interactions [31], temporal dynamics of motifs in general time-dependent networks [34, 45], efficient counting of temporal motifs [39], predicting triangles [7, 43], or using motif features for more effective link predictions [1]. However, none of them considers prediction of general motifs. Moreover, there exists an extensive body of work on graph processing and algorithms, both static and dynamic (also called temporal, time-evolving, or streaming) [11, 14, 16, 24, 33, 48]. Still, they do not consider prediction of motifs.

Finally, GNNs have recently become a subject of intense research [58]. In this work, we use GNNs for making accurate predictions about motif appearance. While we pick DGCNN as a specific model to implement SEAM, other GNN models can also be used; such an analysis is an interesting direction for future work. We implement SEAM within the Pytorch Geometric GNN framework. Still, other GNN frameworks could also be used [22, 28, 37, 57, 60, 66].

7 CONCLUSION & DISCUSSION

Higher-order network analysis is an important approach for mining irregular data. Yet, it lacks methods and tools for predicting the evolution of the associated datasets. For this, we establish a problem of predicting general complex graph structures called motifs, such as cliques or stars. We illustrate its differences to simple link prediction, and then we propose heuristics for motif prediction that are invariant to the motif size and capture potential correlations between links forming a motif. Our analysis enables incorporating domain knowledge, and thus – similarly to link prediction – it can be a foundation for developing motif prediction schemes within specific domains.

While being fast, heuristics leave some space for improvements in prediction accuracy. To address this, we develop a graph neural network (GNN) architecture for predicting motifs. We show that it outperforms the state of the art by up to 32% in area under the curve, offering excellent accuracy, which *improves* with the growing size and complexity of the predicted motif. We also successfully apply our architecture to predicting more arbitrarily structured clusters, indicating its broader potential in mining irregular data.

Acknowledgements We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team for access to the Ault and Daint machines, and for technical support. We thank Timo Schneider for help with computing infrastructure at SPCL. This research received funding from Google European Doctoral Fellowship, Huawei, and the European Research Council (Project DAPP, No. 678880; Project PSAP, No. 101002047).

REFERENCES

[1] G. Abuoda et al. Link prediction via higher-order motif features. In *ECML PKDD*, 2019.
[2] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social networks*, 2003.
[3] M. Al Hasan et al. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, 2006.
[4] M. Al Hasan and M. J. Zaki. A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer, 2011.
[5] V. Batagelj and A. Mrvar. Pajek datasets, 2006. http://vlado.fmf.uni-lj.si/pub/networks/data/.
[6] A. R. Benson et al. Higher-order organization of complex networks. *Science*, 2016.
[7] A. R. Benson et al. Simplicial closure and higher-order link prediction. *PNAS*, 2018.
[8] M. Besta et al. To push or to pull: On reducing communication and synchronization in graph computations. In *ACM HPDC*, pages 93–104. ACM, 2017.
[9] M. Besta et al. Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics. In *ACM/IEEE Supercomputing*, pages 1–25, 2019.
[10] M. Besta et al. Communication-efficient jaccard similarity for high-performance distributed genome comparisons. In *IEEE IPDPS*, pages 1122–1132. IEEE, 2020.
[11] M. Besta et al. High-performance parallel graph coloring with strong guarantees on work, depth, and quality. In *ACM/IEEE Supercomputing*, 2020.
[12] M. Besta et al. Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra. *arXiv preprint arXiv:2103.03653*, 2021.
[13] M. Besta et al. Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. *arXiv preprint arXiv:2104.07582*, 2021.
[14] M. Besta et al. Practice of streaming processing of dynamic graphs: Concepts, models, and systems. *IEEE TPDS*, 2022.
[15] M. Bhattacharyya and S. Bandyopadhyay. Mining the largest quasi-clique in human protein interactome. In *EAIS*. IEEE, 2009.
[16] A. Buluç and J. R. Gilbert. The combinatorial blas: Design, implementation, and applications. *IJHPCA*, 25(4):496–509, 2011.
[17] W. Cao et al. A comprehensive survey on geometric deep learning. *IEEE Access*, 2020.
[18] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2, 2006.
[19] Z. Chen et al. Bridging the gap between spatial and spectral domains: A survey on graph neural networks. *arXiv preprint arXiv:2002.11867*, 2020.
[20] D. J. Cook and L. B. Holder. *Mining graph data*. John Wiley & Sons, 2006.
[21] CSCS. Swiss national supercomputing center, 2021. https://cscs.ch.
[22] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
[23] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR*, 2019.
[24] L. Gianinazzi et al. Communication-avoiding parallel minimum cuts and connected components. In *PPoPP*, volume 53, pages 219–232. ACM, ACM New York, NY, USA, 2018.
[25] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005.
[26] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.
[27] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167. ACM, 2004.
[28] Y. Hu et al. Featgraph: A flexible and efficient backend for graph neural network systems. *arXiv preprint arXiv:2008.11359*, 2020.
[29] S. Jabbour et al. Pushing the envelope in overlapping communities detection. In *IDA*, 2018.
[30] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105, 2013.
[31] D. Jurgens and T.-C. Lu. Temporal motifs reveal the dynamics of editor interactions in wikipedia. In *AAAI ICWSM*, volume 6, 2012.
[32] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 1953.
[33] J. Kepner et al. Mathematical foundations of the graphblas. In *IEEE HPEC*, 2016.
[34] L. Kovanen et al. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.
[35] L. Kovanen et al. Temporal motifs. In *Temporal networks*, pages 119–133. 2013.
[36] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.
[37] S. Li et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.
[38] X.-L. Li et al. Interaction graph mining for protein complexes using local clique merging. *Genome Informatics*, 2005.
[39] P. Liu et al. Sampling methods for counting temporal motifs. In *WSDM*, 2019.
[40] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.
[41] V. Martínez, F. Berzal, and J.-C. Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016.
[42] P. Moritz et al. Ray: A distributed framework for emerging ai applications. *arXiv preprint arXiv:1712.05889*, 2017.
[43] H. Nassar, A. R. Benson, and D. F. Gleich. Pairwise link prediction. In *ASONAM*, 2019.
[44] H. Nassar, A. R. Benson, and D. F. Gleich. Neighborhood and pagerank methods for pairwise link prediction. *Social Network Analysis and Mining*, 2020.
[45] A. Paranjape, A. R. Benson, and J. Leskovec. Motifs in temporal networks. In *WSDM*, 2017.
[46] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.
[47] T. Ramraj and R. Prabhakar. Frequent subgraph mining algorithms-a survey. *Procedia Computer Science*, 47:197–204, 2015.
[48] S. Sakr et al. The future is big graphs! a community view on graph processing systems. *arXiv preprint arXiv:2012.06171*, 2020.
[49] R. Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.
[50] F. Scarselli et al. The graph neural network model. *IEEE TNN*, 2008.
[51] B. Taskar et al. Link prediction in relational data. In *NeurIPS*, 2004.
[52] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.
[53] S. Torkamani and V. Lohweg. Survey on time series motif discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2):e1199, 2017.
[54] C. Von Mering et al. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002.
[55] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 1998.
[56] S. Wu, F. Sun, W. Zhang, and B. Cui. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260*, 2020.
[57] Y. Wu et al. Seastar: vertex-centric programming for graph neural networks. In *EuroSys*, 2021.
[58] Z. Wu et al. A comprehensive survey on graph neural networks. *IEEE TNNLS*, 2020.
[59] C. Zhang, D. Song, et al. Heterogeneous graph neural network. In *KDD*, 2019.
[60] D. Zhang et al. Agl: a scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454*, 2020.
[61] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018.
[62] M. Zhang et al. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.
[63] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Revisiting graph neural networks for link prediction. *arXiv preprint arXiv:2010.16103*, 2020.
[64] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *IEEE TKDM*, 2020.
[65] J. Zhou et al. Graph neural networks: A review of methods and applications. *AI Open*, 2020.
[66] R. Zhu et al. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.

APPENDIX A: PROOFS

We recall the statement of **Observation 1** in Section 3.1:

Consider vertices $v_1, ..., v_k \in V$. Assuming no edges already connecting $v_1, ..., v_k$, there are $2^{\binom{k}{2}} - 1$ motifs (with between 1 and $\binom{k}{2}$ edges) that can appear to connect $v_1, ..., v_k$.

Proof. We denote as $E_k = \{\{i, j\} : i, j \in V_k \wedge i \neq j\}$ the edge set of the undirected subgraph (V_k, E_k) with $V_k \subseteq V$. The number of all possible edges between k vertices is $|E_k| = \binom{k}{2}$. Any subset of E_k, with the exception of the empty set, defines a motif. Thus the set of all possible subsets (i.e., the *power set* $\mathcal{P}$) of E_k is the set of motifs. Then, since $|\mathcal{P}(E_k)| = 2^{\binom{k}{2}}$, we subtract the empty set (which we consider as an invalid motif) from the total count to obtain the desired result. □

We recall the statement of **Proposition 3.1** in Section 3.5:

Let $\{x_1, ..., x_n\}$ be any finite collection of elements from $U = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$. Then, $\forall n \in \mathbb{N}$ we have $\prod_{i=1}^{n} x_i \leq \sum_{i=1}^{n} w_i x_i$, where $w_i \geq 0$ $\forall i \in \{1, ..., n\}$ and subject to the constraint $\sum_{i=1}^{n} w_i = 1$.

Proof. We start by noticing that $\prod_{i=1}^{n} x_i \leq \min\{x_1, ..., x_n\}$. This is trivial to verify if $\exists\ x_i = 0$ for $i \in \{1, ..., n\}$. Otherwise, it can be shown by contradiction: imagine that $\prod_{i=1}^{n} x_i > \min\{x_1, ..., x_n\}$. We know that U is closed with respect to the product (i.e., $\prod_{i=1}^{n} x_i \in U\ \forall\ n \in \mathbb{N}$). Then, we can divide both sides by $\min\{x_1, ..., x_n\}$, since we ruled out the division by zero, to obtain $\prod_{i=1}^{n-1} x_i > 1$. This implies $\prod_{i=1}^{n-1} x_i \notin U$, which contradicts that U is closed to the product. For the right side of the original statement, we know by definition that $x_i \geq \min\{x_1, ..., x_n\}$ $\forall i \in \{1, ..., n\}$. Since $w_i \geq 0$, we can also write that $w_i x_i \geq w_i \min\{x_1, ..., x_n\}$ $\forall i \in \{1, ..., n\}$. Thus, since U is an ordered set, we can state that $\sum_{i=1}^{n} w_i x_i \geq \sum_{i=1}^{n} w_i \min\{x_1, ..., x_n\}$. But then, since $\sum_{i=1}^{n} w_i \min\{x_1, ..., x_n\} = \min\{x_1, ..., x_n\}$, we conclude that $\min\{x_1, ..., x_n\} \leq \sum_{i=1}^{n} w_i x_i$. This ends the proof thanks to the transitive property. □

We also justify some **complexity bounds** from § 4.9. Mining k-stars is independent of k. To find a k-star at a given node x, one chooses $k-1$ random nodes of x. This has a complexity of $O(k+d(x))$. If k is larger than $d(x)$, there is no star and one can skip the node in $O(1)$. Otherwise, it takes $O(d(x))$ to extract a star. Thus, for a given star, we extract t samples in $O(td(x))$. Summing over all nodes is hence $O(tm)$. Next, the bounds for cliques and clusters are straightforward. Finally, for the enclosing subgraph extraction and labeling, we do a BFS starting from each of the k motif vertices that visits all h-hop neighbors. Each node has at most d neighbors, hence there is at most kd^h nodes in the h-hop neighborhood that need to be visited. But, as BFS is also linear in the number of edges, the complexity is $O(kd^{2h})$. The inner labels are a one-hot-encoding of the motif vertices, which can be produced in $O(k^2 d^h)$ time. The outer labels are the distances to the motif nodes, which can be computed at the same time as the BFS traversal for the extraction, so it is the overhead of $O(k^2 d^h)$.

APPENDIX B: DETAILS OF DATASETS

In this section, we provide additional details on the various datasets that we used. We selected networks of different origins (biological, engineering, transportation), with different structural properties (different sparsities and skews in degree distributions).

USAir [5] is a graph with 332 vertices and 2,126 edges representing US cities and the airline connections between them. The vertex degrees range from 1 to 139 with an average degree of 12.8. Yeast [54] is a graph of protein-protein interactions in yeast with 2,375 vertices and 11,693 edges. The vertex degrees range from 1 to 118 with an average of 9.8. Power [55] is the electrical grid of the Western US with 4,941 vertices and 6,594 edges. The vertex degrees range from 1 to 19 with an average degree of 2.7.

Learning Rate	3-star	5-star	7-star	3-clique	5-clique	7-clique
0.000125	80.89 ± 0.91	82.08 ± 0.64	85.35 ± 0.11	84.07 ± 0.95	85.08 ± 1.81	87.56 ± 1.18
0.00025	81.61 ± 0.82	84.68 ± 1.15	87.53 ± 0.36	84.70 ± 0.77	88.01 ± 1.76	91.17 ± 2.35
0.0005	83.80 ± 1.05	86.06 ± 0.42	89.55 ± 0.93	86.03 ± 0.97	91.31 ± 1.11	94.79 ± 1.02
0.001	84.50 ± 0.45	87.18 ± 0.62	90.83 ± 0.05	87.46 ± 0.98	93.64 ± 0.80	96.98 ± 0.53
0.002	86.26 ± 0.66	86.91 ± 0.93	90.42 ± 0.46	88.22 ± 0.53	94.77 ± 0.82	97.80 ± 0.78
0.004	86.92 ± 1.50	88.01 ± 2.25	88.04 ± 1.16	88.12 ± 1.30	94.76 ± 1.22	93.76 ± 8.11
0.008	83.87 ± 2.94	81.85 ± 4.97	84.32 ± 2.19	87.43 ± 2.45	81.89 ± 10.59	76.37 ± 11.53
0.016	71.03 ± 7.58	65.56 ± 5.81	63.58 ± 2.35	76.28 ± 8.69	66.94 ± 14.02	57.13 ± 5.09
0.032	54.14 ± 5.73	54.94 ± 9.88	50.00 ± 0.00	58.18 ± 9.06	56.33 ± 2.30	58.95 ± 7.42

Figure 8: AUC-Score comparison for different learning rates on USAir graph. Number of epochs = 50, training dataset size = 100,000.

Learning Rate	3-star	5-star	7-star	3-clique	5-clique	7-clique
0.000125	82.46 ± 0.92	85.01 ± 0.56	88.56 ± 0.04	85.35 ± 0.82	87.99 ± 1.69	90.85 ± 0.64
0.00025	83.48 ± 0.59	86.83 ± 1.27	90.11 ± 0.20	86.28 ± 0.75	91.33 ± 1.28	94.61 ± 1.58
0.0005	85.62 ± 1.05	88.05 ± 0.21	91.53 ± 0.42	87.92 ± 0.83	94.75 ± 0.81	97.26 ± 0.47
0.001	86.58 ± 0.99	88.70 ± 0.57	92.27 ± 0.20	89.95 ± 1.34	95.77 ± 0.54	98.50 ± 0.37
0.002	88.40 ± 1.50	88.75 ± 0.96	91.79 ± 0.39	90.30 ± 0.43	96.78 ± 0.97	98.67 ± 0.31
0.004	89.68 ± 1.02	91.34 ± 2.97	90.93 ± 0.61	92.06 ± 2.70	96.62 ± 1.15	94.09 ± 8.94
0.008	86.72 ± 1.74	86.01 ± 2.49	88.13 ± 1.27	91.44 ± 3.06	90.45 ± 10.07	85.74 ± 8.56
0.016	75.29 ± 10.46	66.72 ± 7.92	63.58 ± 2.35	78.91 ± 11.09	70.96 ± 17.71	59.60 ± 11.00
0.032	55.32 ± 5.90	55.40 ± 10.80	50.00 ± 0.00	58.18 ± 9.06	56.25 ± 2.28	60.44 ± 6.98

Figure 9: AUC-Score comparison for different learning rates on USAir graph. Number of epochs = 100, training dataset size = 100,000.

Learning Rate	3-star	5-star	7-star	3-clique	5-clique	7-clique
0.000125	83.50 ± 0.98	86.65 ± 0.29	90.12 ± 0.11	86.25 ± 0.72	90.06 ± 1.20	93.14 ± 0.58
0.00025	84.41 ± 0.75	87.66 ± 0.20	91.27 ± 0.41	87.46 ± 0.60	93.39 ± 1.13	96.42 ± 0.84
0.0005	86.47 ± 0.74	88.82 ± 0.39	92.23 ± 0.50	89.23 ± 0.60	95.96 ± 0.63	98.09 ± 0.27
0.001	88.00 ± 0.94	89.06 ± 0.47	92.72 ± 0.30	91.26 ± 1.25	96.46 ± 0.44	98.89 ± 0.33
0.002	89.43 ± 1.58	90.17 ± 1.52	92.37 ± 0.17	91.49 ± 1.43	97.26 ± 0.74	98.87 ± 0.31
0.004	91.52 ± 0.56	93.47 ± 2.44	91.30 ± 0.30	93.66 ± 3.01	97.33 ± 0.78	97.84 ± 1.23
0.008	87.46 ± 1.74	90.53 ± 3.21	90.39 ± 0.32	92.76 ± 2.78	93.93 ± 7.25	87.15 ± 9.33
0.016	76.81 ± 11.74	67.98 ± 10.33	63.58 ± 2.35	80.29 ± 11.41	72.57 ± 17.40	59.65 ± 10.99
0.032	55.32 ± 5.90	56.99 ± 12.11	50.00 ± 0.00	58.18 ± 9.05	56.25 ± 2.28	60.44 ± 6.98

Figure 10: AUC-Score comparison for different learning rates on USAir graph. Number of epochs = 150, training dataset size = 100,000.

APPENDIX C: SEAM MODEL PARAMETERS

Choosing learning rate and number of epochs

To find the optimal learning rate for SEAM we try different learning rates as shown in Figures 8, 9 and 10. The associated hyperparameters are highly dependent on the specific motif to be predicted and on the used dataset. As an example, we analyze the hyperparameters for k-stars and k-cliques on the USAir graph dataset. The plots show that there is a sweet spot for the learning rate at 0.001-0.002. Any value below that rate is too small and our model cannot train its neural network effectively, while for the values above that, the model is unable to learn the often subtle differences between hard negative samples and positive samples. The number of epochs of the learning process can be chosen according to the available computational resources of the user.

Analysis of different training dataset sizes

Figure 11 shows the different accuracy results of SEAM, for different motifs and training dataset sizes. We observe that the accuracy strongly depends on the motif to be predicted. For example, a dense subgraph can be predicted with high accuracy with only 100 training samples. On the other hand, prediction accuracy of the 5-star motif improves proportionally to the amount of training samples while still requiring more samples (than plain dense subgraphs) for a high accuracy score. For all motifs, we set our minimal amount of training samples to 20,000 for positive and for negative ones.

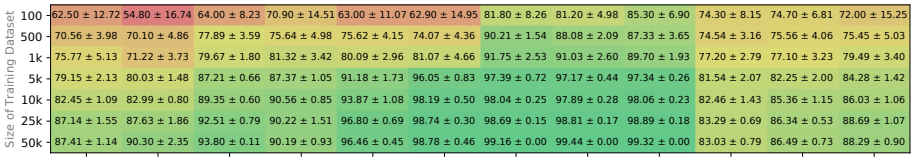


Figure 11: AUC-Score comparison for different training dataset sizes on **USAir** graph. Learning rate = 0.002, number of epochs = 100.

APPENDIX D: ANALYSIS OF DIFFERENT VARIANTS OF MOTIF PREDICTION IN SEAM

Here, we analyze the effects and contributions from different variants of SEAM. First, we investigate the accuracy improvements due to our proposed labeling scheme in Section 4.6. Then, we empirically justify our approach to only sample the h-hop enclosing subgraph for small h (1–2). Finally, we evaluate the performance of every prediction method if there are no motif edges already present.

Labeling Scheme vs. Accuracy

Figure 12 shows that our proposed labeling scheme generally has a positive impact on the accuracy of SEAM. The exception is the k-star motif. For $k = 3$, the labeling scheme significantly improves the accuracy. On the other hand, using $k > 3$ reduces the accuracy while simultaneously increasing the variance of test results. This effect can be explained with the implementation details of our labeling scheme. We remove every edges between all the motif vertices to calculate our k-dimensional distance labels. This procedure seems to misrepresent the structure of k-stars for $k > 3$. There are possible improvements to be gained in future work by further optimizing our labeling scheme.

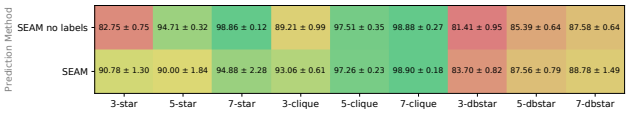


Figure 12: Effect of our proposed labeling scheme on **USAir** graph. h-hop = 1, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000.

h-Hop Enclosing Subgraphs vs. Accuracy

Zhang et al. [61] motivated the use of small h-hop neighborhoods for SEAL with the γ-decaying heuristic. We now provide additional data to backup this decision in SEAM. Figures 14 and 13 show that in most cases there is not much performance to be gained by sampling an h-hop enclosing subgraph with $h > 2$. This effect is especially striking for sparse graph datasets like the Power shown in Figure 14. The accuracy starts to drop significantly for $h > 2$. The only outlier in our little test was the 5-star motif shown in Figure 13. This effect was most likely caused by the specifics of this particular dataset and it does reflect a trend for other graphs. An additional explanation could also be the non-optimal labeling implementation for the 5-star motif. These special cases do not justify to increase the neighborhood size of the motif in a general case.

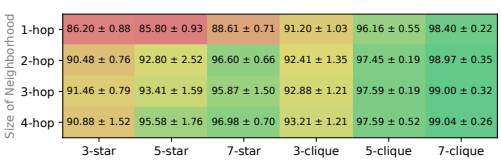


Figure 13: Comparison of different h-hop enclosing subgraphs used in SEAM, for the **USAir** graph. Learning rate = 0.002, number of epochs = 100.

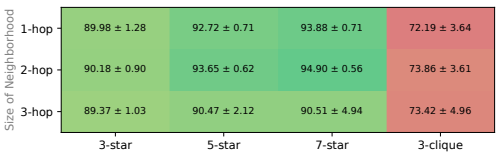


Figure 14: Comparison of different h-hop enclosing subgraphs used in SEAM, for the **Power** graph. Learning rate = 0.002, number of epochs = 100, training dataset size = 100,000. The graph does not contain enough 5-cliques and 7-cliques due to its sparsity.

Presence of Motif Edges vs. Accuracy

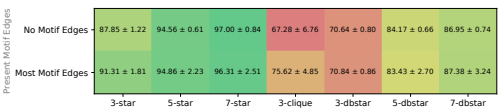


Figure 15: Comparison of the prediction accuracy of SEAM for different already present motif edges for the **Power** graph. h-hop = 1, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000. The graph does not contain enough 5-cliques and 7-cliques due to its sparsity.

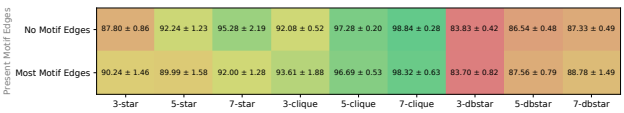


Figure 16: Comparison of the prediction accuracy of SEAM for different already present motif edges for the **USAir** graph. h-hop = 1, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000.

We now illustrate that SEAM also ensures high accuracy when *no* or *very few* motif edges are already present, see Figures 15 and 16. Thus, we can conclude that SEAM's prediction strength relies mostly on the structure of the neighborhood subgraph, embeddings, vertex attributes, and our proposed labeling scheme, and not necessarily on whether a given motif is already partially present. Outliers in this experiment are the 3–clique in the Power graph, the k-star motif with $k > 3$ in the USAir graph, and the 3-star motif in general. Still, there is no general tendency indicating that SEAM would profit greatly from the presence of most motif edges.

APPENDIX F: DETAILS OF IMPLEMENTATION & USED HARDWARE

Our implementation of SEAM and SEAL use the PyTorch Geometric Library [23]. We employ Ray [42] for distributed sampling and preprocessing, and RaySGD for distributed training and inference.

To run our experiments, we used the AULT cluster and the Piz Daint cluster at CSCS [21]. For smaller tasks, we used nodes from the AULT cluster such as AULT9/10 (64 AMD EPYC 7501 @ 2GHz processors, 512 GB memory and 4 Nvidia V100 GPUs), AULT23/24 (32 Intel Xeon 6130 @ 2.10GHz processors, 1.5TB memory and 4 Nvidia V100 GPUs), and AULT25 (128 AMD EPYC 7742 @ 2.25GHz processors, 512 GB memory and 4 Nvidia A100 GPUs). For larger, tasks we used our distributed implementation on the Piz Daint cluster (5704 compute nodes, each with 12 Intel Xeon E5-2690 v3 @ 2.60GHz processors, 64 GB memory and a Nvidia Tesla P100 GPU).

TRIANGLE AND FOUR CYCLE COUNTING WITH PREDICTIONS IN GRAPH STREAMS

Justin Y. Chen, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, and Sandeep Silwal *
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{justc, indyk, shyamsn, ronitt, silwal}@mit.edu

Honghao Lin, David P. Woodruff, Michael Zhang
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{honghaol, dwoodruf, jinyaoz}@andrew.cmu.edu

Tal Wagner
Microsoft Research
Redmond, WA 98052, USA
tal.wagner@gmail.com

Talya Eden
Massachusetts Institute of Technology and Boston University
Cambridge, MA 02139, USA
teden@mit.edu

ABSTRACT

We propose data-driven one-pass streaming algorithms for estimating the number of triangles and four cycles, two fundamental problems in graph analytics that are widely studied in the graph data stream literature. Recently, Hsu et al. (2019a) and Jiang et al. (2020) applied machine learning techniques in other data stream problems, using a trained oracle that can predict certain properties of the stream elements to improve on prior "classical" algorithms that did not use oracles. In this paper, we explore the power of a "heavy edge" oracle in multiple graph edge streaming models. In the adjacency list model, we present a one-pass triangle counting algorithm improving upon the previous space upper bounds without such an oracle. In the arbitrary order model, we present algorithms for both triangle and four cycle estimation with fewer passes and the same space complexity as in previous algorithms, and we show several of these bounds are optimal. We analyze our algorithms under several noise models, showing that the algorithms perform well even when the oracle errs. Our methodology expands upon prior work on "classical" streaming algorithms, as previous multi-pass and random order streaming algorithms can be seen as special cases of our algorithms, where the first pass or random order was used to implement the heavy edge oracle. Lastly, our experiments demonstrate advantages of the proposed method compared to state-of-the-art streaming algorithms.

1 INTRODUCTION

Counting the number of cycles in a graph is a fundamental problem in the graph stream model (e.g., Atserias et al. (2008); Bera & Chakrabarti (2017); Seshadhri et al. (2013); Kolountzakis et al. (2010); Bar-Yossef et al. (2002); Kallaugher et al. (2019)). The special case of counting triangles is widely studied, as it has a vast range of applications. In particular, it provides important insights into the structural properties of networks (Prat-Pérez et al., 2012; Farkas et al., 2011), and is used to discover motifs in protein interaction networks (Milo et al., 2002), understand social networks (Foucault Welles et al., 2010), and evaluate large graph models (Leskovec et al., 2008). See Al Hasan & Dave (2018) for a survey of these and other applications.

*All authors contributed equally.

Because of its importance, a large body of research has been devoted to space-efficient streaming algorithms for $(1+\epsilon)$-approximate triangle counting. Such algorithms perform computation in one or few passes over the data using only a sub-linear amount of space. A common difficulty which arises in all previous works is the existence of *heavy edges*, i.e., edges that are incident to many triangles (four cycles). As sublinear space algorithms often rely on sampling of edges, and since a single heavy edge can greatly affect the number of triangles (four cycles) in a graph, sampling and storing these edges are often the key to an accurate estimation. Therefore, multiple techniques have been developed to determine whether a given edge is heavy or not.

Recently, based on the observation that many underlying patterns in real-world data sets do not change quickly over time, machine learning techniques have been incorporated into the data stream model via the training of heavy-hitter oracles. Given access to such a learning-based oracle, a wide range of significant problems in data stream processing — including frequency estimation, estimating the number of distinct elements, F_p-Moments or (k,p)-Cascaded Norms — can all achieve space bounds that are better than those provided by "classical" algorithms, see, e.g., Hsu et al. (2019a); Cohen et al. (2020); Jiang et al. (2020); Eden et al. (2021); Du et al. (2021). More generally, learning-based approaches have had wide success in other algorithmic tasks, such as data structures (Kraska et al., 2018; Ferragina et al., 2020; Mitzenmacher, 2018; Rae et al., 2019; Vaidya et al., 2021), online algorithms (Lykouris & Vassilvtiskii, 2018; Purohit et al., 2018; Gollapudi & Panigrahi, 2019; Rohatgi, 2020; Wei, 2020; Mitzenmacher, 2020; Lattanzi et al., 2020; Bamas et al., 2020), similarity search (Wang et al., 2016; Dong et al., 2020) and combinatorial optimization (Dai et al., 2017; Balcan et al., 2017; 2018a;b; 2019). See the survey and references therein for additional works (Mitzenmacher & Vassilvitskii, 2020).

Inspired by these recent advancements, we ask: *is it possible to utilize a learned heavy edge oracle to improve the space complexity of subgraph counting in the graph stream model?* Our results demonstrate that the answer is yes.

1.1 Our Results and Comparison to Previous Theoretical Works

We present theoretical and empirical results in several graph streaming models, and with several notions of prediction oracles. Conceptually, it is useful to begin by studying *perfect oracles* that provide exact predictions. While instructive theoretically, such oracles are typically not available in practice. We then extend our theoretical results to *noisy oracles* that can provide inaccurate or wrong predictions. We validate the practicality of such oracles in two ways: by directly showing they can be constructed for multiple real datasets, and by showing that on those datasets, our algorithms attain significant empirical improvements over baselines, when given access to these oracles.

We proceed to a precise account of our results. Let $G = (V, E)$ denote the input graph, and let n, m, and T denote the number of vertices, edges, and triangles (or four-cycles) in G, respectively. There are two major graph edge streaming models: the *adjacency list* model and the *arbitrary order* model. We show that training heavy edge oracles is possible in practice in both models, and that such oracles make it possible to design new algorithms that significantly improve the space complexity of triangle and four-cycle counting, both in theory and in practice. Furthermore, our formalization of a heavy edge prediction framework makes it possible to show provable lower bounds as well. Our results are summarized in Table 1.

In our algorithms, we assume that we know a large-constant approximation of T for the purposes of setting various parameters. This is standard practice in the subgraph counting streaming literature (e.g., see (Braverman et al., 2013, Section 1), (McGregor et al., 2016, Section 1.2) for an extensive discussions on this assumption). Moreover, when this assumption cannot be directly carried over in practice, in Subsection F.3 we discuss how to adapt our algorithms to overcome this issue.

1.1.1 Perfect Oracle

Our first results apply for the case that the algorithms are given access to a perfect heavy edge oracle. That is, for some threshold ρ, the oracle perfectly predicts whether or not a given edge is incident to at least ρ triangles (four cycles). We describe how to relax this assumption in Section 1.1.2.

Adjacency List Model All edges incident to the same node arrive together. We show:

Table 1: Our results compared to existing theoretical algorithms. Δ_E (Δ_V) denotes the maximum number of triangles incident to any edge (vertex), and κ denotes the arboricity of the graph.

Problem	**Previous Results (no oracle)**	**Our Results**
Triangle, Adjacency	$\widetilde{O}(\epsilon^{-2}m/\sqrt{T})$, 1-pass (McGregor et al., 2016)	$\widetilde{O}(\min(\epsilon^{-2}m^{2/3}/T^{1/3}, \epsilon^{-1}m^{1/2}))$, 1-pass
Triangle, Arbitrary	$\widetilde{O}(\epsilon^{-2}m^{3/2}/T)$, 3-pass (McGregor et al., 2016)	$O(\epsilon^{-1}(m/\sqrt{T}+m^{1/2}))$, 1-pass
	$\widetilde{O}(\epsilon^{-2}m/\sqrt{T})$, 2-pass (McGregor et al., 2016)	
	$\widetilde{O}(\epsilon^{-2}(m/\sqrt{T} + m\Delta_E/T))$, 1-pass (Pagh & Tsourakakis, 2012)	
	$\widetilde{O}(\epsilon^{-2}(m/T^{2/3} + m\Delta_E/T + m\sqrt{\Delta_V}/T))$, 2-pass (Kallaugher & Price, 2017)	
	$\widetilde{O}(\mathrm{poly}(\epsilon^{-1})(m\Delta_E/T + m\sqrt{\Delta_V}/T))$, 1-pass (Jayaram & Kallaugher, 2021)	
	$\widetilde{O}(\mathrm{poly}(\epsilon^{-1})m\kappa/T)$, multi-pass (Bera & Sheshadhri, 2020)	
4-cycle, Arbitrary	$\widetilde{O}(\epsilon^{-2}m/T^{1/4})$, 3-pass (McGregor & Vorotnikova, 2020)	$\widetilde{O}(T^{1/3} + \epsilon^{-2}m/T^{1/3})$, 1-pass
	$\widetilde{O}(\epsilon^{-2}m/T^{1/3})$, 3-pass (Vorotnikova, 2020)	

Theorem 1.1. *There exists a one-pass algorithm, Algorithm 1, with space complexity*[1] $\widetilde{O}(\min(\epsilon^{-2}m^{2/3}/T^{1/3}, \epsilon^{-1}m^{1/2}))$ *in the adjacency list model that, using a learning-based oracle, returns a* $(1 \pm \epsilon)$*-approximation to the number* T *of triangles with probability at least*[2] $7/10$.

An overview of Algorithm 1 is given in Section 2, and the full analysis is provided in Appendix B.

Arbitrary Order Model In this model, the edges arrive in the stream in an arbitrary order. We present a one-pass algorithm for triangle counting and another one-pass algorithm for four cycle counting in this model, both reducing the number of passes compared to the currently best known space complexity algorithms. Our next result is as follows:

Theorem 1.2. *There exists a one-pass algorithm, Algorithm 4, with space complexity* $\widetilde{O}(\epsilon^{-1}(m/\sqrt{T} + \sqrt{m}))$ *in the arbitrary order model that, using a learning-based oracle, returns a* $(1 \pm \epsilon)$*-approximation to the number* T *of triangles with probability at least* $7/10$.

An overview of Algorithm 4 is given in Section 3, and full details are provided in Appendix C.

We also show non-trivial space lower bounds that hold even if appropriate predictors are available. In Theorem C.2 in Appendix C.3, we provide a lower bound for this setting by giving a construction that requires $\Omega(\min(m/\sqrt{T}, m^{3/2}/T))$ space even with the help of an oracle, proving that our result is nearly tight in some regimes. Therefore, the triangle counting problem remains non-trivial even when extra information is available.

Four Cycle Counting. For four cycle counting in the arbitrary order model, we give Theorem 1.3 which is proven in Appendix D.

Theorem 1.3. *There exists a one-pass algorithm, Algorithm 5, with space complexity* $\widetilde{O}(T^{1/3} + \epsilon^{-2}m/T^{1/3})$ *in the arbitrary order model that, using a learning-based oracle, returns a* $(1 \pm \epsilon)$*-approximation to the number* T *of four cycles with probability at least* $7/10$.

To summarize our theoretical contributions, for the first set of results of counting triangles in the adjacency list model, our bounds always improve on the previous state of the art due to McGregor et al. (2016) for all values of m and T. For a concrete example, consider the case that $T = \Theta(\sqrt{m})$.

[1] We use $\widetilde{O}(f)$ to denote $O(f \cdot \mathrm{polylog}(f))$.

[2] The success probability can be $1 - \delta$ by running $\log(1/\delta)$ copies of the algorithm and taking the median.

In this setting previous bounds result in an $\widetilde{O}(m^{3/4})$-space algorithm, while our algorithm only requires $\widetilde{O}(\sqrt{m})$ space (for constant ϵ).

For the other two problems of counting triangles and 4-cycles in the arbitrary arrival model, our space bounds have an additional additive term compared to McGregor et al. (2016) (for triangles) and Vorotnikova (2020) (for 4-cycles) but importantly run in a **single pass** rather than multiple passes. In the case where the input graph has high triangles density, $T = \Omega(m/\epsilon^2)$, our space bound is worse due to the additive factor. When $T = O(m/\epsilon^2)$, our results achieve the same dependence on m and T as that of the previous algorithms with an improved dependency in ϵ. Moreover, the case $T \leq m/\epsilon^2$ is natural for many real world datasets: for $\epsilon = 0.05$, this condition holds for all of the datasets in our experimental results (see Table 2). Regardless of the triangle density, a key benefit of our results is that they are achieved in a single pass rather than multiple passes. Finally, our results are for general graphs, and make no assumptions on the input graph (unlike Pagh & Tsourakakis (2012); Kallaugher & Price (2017); Bera & Sheshadhri (2020)). Most of our algorithms are relatively simple and easy to implement and deploy. At the same time, some of our results require the use of novel techniques in this context, such as the use of exponential random variables (see Section 2).

1.1.2 Noisy Oracles

The aforementioned triangle counting results are stated under the assumption that the algorithms are given access to perfect heavy edge oracles. In practice, this assumption is sometimes unrealistic. Hence, we consider several types of noisy oracles. The first such oracle, which we refer to as a K-noisy oracle, is defined below (see Figure 3 in the Supplementary Section C.2).

Definition 1.1. *For an edge $e = xy$ in the stream, define N_e as the number of triangles that contain both x and y. For a fixed constant $K \geq 1$ and for a threshold ρ we say that an oracle O_ρ is a* K-noisy *oracle if for every edge e, $1 - K \cdot \frac{\rho}{N_e} \leq Pr[O_\rho(e) = \text{HEAVY}] \leq K \cdot \frac{N_e}{\rho}$.*

This oracle ensures that if an edge is extremely heavy or extremely light, it is classified correctly with high probability, but if the edge is close to the threshold, the oracle may be inaccurate. We further discuss the properties of this oracle in Section G.

For this oracle, we prove the following two theorems. First, in the adjacency list model, we prove:

Theorem 1.4. *Suppose that the oracle given to Algorithm 1 is a K-noisy oracle as defined in Definition 1.1. Then with probability $2/3$, Algorithm 1 returns a value in $(1 \pm \sqrt{K} \cdot \epsilon)T$, and uses space at most $\widetilde{O}(\min(\epsilon^{-2}m^{2/3}/T^{1/3}, K \cdot \epsilon^{-1}m^{1/2}))$.*

Hence, even if our oracle is inaccurate for edges near the threshold, our algorithm still obtains an effective approximation with low space in the adjacency list model. Likewise, for the arbitrary order model, we prove in Theorem C.1 that the $O(\epsilon^{-1}(m/\sqrt{T} + \sqrt{m}))$ 1-pass algorithm of Theorem 1.2 also works when Algorithm 4 is only given access to a K-noisy oracle.

The proof of Theorem 1.4 is provided in Appendix B.1, and the proof of Theorem C.1 is provided in Appendix C.2. We remark that Theorems 1.4 and C.1 automatically imply Theorems 1.1 and 1.2, since the perfect oracle is automatically a K-noisy oracle.

(Noisy) Value Oracles In the adjacency list model, when we see an edge xy, we also have access to all the neighbors of either x or y, which makes it possible for the oracle to give a more accurate prediction. For an edge xy, let R_{xy} denote the number of triangles $\{x, z, y\}$ so that x precedes z and z precedes y in the stream arrival order. Formally, $R_{xy} = |z : \{x, y, z\} \in \Delta \text{ and } x <_s z <_s y|$ where $x <_s y$ denotes that the adjacency list of x arrives before that of y in the stream.

Motivated by our empirical results in Section F.7, it is reasonable in some settings to assume we have access to oracles that can predict a good approximation to R_{xy}. We refer to such oracles as *value oracles*.

In the first version of this oracle, we assume that the probability of approximation error decays linearly with the error from above but exponentially with the error from below.

Definition 1.2. *Given an edge e, an (α, β) value-prediction oracle outputs a random value $p(e)$ where $\mathbb{E}[p(e)] \leq \alpha R_e + \beta$, and $\mathbf{Pr}[p(e) < \frac{R_e}{\lambda} - \beta] \leq Ke^{-\lambda}$ for some constant K and any $\lambda \geq 1$.*

For this variant, we prove the following theorem.

Theorem 1.5. *Given an oracle with parameters (α, β), there exists a one-pass algorithm, Algorithm 2, with space complexity $O(\epsilon^{-2}\log^2(K/\epsilon)(\alpha + m\beta/T))$ in the adjacency list model that returns a $(1 \pm \epsilon)$-approximation to the number of triangles T with probability at least $7/10$.*

In the second version of this noisy oracle, we assume that the probability of approximation error decays linearly with the error from both above and below. For this variant, we prove that we can achieve the same guarantees as Theorem 1.5 up to logarithmic factors (see Theorem B.1). The algorithms and proofs for both Theorem 1.5 and Theorem B.1 appear in Appendix B.2.

Experiments We conduct experiments to verify our results for triangle counting on a variety of real world networks (see Table 2) in both the arbitrary and adjacency list models. Our algorithms use additional information through predictors to improve empirical performance. The predictors are data dependent and include: memorizing heavy edges in a small portion of the first graph in a sequence of graphs, linear regression, and graph neural networks (GNNs). Our experimental results show that we can achieve up to **5x** decrease in estimation error while keeping the same amount of edges as other state of the art empirical algorithms. For more details, see Section 4. In Section F.7, we show that our noisy oracle models are realistic for real datasets.

Related Empirical Works On the empirical side, most of the focus has been on triangle counting in the arbitrary order model for which there are several algorithms that work well in practice. We primarily focus on two state-of-the-art baselines, ThinkD (Shin et al., 2018) and WRS (Shin, 2017). In these works, the authors compare to previous empirical benchmarks such as the ones given in Stefani et al. (2017); Han & Sethu (2017); Lim & Kang (2015) and demonstrate that their algorithms achieve superior estimates over these benchmarks. There are also other empirical works such as Ahmed et al. (2017) and Ahmed & Duffield (2020) studying this model but they do not compare to either ThinkD or WRS. While these empirical papers demonstrate that their algorithm returns unbiased estimates, their theoretical guarantees on space is incomparable to the previously stated space bounds for theoretical algorithms in Table 1. Nevertheless, we use ThinkD and WRS as part of our benchmarks due to their strong practical performance and code accessibility.

Implicit Predictors in Prior Works The idea of using a predictor is implicit in many prior works. The optimal two pass triangle counting algorithm of McGregor et al. (2016) can be viewed as an implementation of a heavy edge oracle after the first pass. This oracle is even stronger than the K-noisy oracle as it is equivalent to an oracle that is always correct on an edge e if N_e either exceeds or is under the threshold ρ by a constant multiplicative factor. This further supports our choice of oracles in our theoretical results, as a stronger version of our oracle can be implemented using one additional pass through the data stream (see Section G). Similarly, the optimal triangle counting streaming algorithm (assuming a *random* order) given in McGregor & Vorotnikova (2020) also implicitly defines a heavy edge oracle using a small initial portion of the random stream (see Lemma 2.2 in McGregor & Vorotnikova (2020)). The random order assumption allows for the creation of such an oracle since heavy edges are likely to have many of their incident triangle edges appearing in an initial portion of the stream. We view these two prior works as theoretical justification for our oracle definitions. Lastly, the WRS algorithm also shares the feature of defining an implicit oracle: some space is reserved for keeping the most recent edges while the rest is used to keep a random sample of edges. This can be viewed as a specific variant of our model, where the oracle predicts recent edges as heavy.

Preliminaries. $G = (V, E)$ denotes the input graph, and n, m and T denote the number of vertices, edges and triangles (or four-cycles) in G, respectively. We use $N(v)$ to denote the set of neighbors of a node v, and Δ to denote the set of triangles. In triangle counting, for each $xy \in E(G)$, we recall that $N_{xy} = |z : \{x, y, z\} \in \Delta|$ is the number of triangles incident to edge xy, and $R_{xy} = |z : \{x, y, z\} \in \Delta, x <_s z <_s y|$ is the number of triangles adjacent to xy with the third vertex z of the triangle between x and y in the adjacency list order. Table A summarizes the notation.

2 Triangle Counting in the Adjacency List Model

We describe an algorithm with a heavy edge oracle, and one with a value oracle.

Heavy Edge Oracle. We present an overview of our one-pass algorithm, Algorithm 1, with a space complexity of $\widetilde{O}(\min(\epsilon^{-2}m^{2/3}/T^{1/3}, \epsilon^{-1}m^{1/2}))$, given in Theorem 1.1. We defer the pseudocode

and proof of the theorem to Appendix B. The adaptations and proofs for Theorems 1.4, 1.5 and B.1 for the various noisy oracles appear in Appendix B.1 and Appendix B.2.

Our algorithm works differently depending on the value of T. We first consider the case that $T \geq (m/\epsilon)^{1/2}$. Assume that for each sampled edge xy in the stream, we can exactly know the number of triangles R_{xy} this edge contributes to T. Then the rate at which we would need to sample each edge would be proportional to $p_{naive} \approx \epsilon^{-2}\Delta_E/T$, where Δ_E is the maximum number of triangles incident to any edge. Hence, our first idea is to separately consider light and non-light edges using the heavy edge oracle. This allows us to sample edges that are deemed light by the oracle at a lower rate, p_1, and compute their contribution by keeping track of R_{xy} for each such sampled edge. Intuitively, light edges offer us more flexibility and thus we can sample them with a lower probability while ensuring the estimator's error does not drastically increase. In order to estimate the contribution due to non-light edges, we again partition them into two types: medium and heavy, according to some threshold ρ. We then use an observation from McGregor et al. (2016), that since for heavy edges $R_{xy} > \rho$, it is sufficient to sample from the entire stream at rate $p_3 \approx \epsilon^{-2}/\rho$, in order to both detect if some edge xy is heavy and if so to estimate R_{xy}.

Therefore, it remains to estimate the contribution to T due to medium edges (these are the edges that are deemed non-light by the oracle, and also non-heavy according to the sub-sampling above). Since the number of triangles incident to medium edges is higher than that of light ones, we have to sample them at some higher rate $p_2 > p_1$. However, since their number is bounded, this is still space efficient. We get the desired bounds by correctly setting the thresholds between light, medium and heavy edges.

When $T < (m/\epsilon)^{1/2}$ our algorithm becomes much simpler. We only consider two types of edges, light and heavy, according to some threshold T/ρ. To estimate the contribution due to heavy edges we simply store them and keep track of their R_{xy} values. To estimate the contribution due to light edges we sub-sample them with rate $\epsilon^{-2} \cdot \Delta_E/T = \epsilon^{-2}/\rho$. The total space we use is $\widetilde{O}(\epsilon^{-1}\sqrt{m})$, which is optimal in this case. See Algorithm 1 for more details of the implementation.

Value-Based Oracle. We also consider the setting where the predictor returns an estimate $p(e)$ of R_e, and we assume $R_e \leq p(e) \leq \alpha \cdot R_e$, where $\alpha \geq 1$ is an approximation factor. We relax this assumption to also handle additive error as well as noise, but for intuition we focus on this case. The value-based oracle setting requires the use of novel techniques, such as the use of exponential random variables (ERVs). Given this oracle, for an edge e, we compute $p(e)/u_e$, were u_e is a standard ERV. We then store the $O(\alpha \log(1/\epsilon))$ edges e for which $p(e)/u_e$ is largest. Since we are in the adjacency list model, once we start tracking edge $e = xy$, we can also compute the true value R_e of triangles that the edge e participates in (since for each future vertex z we see, we can check if x and y are both neighbors of z). Note that we track this quantity only for the $O(\alpha \log(1/\epsilon))$ edges that we store. Using the max-stability property of ERVs, $\max_e R_e/u_e$ is equal in distribution to T/u, where u is another ERV. Importantly, using the density function of an ERV, one can show that the edge e for which R_e/u_e is largest is, with probability $1 - O(\epsilon^3)$, in our list of the $O(\alpha \log(1/\epsilon))$ largest $p(e)/u_e$ values that we store. Repeating this scheme $r = O(1/\epsilon^2)$ times, we obtain independent estimates $T/u^1, \ldots, T/u^r$, where $u^1, \ldots, u^r$ are independent ERVs. Taking the median of these then gives a $(1 \pm \epsilon)$-approximation to the total number T of triangles. We note that ERVs are often used in data stream applications (see, e.g., Andoni (2017)), though to the best of our knowledge they have not previously been used in the context of triangle estimation. We also give an alternative algorithm, based on subsampling at $O(\log n)$ scales, which has worse logarithmic factors in theory but performs well empirically.

3 Triangle Counting in the Arbitrary Order Model

In this section we discuss Algorithm 4 for estimating the number of triangles in an arbitrary order stream of edges. The pseudo-code of the algorithm as well as omitted proofs for the different oracles are given in Supplementary Section C. Here we give the intuition behind the algorithm. Our approach relies on sampling the edges of the stream as they arrive and checking if every new edge forms a triangle with the previously sampled edges. However, as previously discussed, this approach alone fails if some edges have a large number of triangles incident to them as "overlooking" such edges might lead to an underestimation of the number of triangles. Therefore, we utilize a heavy edge oracle, and refer to edges that are not heavy as *light*. Whenever a new edge arrives, we first

query the oracle to determine if the edge is heavy. If the edge is heavy we keep it, and otherwise we sample it with some probability. As in the adjacency list arrival model case, this strategy allows us to reduce the variance of our estimator. By balancing the sampling rate and our threshold for heaviness, we ensure that the space requirement is not too high, while simultaneously guaranteeing that our estimate is accurate.

In more detail, our algorithm works as follows. First, we set a heaviness threshold ρ, so that if we predict an edge e to be part of ρ or more triangles, we label it as heavy. We also set a sampling parameter p. We let H be the set of edges predicted to be heavy and let S_L be a random sample of edges predicted to be light. Then, we count three types of triangles. The first counter, ℓ_1, counts the triangles $\Delta = (e_1, e_2, e)$, where the first two edges seen in this triangle by the algorithm, represented by e_1 and e_2, are both in S_L. Note that we only count the triangle if e_1 and e_2,were both in S_L at the time e arrives in the stream. Similarly, ℓ_2 counts triangles whose first two edges are in S_L and H (in either order), and ℓ_3 counts triangles whose first two edges are in H. Finally, we return the estimate $\ell = \ell_1/p^2 + \ell_2/p + \ell_3$. Note that if the first two edges in any triangle are light, they will both be in S_L with probability p^2, and if exactly one of the first two edges is light, it will be in S_L with probability p. Therefore, we divide ℓ_1 by p^2 and ℓ_2 by p so that ℓ is an unbiased estimator.

4 Experiments

We now evaluate our algorithm on real and synthetic data whose properties are summarized in Table 2 (see Appendix F for more details).

Table 2: Datasets used in our experiments. Snapshot graphs are a sequence of graphs over time (the length of the sequence is given in parentheses) and temporal graphs are formed by edges appearing over time. The listed values for n (number of vertices), m (number of edges), and T (number of triangles) for Oregon and CAIDA are approximated across all graphs. The Oregon and CAIDA datasets come from Leskovec & Krevl (2014); Leskovec et al. (2005), the Wikibooks dataset comes from Rossi & Ahmed (2015), the Reddit dataset comes from Leskovec & Krevl (2014); Kumar et al. (2018), the Twitch dataset comes from Rozemberczki et al. (2019), the Wikipedia dataset comes from Rossi & Ahmed (2015), and the Powerlaw graphs are sampled from the Chung-Lu-Vu random graph model with expected degree of the i-th vertex proportional to $1/i^2$ (Chung et al., 2003).

Name	**Type**	**Predictor**	n	m	T
Oregon	Snapshot (9)	1st graph	$\sim 10^4$	$\sim 2.2 \cdot 10^4$	$\sim 1.8 \cdot 10^4$
CAIDA 2006	Snapshot (52)	1st graph	$\sim 2.2 \cdot 10^4$	$\sim 4.5 \cdot 10^4$	$\sim 3.4 \cdot 10^4$
CAIDA 2007	Snapshot (46)	1st graph	$\sim 2.5 \cdot 10^4$	$\sim 5.1 \cdot 10^4$	$\sim 3.9 \cdot 10^4$
Wikibooks	Temporal	Prefix	$\sim 1.3 \cdot 10^5$	$\sim 3.9 \cdot 10^5$	$\sim 1.8 \cdot 10^5$
Reddit	Temporal	Regression	$\sim 3.6 \cdot 10^4$	$\sim 1.2 \cdot 10^5$	$\sim 4.1 \cdot 10^5$
Twitch	-	GNN	$\sim 6.5 \cdot 10^3$	$\sim 5.7 \cdot 10^4$	$\sim 5.4 \cdot 10^4$
Wikipedia	-	GNN	$\sim 4.8 \cdot 10^3$	$\sim 4.6 \cdot 10^4$	$\sim 9.5 \cdot 10^4$
Powerlaw	Synthetic	EV	$\sim 1.7 \cdot 10^5$	$\sim 10^6$	$\sim 3.9 \cdot 10^7$

We now describe the edge heaviness predictors that we use (see also Table 2). Our predictors adapt to the type of dataset and information available for each dataset. Some datasets we use contain only the graph structure (nodes and edges) without semantic features, thus not enabling us to train a classical machine learning predictor for edge heaviness. In those cases we use the true counts on a small prefix of the data (either 10% of the first graph in a sequence of graphs, or a prefix of edges in a temporal graph) as predicted counts for subsequent data. However, we are able to create more sophisticated predictors on three of our datasets, using feature vectors in linear regression or a Graph Neural Network. Precise details of the predictors follow.

• **Snapshot**: For Oregon / CAIDA graph datasets, which contain a sequence of graphs, we use exact counting on a *small* fraction of the first graph as the predictor for *all the subsequent graphs*. Specifically, we count the number of triangles per edge, N_e, on the first graph for each snapshot dataset. We then only store 10% of the top heaviest edges and use these values as estimates for edge heaviness in all later graphs. If a queried edge is not stored, its predicted N_e value is 0.

• **Prefix**: In the WikiBooks temporal graph, we use the exact N_e counts on the first half of the graph edges (when sorted by their timestamps) as the predicted values for the second half.

• **Linear Regression**: In the Reddit Hyperlinks temporal graph, we use a separate dataset (Kumar et al., 2019) that contains 300-dimensional feature embeddings of subreddits (graph nodes). Two embeddings are close in the feature space if their associated subreddits have similar sub-communities. To produce an edge $f(e)$ embedding for an edge $e = uv$ from the node embedding of its endpoints, $f(u)$ and $f(v)$, we use the 602-dimensional embedding $(f(u), f(v), \|(f(u) - f(v)\|_1, \|(f(u) - f(v)\|_2)$. We then train a linear regressor to predict N_e given the edge embedding $f(e)$. Training is done on a prefix of the first half of the edges.

• **Link Prediction (GNN)**: For each of the two networks, we start with a graph that has twice as many edges as listed in Table 2, ($\sim 1.1 \cdot 10^5$ edges for Twitch and $9.2 \cdot 10^4$ edges for Wikipedia). We then randomly remove 50% of the total edges to be the training data set, and use the remaining edges as the graph we test on. We use the method proposed in Zhang & Chen (2018) to train a link prediction oracle using a Graph Neural Network (GNN) that will be used to predict the heaviness of the testing edges. For each edge that arrives in the stream of the test edges, we use the predicted likelihood of forming an edge given by the the neural network to the other vertices as our estimate for N_{uv}, the number of triangles on edge uv. See Section F.2 for details of training methodology.

• **Expected Value (EV)**: In the Powerlaw graph, the predicted number of triangles incident to each N_e is its expected value, which can be computed analytically in the CLV random graph model.

Baselines. We compare our algorithms with the following baselines.

• **ThinkD and WRS** (Arbitrary Order): These are the state of the art empirical one-pass algorithms from Shin et al. (2018) and Shin (2017) respectively. The ThinkD paper presents two versions of the algorithm, called 'fast' and 'accurate'. We use the 'accurate' version since it provides better estimates. We use the authors' code for our experiments (Shin et al., 2020; Shin, 2020).

• **MVV** (Arbitrary Order and Adjacency List): We use the one pass streaming algorithms given in McGregor et al. (2016) for the arbitrary order model and the adjacency list model.

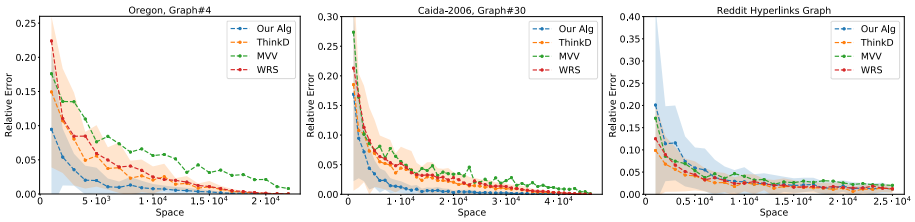
Error measurement. We measure accuracy using the relative error $|1 - \widetilde{T}/T|$, where T is the true triangle count and $\widetilde{T}$ is the estimate returned by an algorithm. Our plots show the space used by an algorithm (in terms of the number of edges) versus the relative error. We report median errors over 50 independent executions of each experiment, $\pm$ one standard deviation.

4.1 Results for Arbitrary Order Triangle Counting Experiments

In this section, we give experimental results for Algorithm 4 which approximates the triangle count in arbitrary order streams. Note that we need to set two parameters for Algorithm 4: p, which is the edge sampling probability, and ρ, which is the heaviness threshold. In our theoretical analysis, we assume knowledge of a lower bound on T in order to set p and ρ, as is standard in the theoretical streaming literature. However, in practice, such an estimate may not be available; in most cases, the only parameter we are given is a space bound for the number of edges that can be stored. To remedy this discrepancy, we modify our algorithm slightly by setting a fixed fraction of space to use for heavy edges (10% of space for all of our experiments) and setting p correspondingly to use up the rest of the space bound given as input. See details in Supplementary Section F.3.

Oregon and CAIDA In Figures 1(a) and 1(b), we display the relative error as a function of increasing space for graph $\#4$ in the dataset for Oregon and graph $\#30$ for CAIDA 2006. These figures show that our algorithm outperforms the other baselines by as much as a **factor of 5**. We do not display the error bars for MVV and WRS for the sake of visual clarity, but they are comparable to or larger than both ThinkD and our algorithm. A similar remark applies to all figures in Figure 1. As shown in Figure 2, these specific examples are reflective of the performance of our algorithm across the whole sequence of graphs for Oregon and CAIDA. We also show qualitatively similar results for CAIDA 2007 in Figure 4(a) in Supplementary Section F.4.

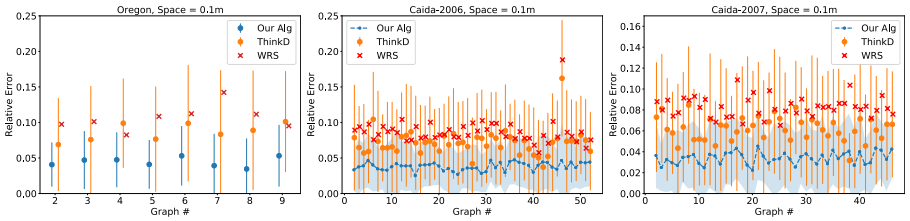
We also present accuracy results over the various graphs of the Oregon and CAIDA datasets. We fix the space to be 10% of the number of edges (which varies across graphs). Our results for the Oregon dataset are plotted in Figure 2(a) and the results for CAIDA 2006 and 2007 are plotted in



(a) Oregon (b) CAIDA 2006 (c) Reddit

(d) Wikibooks (e) Wikipedia (f) Powerlaw

Figure 1: Error as a function of space in the arbitrary order model.



(a) Oregon (b) CAIDA 2006 (c) CAIDA 2007

Figure 2: Error across snapshot graphs with space $0.1m$ in the arbitrary order model.

Figures 2(b) and 2(c), respectively. These figures illustrate that the quality of the predictor remains consistent over time even after a year has elapsed between when the first and the last graphs were created as our algorithm outperforms the baselines on average by up to a factor of 2.

Reddit Our results are displayed in Figure 1(c). All four algorithms are comparable as we vary space. While we do not improve over baselines in this case, this dataset serves to highlight the fact that predictors can be trained using node or edge semantic information (i.e., features).

Wikibooks and Powerlaw For Wikibooks, we see in Figure 1(d) that our algorithm is outperforming ThinkD and WRS by at least a factor of 2, and these algorithms heavily outperform the MVV algorithm. Nonetheless, it is important to note that our algorithm uses the exact counts on the first half of the edges (encoded in the predictor), which encode a lot information not available to the baselines. Thus the takeaway is that in temporal graphs, where edges arrive continuously over time (e.g., citation networks, road networks, etc.), using a prefix of the edges to form noisy predictors can lead to a significant advantage in handling future data on the same graph. Our results for Powerlaw are presented in Figure 1(f). Here too we see that as the space allocation increases, our algorithm outperforms ThinkD, MVV, and WRS.

Twitch and Wikipedia In Figure 1(e), we see that three algorithms, ours, MVV, and WRS, are all comparable as we vary space. The qualitatively similar result for Twitch is given in Figure 4(b). These datasets serve to highlight that predictors can be trained using modern ML techniques such as Graph Neural Networks. Nevertheless, these predictors help our algorithms improve over baselines for experiments in the adjacency list model (see Section below).

Results for Adjacency List Experiments Our experimental results for adjacency list experiments, which are qualitatively similar to the arbitrary order experiments, are given in full detail in Sections F.5 and F.6.

ACKNOWLEDGMENTS

Justin is supported by the NSF Graduate Research Fellowship under Grant No. 1745302 and MathWorks Engineering Fellowship. Sandeep and Shyam are supported by the NSF Graduate Research Fellowship under Grant No. 1745302. Ronitt was supported by NSF awards CCF-2006664, DMS 2022448, and CCF-1740751. Piotr was supported by the NSF TRIPODS program (awards CCF-1740751 and DMS-2022448), NSF award CCF-2006798 and Simons Investigator Award. Talya is supported in part by the NSF TRIPODS program, award CCF-1740751 and Ben Gurion University Postdoctoral Scholarship. Honghao Lin and David Woodruff would like to thank for partial support from the National Science Foundation (NSF) under Grant No. CCF-1815840.

REFERENCES

Nesreen Ahmed and Nick Duffield. Adaptive shrinkage estimation for streaming graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 10595–10606. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/780261c4b9a55cd803080619d0cc3e11-Paper.pdf.

Nesreen K. Ahmed, Nick Duffield, Theodore L. Willke, and Ryan A. Rossi. On sampling from massive graph streams. *Proc. VLDB Endow.*, 10(11):1430–1441, August 2017. ISSN 2150-8097. doi: 10.14778/3137628.3137651. URL https://doi.org/10.14778/3137628.3137651.

Mohammad Al Hasan and Vachik S Dave. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(2):e1226, 2018.

Alexandr Andoni. High frequency moments via max-stability. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pp. 6364–6368. IEEE, 2017.

Alexandr Andoni, Collin Burns, Ying Li, Sepideh Mahabadi, and David P. Woodruff. Streaming complexity of svms. In *APPROX-RANDOM*, 2020.

Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999. doi: 10.1017/CBO9780511624216.

Albert Atserias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. In *49th Annual IEEE Symposium on Foundations of Computer Science*, 2008.

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Conference on Learning Theory*, pp. 213–274. PMLR, 2017.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, 2018a.

Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 603–614. IEEE, 2018b.

Maria-Florina Balcan, Travis Dick, and Manuel Lang. Learning to link. *arXiv preprint arXiv:1907.00533*, 2019.

Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems*, 2020.

Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, 2017.

Suman K. Bera and C. Sheshadhri. How the degeneracy helps for traingle counting in graph streams. In *ACM Symposium on Principles of Database Systems*, June 2020.

Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *International Colloquium on Automata, Languages, and Programming*, pp. 244–254. Springer, 2013.

Fan Chung, Linyuan Lu, and Van Vu. The spectra of random graphs with given expected degrees. *Internet Math.*, 1(3):257–275, 2003. URL https://projecteuclid.org:443/euclid.im/1109190962.

Edith Cohen, Ofir Geri, and Rasmus Pagh. Composable sketches for functions of frequencies: Beyond the worst case. In *International Conference on Machine Learning*. PMLR, 2020.

Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6351–6361, 2017.

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2107.09770.

Yihe Dong, Piotr Indyk, Ilya P Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. *ICLR*, 2020.

Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In *International Conference on Machine Learning*, 2021.

Talya Eden, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, and Tal Wagner. Learning-based support estimation in sublinear time. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=tilovEHA3YS.

Illes J Farkas, Imre Derényi, A-L Barabási, and Tamas Vicsek. Spectra of" real-world" graphs: Beyond the semicircle law. In *The Structure and Dynamics of Networks*, pp. 372–383. Princeton University Press, 2011.

Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In *International Conference on Machine Learning*, pp. 3123–3132. PMLR, 2020.

Brooke Foucault Welles, Anne Van Devender, and Noshir Contractor. Is a "friend" a friend? investigating the structure of friendship networks in virtual worlds. In *CHI 2010 - The 28th Annual CHI Conference on Human Factors in Computing Systems, Conference Proceedings and Extended Abstracts*, pp. 4027–4032, June 2010. ISBN 9781605589312. doi: 10.1145/1753846.1754097. 28th Annual CHI Conference on Human Factors in Computing Systems, CHI 2010 ; Conference date: 10-04-2010 Through 15-04-2010.

Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning*, pp. 2319–2327. PMLR, 2019.

Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *SIGKDD Explor.*, 21:6–22, 2019.

Guyue Han and Harish Sethu. Edge sample and discard: A new algorithm for counting triangles in large dynamic graphs. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ASONAM '17, pp. 44–49, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349932. doi: 10.1145/3110025.3110061. URL https://doi.org/10.1145/3110025.3110061.

Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=r1lohoCqY7.

Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019b.

Zachary Izzo, Sandeep Silwal, and Samson Zhou. Dimensionality reduction for wasserstein barycenter. In *Advances in Neural Information Processing Systems*, 2021.

Rajesh Jayaram and John Kallaugher. An optimal algorithm for triangle counting in the stream. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P. Woodruff. Learning-augmented data stream algorithms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyxJ1xBYDH.

John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In Philip N. Klein (ed.), *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pp. 1778–1797. SIAM, 2017.

John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, pp. 119–133, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362276. doi: 10.1145/3294052.3319706. URL https://doi.org/10.1145/3294052.3319706.

Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Lecture Notes in Computer Science*, pp. 15–24, 2010. ISSN 1611-3349. doi: 10.1007/978-3-642-18009-5_3. URL http://dx.doi.org/10.1007/978-3-642-18009-5_3.

Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pp. 489–504. ACM, 2018.

Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 933–943. International World Wide Web Conferences Steering Committee, 2018.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1269–1278. ACM, 2019.

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1859–1877. SIAM, 2020.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pp. 177–187, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 159593135X. doi: 10.1145/1081870.1081893. URL https://doi.org/10.1145/1081870.1081893.

Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 462–470, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581934. doi: 10.1145/1401890.1401948. URL https://doi.org/10.1145/1401890.1401948.

Yongsub Lim and U Kang. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 685–694, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783285. URL https://doi.org/10.1145/2783258.2783285.

Mario Lucic, Matthew Faulkner, Andreas Krause, and Dan Feldman. Training gaussian mixture models at scale via coresets. *Journal of Machine Learning Research*, 18(160):1–25, 2018. URL http://jmlr.org/papers/v18/15-506.html.

Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pp. 3296–3305. PMLR, 2018.

M. Mahoney. Large text compression benchmark., 2011.

Andrew McGregor and Sofya Vorotnikova. Triangle and four cycle counting in the data stream model. PODS'20, pp. 445–456, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371087. doi: 10.1145/3375395.3387652. URL https://doi.org/10.1145/3375395.3387652.

Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, June 2016.

R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. ISSN 0036-8075. doi: 10.1126/science.298.5594.824. URL https://science.sciencemag.org/content/298/5594/824.

Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, 2018.

Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, 2020.

Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.

Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett.*, 112(7):277–281, 2012. doi: 10.1016/j.ipl.2011.12.007. URL https://doi.org/10.1016/j.ipl.2011.12.007.

Arnau Prat-Pérez, David Dominguez-Sal, Josep M Brunat, and Josep-Lluis Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 1677–1681, 2012.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pp. 9661–9670, 2018.

Jack Rae, Sergey Bartunov, and Timothy Lillicrap. Meta-learning neural bloom filters. In *International Conference on Machine Learning*, pp. 5271–5280, 2019.

Piyush Rai, Hal Daumé, and Suresh Venkatasubramanian. Streamed learning: One-pass svms. *ArXiv*, abs/0908.0572, 2009.

Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1834–1845. SIAM, 2020.

Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL http://networkrepository.com.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.

C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Fast triangle counting through wedge sampling. In *the International Conference on Data Mining (ICDM)*, 2013.

Kijung Shin. Wrs: Waiting room sampling for accurate triangle counting in real graph streams. *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 1087–1092, 2017.

Kijung Shin. Wrs: Waiting room sampling for accurate triangle counting in real graph streams. https://github.com/kijungs/waiting_room, 2020.

Kijung Shin, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Think before you discard: Accurate triangle counting in graph streams with deletions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 141–157. Springer, 2018.

Kijung Shin, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Think before you discard: Accurate triangle counting in graph streams with deletions. https://github.com/kijungs/thinkd, 2020.

Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. TriÈst: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Trans. Knowl. Discov. Data*, 11(4), June 2017. ISSN 1556-4681. doi: 10.1145/3059194. URL https://doi.org/10.1145/3059194.

Kapil Vaidya, Eric Knorr, Tim Kraska, and Michael Mitzenmacher. Partitioned learned bloom filter. In *International Conference on Learning Representations*, 2021.

Sofya Vorotnikova. Improved 3-pass algorithm for counting 4-cycles in arbitrary order streaming. *CoRR*, abs/2007.13466, 2020. URL https://arxiv.org/abs/2007.13466.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10:1–157, 2014.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 5171–5181, 2018.

A NOTATIONS TABLE

Notation	Definition
$G = (V, E)$	a graph G with vertex set V and edge set E
n	number of triangles
m	number of edges
T	number of triangles (or 4-cycles)
ϵ	approximation parameter
$<_s$	total ordering on the vertices according to their arrival in the adjacency list arrival model
Δ	set of triangles in G
$\square$	set of 4-cycles in G
$N(v)$	set of neighbors of node v
N_{xy}	number of triangles (4-cycles) incident to edge xy, i.e., $\lvert\{z \lvert (x,y,z) \in \Delta\}\rvert$ ($\lvert\{w,z \lvert (x,y,w.z) \in \square\}\rvert$)
R_{xy}	number of triangles (x,z,y) where x precedes z and z precedes y in the adjacency list arrival model, i.e., $\lvert\{z \lvert (x,z,y) \in \Delta, x <_s z <_s y\}\rvert$
Δ_E	maximum number of triangles incident to any edge
Δ_V	maximum number of triangles incident to any vertex
O_ρ	heavy edge oracle with threshold ρ

B OMITTED PROOFS FROM SECTION 2

In this section, we prove correctness and bound the space of Algorithm 1, thus proving Theorem 1.1. In the adjacency list model, recall that we have a total ordering on nodes $<_s$ based on the stream ordering, where $x <_s y$ if and only if the adjacency list of x arrives before the adjacency list of y in the stream. For each $xy \in E(G)$, we define $R_{xy} = |z : \{x,y,z\} \in \Delta$ and $x <_s z <_s y|$. Note that if $y <_s x$, then $R_{xy} = 0$. It holds that $\sum_{xy \in E} R_{xy} = T$.

We first give a detailed overview of the algorithm. We first consider the case when $T \geq (m/\epsilon)^{1/2}$. Let $\rho = (mT)^{1/3}$. We define the edge xy to be *heavy* if $R_{xy} \geq \rho$, *light* if $R_{xy} \leq \frac{T}{\rho}$, and *medium* if $\frac{T}{\rho} < R_{xy} < \rho$. We train the oracle to predict, upon seeing xy, whether R_{xy} is light or not. We define H, M, and L as the set of heavy, medium, and light edges, respectively. Define $T_L = \sum_{xy \in L} R_{xy}$. We define T_H and T_M similarly. Since $\sum_{xy \in E} R_{xy} = T = T_L + T_M + T_H$, it suffices to estimate each of these three terms.

For the light edges, as shown in Lemma B.2, if we sample edges with rate $p_1 \approx \epsilon^{-2}/\rho$, with high probability we obtain an estimate of T_L within error $\pm\epsilon T$. For the heavy edges, we recall an observation in McGregor et al. (2016): for an edge xy where R_{xy} is large, even if we do not sample xy directly, if we sample from the stream (at rate $p_3 \approx \epsilon^{-2}/\rho$), we are likely to sample some edges in the set $|xz : \{x,y,z\} \in \Delta$ and $x <_s z <_s y|$. We refer to the sampled set of these edges as S_{aux}.

Further, we will show that not only can we use S_{aux} to recognize whether R_{xy} is large, but also to estimate R_{xy} (Lemma B.3). What remains to handle is the medium edges. Since medium edges have higher values of R_{xy} than light edges, we sample with a larger rate, p_2, to reduce variance. However, we show there cannot be too many medium edges, so we do not pay too much extra storage space. The overall space we use is $\widetilde{O}(\epsilon^{-2}m^{2/3}/T^{1/3})$.

When $T < (m/\epsilon)^{1/2}$ our algorithm becomes much simpler. Let $\rho = \sqrt{m}/\epsilon$. We define the edge xy to be *heavy* if $R_{xy} \geq T/\rho$, and *light* otherwise (so in this case all the medium edges become heavy

edges). We directly store all the heavy edges and sub-sample with rate ϵ^{-2}/ρ to estimate T_L. The total space we use is $\widetilde{O}(\epsilon^{-1}\sqrt{m})$, which is optimal in this case. See Algorithm 1 for more details of the implementation.

Algorithm 1 Counting Triangles in the Adjacency List Model

Input: Adjacency list edge stream and an oracle O that outputs HEAVY if $R_{xy} \geq T/\rho$ and LIGHT otherwise.
Output: An estimate of the triangle count T.
Initialize $A_l, A_m, A_h = 0$, $S_L, S_M, S_{aux} = \emptyset$
if $T \geq (m/\epsilon)^{1/2}$ **then**
  Set $\rho = (mT)^{1/3}$, $p_1 = \alpha\epsilon^{-2}/\rho$, $p_2 = \min(\beta\epsilon^{-2}\rho/T, 1)$, $p_3 = \gamma\epsilon^{-2}\log n/\rho$.
else
  Set $\rho = m^{1/2}/\epsilon$, $p_1 = 0$, $p_2 = 1$, $p_3 = \gamma\epsilon^{-2}/\rho$. {$\alpha$, β, and γ are large enough constants.}
while seeing edges adjacent to v **do**
  for all $ab \in S_L \cup S_M$ **do**
    if $a, b \in N(v)$ **then** $C_{ab} = C_{ab} + 1$. {Update the counter for all the sampled light and medium edges}
  for all $av \in S_{aux}$ **do**
    $B_{av} = 1$ {seen av twice}.
  for each incident edge vu **do**
    W.p. p_3: $S_{aux} = \{vu\} \cup S_{aux}$, $B_{vu} = 0$
    if $O(uv)$ outputs LIGHT **then**
      if $uv \in S_L$ **then** $A_l = A_l + C_{uv}$. {Finished counting R_{uv}}
      else w.p. p_1, $S_L = \{vu\} \cup S_L$
    else {uv is either MEDIUM or HEAVY}
      $\# := |\{z : uz \in S_{aux}, B_{uz} = 1, z \in N(v)\}|$.
      if $\# \geq p_3\rho$ **then** {uv is HEAVY}
        $A_h = A_h + \#$.
      else if $uv \in S_M$ **then**
        $A_m = A_m + C_{uv}$. {Finished counting R_{uv}}
      else
        With probability p_2, $S_M = \{vu\} \cup S_M$.
return: $A_l/p_1 + A_m/p_2 + A_h/p_3$.

Remark B.1. *In the adjacency list model, we assume the oracle can predict whether $R_{xy} \geq \frac{T}{c}$ or not, where R_{xy} is dependent on the node order. This may sometimes be impractical. However, observe that $N_{xy} \geq R_{xy}$ and $\sum N_{xy} = 3T$. Hence, our proof still holds if we assume what the oracle can predict is whether $N_{xy} \geq \frac{T}{c}$, which could be a more practical assumption.*

Recall the following notation from Section 2. We define the edge xy to be *heavy* if $R_{xy} \geq \rho$, *light* if $R_{xy} \leq \frac{T}{\rho}$, and *medium* if $\frac{T}{\rho} < R_{xy} < \rho$. We define H, M, and L as the set of heavy, medium, and light edges, respectively. Define $T_L = \sum_{xy \in L} R_{xy}$. We define T_H and T_M similarly. Also, recall that since $\sum_{xy \in E} R_{xy} = T = T_L + T_M + T_H$, it suffices to estimate of these three terms.

Lemma B.2. *Let A be an edge set such that for each edge $xy \in A$, $R_{xy} \leq \frac{T}{c}$, for some parameter c, and let S be a subset of A such that every element in A is included in S with probability $p = \frac{1}{\tau}$ for $\tau = \epsilon^2 \cdot c/\alpha$, so that $1/\tau = \alpha\epsilon^{-2}\frac{1}{c}$ independently, for a sufficient large constant α. Then, with probability at least $9/10$, we have*

$$\tau \sum_{xy \in S} R_{xy} = \sum_{xy \in A} R_{xy} \pm \epsilon T$$

Proof. We let $Y = \tau \sum_{i \in S} R_i = \tau \sum_{i \in A} f_i R_i$, where $f_i = 1$ when $i \in S$ and $f_i = 0$ otherwise. It follows that $\mathbb{E}[Y] = T_A$. Then,

$$\begin{aligned}
\mathbf{Var}[Y] &= \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 \\
&= \tau^2 \left(\mathbb{E}[\sum_{i \in A} f_i R_i^2 + \sum_{i \neq j} f_i f_j R_i R_j] \right) - T_L^2 \\
&\leq \tau^2 \left(\mathbb{E}[\sum_{i \in A} f_i R_i^2] + \sum_{i,j} \frac{1}{\tau^2} R_i R_j \right) - T_L^2 \\
&= \tau \sum_{i \in A} R_i^2 \ .
\end{aligned}$$

To bound this, we notice that $R_i \leq \frac{T}{c}$ and $\sum_i R_i = T_A \leq T$, so

$$\tau \sum_{i \in A} R_i^2 \leq \tau \cdot \max_{i \in A} R_i \cdot \sum_{i \in A} R_i \leq \tau \cdot \frac{T}{c} \cdot T = \frac{\tau}{c} T^2 = \frac{1}{\alpha} \epsilon^2 T^2 \ .$$

By Chebyshev's inequality, we have that

$$\Pr[|Y - T_A| > \epsilon T] \leq \frac{\frac{1}{\alpha} \epsilon^2 T^2}{\epsilon^2 T^2} \leq \frac{1}{\alpha} \ .$$

□

The following lemma shows that for the heavy edges, we can use the edges we have sampled to recognize and estimate them.

Lemma B.3. *Let xy be an edge in E. Assume we sample the edges with rate $p_3 = \beta \epsilon^{-2} \log n / \rho$ in the stream for a sufficiently large constant β, and consider the set*

$$\# = |xz \text{ has been sampled} : \{x, y, z\} \in \Delta \text{ and } x <_s z <_s y| \ .$$

Then we have the following: if $R_{xy} \geq \frac{\rho}{2}$, with probability at least $1 - \frac{1}{n^5}$, we have $p_3^{-1} \cdot \# = (1 \pm \epsilon) R_{xy}$. If $R_{xy} < \frac{\rho}{2}$, with probability at least $1 - \frac{1}{n^5}$, we have $p_3^{-1} \cdot \# < \rho$.

Proof. We first consider the case when $R_{xy} \geq \frac{\rho}{2}$.

For each edge $i \in |xz : \{x, y, z\} \in \Delta$ and $x <_s z <_s y|$, let X_i = 1 if i has been sampled, or $X_i = 0$ otherwise. We can see the X_i are i.i.d. Bernoulli random variables and $\mathbb{E}[X_i] = p_3$. By a Chernoff bound we have

$$\mathbf{Pr}\left[|\sum_i X_i - p_3 R_{xy}| \geq \epsilon p_3 \cdot R_{xy} \right] \leq 2 \exp\left(-\epsilon^2 p_3 \cdot R_{xy} / 3 \right) \leq \frac{1}{n^5} \ .$$

Therefore, we have

$$\mathbf{Pr}\left[|p_3^{-1} \cdot \# - R_{xy}| \geq \epsilon R_{xy} \right] = \mathbf{Pr}\left[|p_3^{-1} \cdot \sum_i X_i - R_{xy}| \geq \epsilon R_{xy} \right] \leq \frac{1}{n^5} \ .$$

Alternatively, when $R_{xy} < \frac{\rho}{2}$, we have

$$\mathbf{Pr}\left[|p_3^{-1} \cdot \# - R_{xy}| \geq \frac{1}{2} \rho \right] \leq \frac{1}{n^5} \ ,$$

which means $p_3^{-1} \cdot \# < \rho$ with probability at least $1 - \frac{1}{n^5}$. □

We are now ready to prove Theorem 1.1, restated here for the sake of convenience.

Theorem 1.1. *There exists a one-pass algorithm, Algorithm 1, with space complexity*[3] $\widetilde{O}(\min(\epsilon^{-2}m^{2/3}/T^{1/3}, \epsilon^{-1}m^{1/2}))$ *in the adjacency list model that, using a learning-based oracle, returns a* $(1\pm\epsilon)$*-approximation to the number* T *of triangles with probability at least*[4] $7/10$.

Proof. We first consider the case when $T \geq (m/\epsilon)^{1/2}$.

For each edge $xy \in H$, from Lemma B.3 we have that with probability at least $1 - \frac{1}{n^5}$, $\#/p_3 = (1\pm\epsilon)R_{xy}$ in Algorithm 1. If $xy \notin H$, then with probability $1 - \frac{1}{n^5}$, xy will not contribute to A_h. Taking a union bound, we have that with probability at least $1 - 1/n^3$,

$$A_h/p_3 = (1\pm\epsilon)\sum_{xy\in H} R_{xy} = (1\pm\epsilon)T_H \ .$$

We now turn to deal with the light and medium edges. For a light edge xy, it satisfies $R_{xy} \leq T/\rho$ and therefore we can invoke Lemma B.2 with the parameter c set to ρ. For a medium edge xy, it satisfies $R_{xy} \leq \rho = T/(T/\rho)$, and therefore we can invoke Lemma B.2 with the parameter c set to T/ρ. Hence, we have that with probability $4/5$,

$$A_m/p_2 = T_M \pm \epsilon T, A_l/p_1 = T_L \pm \epsilon T \ .$$

Putting everything together, we have that with probability $7/10$,

$$|A_h/p_3 + A_m/p_2 + A_l/p_1 - T| \leq 3\epsilon T.$$

Now we analyze the space complexity. We note that there are at most ρ medium edges. Hence, the expected total space we use is $O(mp_3 + \rho p_2 + mp_1) = O(\epsilon^{-2}m^{2/3}/T^{1/3}\log n)$[5].

When $T < (m/\epsilon)^{1/2}$, as described in Section 2, we only have two classes of edges: heavy and light. We will save all the heavy edges and use sub-sampling to estimate T_L. The analysis is almost the same. □

B.1 Noisy Oracles for the Adjacency List Model

Proof of Theorem 1.4. We first consider the case that $T < (m/\epsilon)^{1/2}$. Recall that at this time, our algorithms becomes that we save all the heavy edges xy such that $p(e) \geq \epsilon T/\sqrt{m}$ and sampling the light edges with rate $p_l = \epsilon^{-1}/\sqrt{m}$. We define $\mathcal{L}$ as the set of deemed light edges, $\mathcal{H}$ as the set of deemed heavy edges and $L = |\mathcal{L}|$, $H = |\mathcal{H}|$. Let $\rho = \epsilon T/\sqrt{m}$, then we have the condition that for every edge e,

$$1 - K\cdot\frac{\rho}{R_e} \leq Pr[O_\rho(e) = \text{HEAVY}] \leq K\cdot\frac{R_e}{\rho}.$$

We will show that, (i) $\mathbb{E}[H] \leq (K+1)\frac{\sqrt{m}}{\epsilon}$, (ii) $\mathbf{Var}[A_L/p_1] \leq (4K+2)\epsilon^2T^2$. Under the above conditions we can get that with probability at least $9/10$ we can get a $(1\pm O(\sqrt{K})\cdot\epsilon)$-approximation of T by Chebyshev's inequality.

For (i), we divide $\mathcal{H} = H_h \cup H_l$, where the edges in H_h are indeed heavyedges, and the edges in H_l are indeed lightedges. Then, it is easy to see that $|H_h| \leq T/\rho$. For every light edges, the probability that it is included in H_l is at most $K\cdot\frac{R_e}{\rho}$, hence we have

$$\mathbb{E}[|H_h|] \leq K\cdot\sum_{e\in\mathcal{H}}(\frac{R_e}{\rho}) \leq K\cdot\frac{T}{\rho} = K\frac{\sqrt{m}}{\epsilon}$$

which implies $\mathbb{E}[H] \leq (K+1)\frac{\sqrt{m}}{\epsilon}$.

For (ii), we also divide $\mathcal{L} = L_l \cup L_h$ similarly. Then we have $\mathbf{Var}[A_L] \leq 2(\mathbf{Var}[X] + \mathbf{Var}[Y])$, where $X = \sum_{e\in L_l, e \text{ has been sampled}} R_e$, $Y = \sum_{e\in L_h, e \text{ has been sampled}} R_e$. Similar to the proof in Lemma B.2, we have

$$\mathbf{Var}[X] \leq p_1\sum_{e\in\mathcal{L}} R_e^2 \leq p_1\frac{T}{\rho}\rho^2 = p_1T\rho.$$

[3] We use $\widetilde{O}(f)$ to denote $O(f\cdot\text{polylog}(f))$.

[4] The success probability can be $1-\delta$ by running $\log(1/\delta)$ copies of the algorithm and taking the median.

[5] Using Markov's inequality, we have that with probability at least $9/10$, the total space we use is less than 10 times the expected total space.

Now we bound $\mathbf{Var}[Y]$. For every heavy edge we have that $\mathbf{Pr}[e \in L_h] \leq K \cdot \frac{\rho}{R_e}$. Then, condition on the oracle O_ρ, we have

$$\mathbb{E}[\mathbf{Var}[Y|O_\rho]] \leq p_1 K \sum_{e \in \mathcal{H}} \frac{\rho}{R_e} R_e^2 \leq p_1 K \sum_{e \in \mathcal{H}} \rho R_e \leq p_1 K \rho T \ .$$

Also, we have

$$\mathbf{Var}[\mathbb{E}[Y|O_\rho]] < p_1 K \sum_{e \in \mathcal{H}} \frac{\rho}{R_e} R_e^2 \leq p_1 K \rho T \ .$$

From $\mathbf{Var}[Y] = \mathbb{E}[\mathbf{Var}[Y|O_\rho]] + \mathbf{Var}[\mathbb{E}[Y|O_\rho]]$ we know $\mathbf{Var}[Y] \leq 2p_1 K \rho T$, which means $\mathbf{Var}[A_l/p_1] \leq \frac{1}{p_1}(4K+2)\rho T = (4K+2)\epsilon^2 T^2$.

When $T \geq (m/\epsilon)^{1/2}$, recall that the oracle O_ρ only needs to predict the edges that are medium edges or light edges. So, we can use the same way to bound the expected space for the deemed medium edges and the variance of the deemed light edges. □

B.2 A Value Prediction Oracle for the Adjacency List Model

In this section, we consider two types of value-prediction oracles, and variants of Algorithm 1 that take advantage of these oracles. Recall that given an edge xy, $R_{xy} = |z : \{x, y, z\} \in \Delta \text{ and } x <_s z <_s y|$.

Definition B.4. *A value-prediction oracle with parameter (α, β) is an oracle that, for any edge e, outputs a value $p(e)$ for which $\mathbb{E}[p(e)] \leq \alpha R_e + \beta$, and $\mathbf{Pr}[p(e) < \frac{R_e}{\lambda} - \beta] \leq Ke^{-\lambda}$ for some constant K and any $\lambda \geq 1$.*

Our algorithm is based on the following stability property of exponential random variables (see e.g. Andoni (2017))

Lemma B.5. *Let $x_1, x_2, ..., x_n > 0$ be real numbers, and let $u_1, u_2, ...u_n$ be i.i.d. standard exponential random variables with parameter $\lambda = 1$. Then we have that the random variable $\max(\frac{x_1}{u_1}, ..., \frac{x_n}{u_n}) \sim \frac{x}{u}$, where $x = \sum_i x_i$ and $u \sim \text{Exp}(1)$.*

Lemma B.6. *Let $r = O(\ln(1/\delta)/\epsilon^2)$ and $\epsilon \leq 1/3$. If we take r independent samples $X_1, X_2, ..., X_r$ from $\text{Exp}(1)$ and let $X = \text{median}(X_1, X_2, ..., X_r)$, then with probability $1 - \delta$, $X \in [\ln 2(1-\epsilon), \ln 2(1+5\epsilon)]$.*

Proof. We first prove the following statement: with probability $1 - \delta$, $F(X) \in [1/2 - \epsilon, 1/2 + \epsilon]$, where F is the cumulative density function of a standard exponential random variable.

Consider the case when $F(X) \leq 1/2 - \epsilon$. We use the indicator variable Z_i where $Z_i = 1$ if $F(X_i) \leq 1/2 - \epsilon$ or $Z_i = 0$ otherwise. Notice that when $F(X) \leq 1/2 - \epsilon$, at least half of the Z_i are 1, so $Z = \sum_i Z_i \geq r/2$. On the other hand, we have $\mathbb{E}[Z] = r/2 - r\epsilon$, by a Chernoff bound. We thus have

$$\mathbf{Pr}[|Z - \mathbb{E}[Z]| \geq \epsilon r] \leq 2e^{-\epsilon^2 r/3} \leq \delta/2 \ .$$

Therefore, we have with probability at most $\delta/2$, $F(X) \leq 1/2 - \epsilon$. Similarly, we have that with probability at most $\delta/2$, $F(X) \geq 1/2 + \epsilon$. Taking a union bound, we have that with probability at least $1 - \delta$, $F(X) \in [1/2 - \epsilon, 1/2 + \epsilon]$.

Now, condition on $F(X) \in [1/2 - \epsilon, 1/2 + \epsilon]$. Note that $F^{-1}(x) = -\ln(1-x)$, so if we consider the two endpoints of $F(X)$, we have

$$F^{-1}(1/2 - \epsilon) = -\ln(1/2 + \epsilon) \geq (1 - \epsilon)\ln 2,$$

and

$$F^{-1}(1/2 + \epsilon) = -\ln(1/2 - \epsilon) \leq (1 + 5\epsilon)\ln 2.$$

which indicates that $X \in [\ln 2(1-\epsilon), \ln 2(1+5\epsilon)]$. □

The algorithm utilizing this oracle is shown in Algorithm 2. As described in Section 2, here we runs $O(\frac{1}{\epsilon^2})$ copies of the subroutine and takes a medium of them. For each subroutine, we aim to output the maximum value $\max_e R_e/u_e$. We will show that with high probability for each subroutine,

this maximum will be at least $T/\log(K/\epsilon)$. Hence, we only need to save the edge e, such that $(R_e+\beta)/u_e$ is larger than $T/\log(K/\epsilon)$. However, one issue here is we need the information of T. We can remove this assumption when $\beta = 0$ using the following way: we will show that with high probability, the total number of edges we save is less than $H = O(\frac{1}{\epsilon^2}\log^2(K/\epsilon)(\alpha + m\beta/T))$, so every time when a new edge comes, we can search the minimum value M such that the total number of edges e that in at least one subroutine $(p(e)+\beta)/u_e > M$ is less than H(we count an edge multiple times if it occurs in multiple subroutines), then use M as the threshold. We can see M will increase as the new edge arriving and it will be always less than H.

Algorithm 2 Counting Triangles in the Adjacency List Model with a Value Prediction Oracle

Input: Adjacency list edge stream and an value prediction oracle with parameter(α, β).
Output: An estimate of the number T of triangles.
Initialize $X \leftarrow \emptyset$ and $H = O(\frac{1}{\epsilon^2}\log^2(K/\epsilon)(\alpha + m\beta/T))$, and let c be some large constant.
for $i = 0$ **to** $c\epsilon^{-2}$ **do** {Initialize sets for $c\epsilon^{-2}$ copies of samples, where S_i stores edges, Q_i stores all the exponential random variables for each sampled edge, and A_i stores the current maximum of each sample}
    $S^i \leftarrow \emptyset, Q^i \leftarrow \emptyset$, and $A^i \leftarrow 0$.
while seeing edges adjacent to y in the stream **do**
    for $i = 0$ **to** $c\epsilon^{-2}$ **do** {Update the counters for each sample}
        For all $ab \in S^i$: if $a, b \in N(y)$ then $C^i_{ab} \leftarrow C^i_{ab} + 1$
    for each incident edge yx **do**
        for $i = 0$ **to** $c\epsilon^{-2}$ **do**
            if $xy \in S^i$ **then** {Finished counting C_{xy}, update the current maximum}
                $A^i \leftarrow \max(A^i, C^i_{xy}/u^i_{xy})$.
            else {Put yx into the sample sets temporarily}
                Let $\hat{R}_{yx}$ be the predicted value of R_{yx}.
                $\hat{R}_{yx} \leftarrow \hat{R}_{yx} + \beta$.
                Generate $u^i_{yx} \sim \text{Exp}(1)$. Set $Q^i \leftarrow Q^i \cup \{u^i_{yx}\}, S^i \leftarrow S^i \cup \{yx\}$.
    Set M to be the minimal integer such that $\sum_i s_i \le H$, where $s_i = |\{e \in E \mid \hat{R}_e/u^i_e \ge M\}|$.
    for $i = 0$ **to** $c\epsilon^{-2}$ **do** {Adjust the samples to be within the space limit}
        Let $Q^i \leftarrow Q^i \setminus \{u^i_{ab}\}, S^i \leftarrow S^i \setminus \{ab\}$ if $\hat{R}_{ab}/u^i_{ab} < M$ for all $ab \in E$.
for $i = 0$ **to** $c\epsilon^{-2}$ **do**
    $X \leftarrow X \cup \{A^i\}$
return: $\ln 2 \cdot \text{median}(X)$.

We are now ready to prove Theorem 1.5. We first recall the theorem.

Theorem 1.5. *Given an oracle with parameters (α, β), there exists a one-pass algorithm, Algorithm 2, with space complexity $O(\epsilon^{-2}\log^2(K/\epsilon)(\alpha + m\beta/T))$ in the adjacency list model that returns a $(1 \pm \epsilon)$-approximation to the number of triangles T with probability at least $7/10$.*

Proof. It is easy to see that for the i-th subroutine, the value $f(i) = \max_e R_e/u^i_e$ is the sample from the distribution T/u, where $u \sim \text{Exp}(1)$. And if the edge e_i, for which the true value $R_{e_i}/u^i_{e_i}$ is the maximum value among E, is included in S^i, then the output of the i-th subroutine will be a sample from T/u. Intuitively, the prediction value $p(e)$ will help us find this edge.

We will first show that with probability at least $9/10$ the following events will happen:

- (i) $f(i) \ge c_1 T/\log(1/\epsilon)$ for all i, where c_1 is a constant.
- (ii) Let s_i be the number of the edge e in the i-th subroutine such that $(p(e)+\beta)/u^i_e \ge c_1^2 T/\log^2(K/\epsilon)$, then $\sum_i s_i \le c_2(\frac{1}{\epsilon^2}\log^2(K/\epsilon)(\alpha + m\beta/T))$ for some constant c_2.
- (iii) $p(e_i) + \beta \ge c_1 f(i)/\log(K/\epsilon)$ for all i.

For (i), recall that $f(i) \sim T/u$, where $u \sim \text{Exp}(1)$. Hence, we have

$$\mathbf{Pr}[f(i) < c_1 T/\log(1/\epsilon)] = e^{-\log(1/\epsilon)/c_1} \leq \frac{1}{100c}\frac{1}{\epsilon^2}\ .$$

Taking an union bound we get that with probability at least $99/100$, (i) will happen. For each edge e, we have

$$\mathbf{Pr}[p(e) + \beta < R_e/\lambda] < Ke^{-\lambda},$$

similarly we can get that with probability at least $99/100$, (iii) will happen.

For (ii), we first bound the expectation of s_i. For each edge e, we have

$$\begin{aligned}\mathbf{Pr}\left[(p(e)+\beta)/u_e^i \geq c_1^2 T/\log^2(K/\epsilon)\right] &= \mathbf{Pr}[u_e \leq c_1^2(p(e)+\beta)\log^2(K/\epsilon)/T] \\ &\leq c_1^2(p(e)+\beta)\log^2(K/\epsilon)/T \\ &\leq c_1^2(\alpha R_e + 2\beta)\log^2(K/\epsilon)/T\ .\end{aligned}$$

Hence, we have

$$\mathbb{E}[s_i] = \sum_e c_1^2 \frac{(\alpha R_e + 2\beta)\log^2(K/\epsilon)}{T} = O(\log^2(K/\epsilon)(\alpha + m\beta/T)).$$

Using Markov's inequality, we get that with probability at least $99/100$, (ii) will happen. Finally, taking a union bound, we can get with probability at least $9/10$, all the three events will happen.

Now, conditioned on the above three events, we will show that all the subroutine i will output $f(i)$. For the subroutine i, from (ii) we know that M will be always smaller than $O(T/\log^2(K/\epsilon))$, and from (i) and (iii) we get that $(p(e_i)+\beta)/u_{e_i}^i \geq \Omega(T/\log^2(K/\epsilon))$. Hence, the edge e_i will be saved in the subroutine i. From Lemma B.6 we get that with probability at least $7/10$, we can finally get a $(1\pm\epsilon)$-approximation of T. □

We note that when $\beta = 0$, the space complexity will become $O(\frac{1}{\epsilon^2}\log^2(K/\epsilon)\alpha)$, and in this case we will not need a lower bound of T.

We will also consider the following noise model.

Definition B.7. *A value-prediction oracle with parameter (α, β) is an oracle that, for any edge e, outputs a value $p(e)$ for which $\mathbf{Pr}[p(e) > \lambda\alpha R_e + \beta] \leq \frac{K}{\lambda}$, and $\mathbf{Pr}[p(e) < \frac{R_e}{\lambda} - \beta] \leq \frac{K}{\lambda}$ for some constant K and any $\lambda \geq 1$.*

The algorithm for this oracle is shown in Algorithm 3.

We now state our theorem.

Theorem B.1. *Given an oracle with parameter (α, β), there exists a one-pass algorithm, Algorithm 3, with space complexity $O(\frac{K}{\epsilon^2}\left(\alpha(\log n)^3 \log\log n + m\beta/T\right))$ in the adjacency list model that returns a $(1 \pm \sqrt{K}\cdot\epsilon)$-approximation to T with probability at least $7/10$.*

Proof. Define $q(e) = p(e) + \beta$, it is easy to see that from the condition we can get that

$$\mathbf{Pr}[q(e) > 2\lambda\alpha R_e] \leq K\frac{1}{\lambda} \text{ when } R_e > 2\beta, \mathbf{Pr}\left[q(e) < \frac{R_e}{\lambda}\right] \leq K\frac{1}{\lambda}$$

We define the set E_i such that $e \in E_i$ if and only if $q(e) \in I_i$. We can see $\mathbb{E}[A_i/p_i] = \sum_{e\in E_i} R_e$. Now, we bound $\mathbf{Var}[A_i/p_i]$ when $i \geq 1$.

Let $X = \sum_{e\in E_i, R_e \leq 2^{i+1}\beta} R_e$ and $Y = \sum_{e\in E_i, R_e > 2^{i+1}\beta} R_e$, then we have $\mathbf{Var}[A_i] \leq 2(\mathbf{Var}[X] + \mathbf{Var}[Y])$.

Similar to the proof in Lemma B.2, we can get that

$$\mathbf{Var}[X] \leq p_i \sum_{e\in E_i, R_e \leq 2^{i+1}\beta} R_e^2 \leq p_i(2^{i+1}\beta)^2 \frac{T}{2^{i+1}\beta} = p_i 2^{i+1}\beta T.$$

Algorithm 3 Counting Triangles in the Adjacency List Model with a Value Prediction Oracle

Input: Adjacency list edge stream and an value prediction oracle with parameter(α, β).
Output: An estimate of the number T of triangles.
Initialize $S_i^j \leftarrow \emptyset$ and $A_i^j \leftarrow 0$ where $i = O(\log n)$ and $j = O(\log\log n)$. $p_0 = c\epsilon^{-2}2^i\beta/T$ and $p_i = c\epsilon^{-2}2^i\beta(\log n)^2/T$ for some constant c. Define $I_0 = [0, 2\beta)$ and $I_i = [2^i\beta, 2^{i+1}\beta)$, $H \leftarrow 0$.
while seeing edges adjacent to y in the stream **do**
    For all $ab \in S_i^j$: if $a, b \in N(y)$ then $C_{ab} \leftarrow C_{ab} + 1$
    for each incident edge yx **do**
        if $xy \in S_i^j$ **then**
            $A_i^j \leftarrow A_i^j + C_{yx}$.
        else
            Let $\hat{R}_{yx}$ be the predicted value of R_{yx}.
            Search i such that $(\hat{R}_{yx} + \beta) \in I_i$
            For each j let $S_i^j \leftarrow S_i^j \cup \{yx\}$ with probability p_i.
for $i = 0$ **to** $O(\log n)$ **do**
    $H \leftarrow H + \text{median}(A_i^j)/p_i$
return: H .

For $\mathbf{Var}[Y]$, recall that for an edge e such that $R_e \geq 2^{i+1}\beta$, we have $\mathbf{Pr}[e \in E_i] \leq K\frac{q(e)}{R_e} \leq K\frac{2^{i+1}\beta}{R_e}$. Then, condition on the oracle O_θ, we have

$$\mathbb{E}[\mathbf{Var}[Y|O_\theta]] \leq p_i \sum_{R_e \geq 2^{i+1}\beta} K\frac{2^{i+1}\beta}{R_e}R_e^2 = p_i K 2^{i+1}\beta \sum_{R_e \geq 2^{i+1}\beta} R_e = p_i K 2^{i+1}\beta T.$$

And

$$\mathbf{Var}[\mathbb{E}[Y|O_\theta]] < p_i \sum_{R_e \geq 2^{i+1}\beta} K\frac{2^{i+1}\beta}{R_e}R_e^2 = p_i K 2^{i+1}\beta T.$$

From $\mathbf{Var}[Y] = \mathbb{E}[\mathbf{Var}[Y|O_\theta]] + \mathbf{Var}[\mathbb{E}[Y|O_\theta]]$ we get that $\mathbf{Var}[Y] \leq 2p_i K 2^{i+1}\beta T$, from which we can get that $\mathbf{Var}[A_i/p_i] = \frac{1}{p_i^2}\mathbf{Var}[A_i] = O(K(\epsilon/\log n)^2 T^2)$. Using Chebyshev's inequality and taking a median over $O(\log\log n)$ independent trials, we can get $X_i = \text{median}(A_i^j)$ satisfies $|X_i - \sum_{e\in E_i} R_e| \leq \sqrt{K}(\epsilon/\log n)T$ with probability $1 - O(1/\log n)$. A similar argument also holds for I_0, which means that after taking a union bound, with probability $9/10$, the output of the algorithm 3 is a $(1 \pm \sqrt{K}\cdot\epsilon)$- approximation of T.

Now we analyze the space complexity of Algorithm 3.

For $i \geq 1$, we have

$$\begin{aligned}\mathbb{E}\left[|E_i|\right] &= \mathbb{E}\left[\sum_{R_e \leq 2^{i-1}\beta/\alpha} [e \in E_i] + \sum_{R_e > 2^{i-1}\beta/\alpha} [e \in E_i]\right] \leq \mathbb{E}\left[\sum_{R_e \leq 2^{i-1}\beta/\alpha} [e \in E_i]\right] + \frac{T\alpha}{2^{i-1}\beta} \\ &\leq \sum_{R_e \leq 2^{i-1}\beta/\alpha} 2K\frac{R_e}{p(e)} + \frac{T\alpha}{2^{i-1}\beta} \leq \sum_{R_e \leq 2^{i-1}\beta/\alpha} K\frac{R_e}{2^{i-1}\beta} + \frac{T\alpha}{2^{i-1}\beta} \leq (K+1)\frac{T\alpha}{2^{i-1}\beta},\end{aligned}$$

and $|E_0| \leq m$, hence we have that the total expectation of the space is $\sum_i p_i\mathbb{E}[|E_i|] = O(\frac{K}{\epsilon^2}\left(\alpha(\log n)^3\log\log n + m\beta/T\right))$. □

C Omitted Proofs from Section 3

We first present Algorithm 4, and then continue to prove Theorem 1.2. In the following algorithm, for two sets A, B of edges, we let $w_{A,B}$ represent the set of pairs of edges (u, v), where $u \in A, v \in B$, and u, v share a common vertex.

Algorithm 4 Counting Triangles in the Arbitrary Order Model

Input: Arbitrary order edge stream and an oracle O_ρ that outputs HEAVY if $N_{xy} > \rho$ and LIGHT otherwise.
Output: An estimate of the triangle count T.
Initialize $\rho = \max\left\{\frac{\epsilon T}{\sqrt{m}}, 1\right\}$, $p = C\max\left\{\frac{1}{\epsilon\sqrt{T}}, \frac{\rho}{\epsilon^2 \cdot T}\right\}$ for a constant C.
Initialize $\ell_1, \ell_2, \ell_3 = 0$, and $S_L, H = \emptyset$.
for every arriving edge e in the stream **do**
  if the oracle O_ρ outputs HEAVY **then**
    Add e to H
  else
    Add e to S_L with probability p.
  for each $w \in w_{S_L,S_L}$ **do**
    if (e, w) is a triangle **then**
      Increment $\ell_1 = \ell_1 + 1$.
  for each $w \in w_{S_L,H}$ **do**
    if (e, w) is a triangle **then**
      Increment $\ell_2 = \ell_2 + 1$.
  for each $w \in w_{H,H}$ **do**
    if (e, w) is a triangle **then**
      Increment $\ell_3 = \ell_3 + 1$.
return: $\ell = \ell_1/p^2 + \ell_2/p + \ell_3$.

C.1 Proof of Theorem 1.2

Recall that we first assume that Algorithm 4 is given access to a perfect oracle O_ρ. That is, for every edge $e \in E$,

$$O_\rho(e) = \begin{cases} \text{LIGHT} & \text{if } T_e \le \rho \\ \text{HEAVY} & \text{if } T_e > \rho, \end{cases}$$

where T_e as the number of triangles incident to the edge e.

Theorem 1.2. First, we assume that $T \ge \epsilon^{-2}$. If not, our algorithm can just store all of the edges, since $m \le \epsilon^{-1} \cdot m/\sqrt{T}$.

For each integer $i \ge 1$, let $\mathcal{E}(i)$ denote the set of edges that are part of exactly i triangles, and let $E_i = |\mathcal{E}(i)|$. Since each triangle has 3 edges, we have that $\sum_{i\ge1} i \cdot E(i) = 3T$.

Let $\mathcal{T}_1$ denote the set of triangles whose first two edges in the stream are light (according to O_ρ). For every triangle t_i in $\mathcal{T}_1$, let χ_i denote the indicator of the event that the first two edges of t_i are sampled to S_L, and let $\ell_1 = \sum_i \chi_i$. Since each χ_i is 1 with probability p^2, $\mathbf{Ex}[\chi_i] = p^2$, and $\mathbf{Var}[\chi_i] = p^2 - p^4 \le p^2$. For any two variables χ_i, χ_j, they must be uncorrelated unless the triangles t_i, t_j share a light edge that is among the first two edges of t_i and among the first two edges of t_j. Moreover, in this case, $\mathbf{Cov}[\chi_i, \chi_j] \le \mathbf{Ex}[\chi_i\chi_j] = \mathbb{P}(\chi_i = \chi_j = 1)$. This event only happens if the first two edges of both t_i and t_j are sampled in S_L, but due to the shared edge, this comprises 3 edges in total, so $\mathbb{P}(\chi_i = \chi_j = 1) = p^3$. Hence, if we define t_i^{12} as the first two edges of t_i in the stream for each triangle t_i,

$$\mathbf{Ex}[\ell_1] = p^2|\mathcal{T}_1|,$$

and

$$\mathbf{Var}[\ell_1] \le \sum_{t_i\in\mathcal{T}_1} p^2 + \sum_{\substack{t_i,t_j\in\mathcal{T}_1 \\ t_i^{12}\cap t_j^{12}\neq\emptyset}} p^3 \;\le\; p^2|\mathcal{T}_1| + p^3 \sum_{T_e\le\rho} T_e^2$$

$$= p^2|\mathcal{T}_1| + p^3 \cdot \sum_{t=1}^{\rho} t^2 \cdot E(t) \;\le\; (p^2 + 3p^3 \cdot \rho) \cdot T \quad .$$

The first line follows by adding the covariance terms; the second line follows since each light edge e has at most T_e^2 pairs (t_i, t_j) intersecting at e; the third line follows by summing over $t = T_e$, ranging from 1 to ρ, instead over e; and the last line follows since $\sum_{t\ge1} t \cdot E(t) = 3T$.

Now, let $\mathcal{T}_2$ denote the set of triangles whose first two edges in the stream are light and heavy (in either order). For every triangle $t = (e_1, e_2, e_3)$ in $\mathcal{T}_2$, e_1, e_2 are sampled to $S_L \cup H$ with probability p. Also all other triangles have no chance to contribute to ℓ_2. Hence, $\mathbf{Ex}[\ell_2] = p \cdot \mathcal{T}_2$. Also, two triangles $t_i, t_j \in \mathcal{T}_2$ that intersect on an edge e have $\mathbf{Cov}(\chi_i, \chi_j) \neq 0$ only if e is light and $e \in t_i^{12}, t_j^{12}$. In this case.

$$\mathbf{Cov}(\chi_i, \chi_j) \leq \mathbf{Ex}[\chi_i \chi_j] \leq p,$$

since if e is added to S_L (which happens with probability p) then $\chi_i = \chi_j = 1$ as the other edges in t_i^{12}, t_j^{12} are heavy and are added to H with probability 1. Therefore,

$$\mathbf{Var}[\ell_2] = \sum_{t_i \in \mathcal{T}_2} p + \sum_{\substack{t_i, t_j \in \mathcal{T}_2 \\ t_i^{12} \cap t_j^{12} = e,\ O_\rho(e) = \text{LIGHT}}} p \leq (p + 3p \cdot \rho) \cdot T,$$

by the same calculations as was done to compute $\mathbf{Var}[\ell_1]$.

Finally, let $\mathcal{T}_3$ denote the set of triangles whose first two edges in the stream are heavy. Then $\ell_3 = |\mathcal{T}_3|$.

Now, using the well known fact that $\mathbf{Var}[X+Y] \leq 2(\mathbf{Var}[X]+\mathbf{Var}[Y])$ for any (possibly correlated) random variables X, Y, we have that

$$\begin{aligned}\mathbf{Var}[p^{-2}\ell_1 + p^{-1}\ell_2 + \ell_3] &\leq 2\left(p^{-4}(p^2 + 3p^3\rho)T + p^{-2}(p + 3p\rho)T\right) \\ &\leq 4T(p^{-2} + 3p^{-1}\rho),\end{aligned}$$

since ℓ_3 has no variance. Moreover, $\mathbf{Ex}[p^{-2}\ell_1 + p^{-1}\ell_2 + \ell_3] = |\mathcal{T}_1| + |\mathcal{T}_2| + |\mathcal{T}_3| = T$. So, by Chebyshev's inequality, since $\ell = \ell_1/p^2 + \ell_2/p + \ell_3$,

$$\Pr[|\ell - T| > \epsilon T] < \frac{\mathbf{Var}[\ell]}{\epsilon^2 \cdot T^2} < \frac{4(p^{-2} + 3p^{-1}\rho)}{\epsilon^2 \cdot T}.$$

Therefore, setting $p = C \cdot \max\left\{1/(\epsilon \cdot \sqrt{T}), \rho/(\epsilon^2 \cdot T)\right\}$ for some fixed constant C implies that, with probability at least $2/3$, ℓ is a $(1 \pm \epsilon)$-multiplicative estimate of T.

Furthermore, the expected space complexity is $O(mp + H) = O(mp + T/\rho)$. Setting $\rho = \max\{\epsilon T/\sqrt{m}, 1\}$ implies that the space complexity is $O(\epsilon^{-1}m/\sqrt{T} + \epsilon^{-2}m/T + \epsilon^{-1}\sqrt{m} \cdot T/T) = O(\epsilon^{-1}m/\sqrt{T} + \epsilon^{-1}\sqrt{m} \cdot T/T)$, since we are assuming that $T \geq \epsilon^{-2}$. Hence, assuming that T is a constant factor approximation of T, the space complexity is $O(\epsilon^{-1}m/\sqrt{T} + \epsilon^{-1}\sqrt{m})$. □

C.2 Proof of Theorem 1.2 for the K-noisy oracle

In this section, we prove Theorem 1.2 for the case that the given oracle is a K-noisy oracle, as defined in Definition 1.1. That is, we prove the following:

Theorem C.1. *Suppose that the oracle in Algorithm 4 is a K-noisy oracle as defined in Definition 1.1 for a fixed constant K. Then with probability $2/3$, Algorithm 4 returns $\ell \in (1 \pm \epsilon)T$, and uses space at most $O\left(\epsilon^{-1}\left(m/\sqrt{T} + \sqrt{m}\right)\right)$.*

Recall that in Definition 1.1, we defined a K-noisy oracle if the following holds. For every edge e, $1 - K \cdot \frac{\rho}{N_e} \leq \Pr[O_\rho(e) = \text{HEAVY}] \leq K \cdot \frac{N_e}{\rho}$. We first visualize this model. To visualize this error model, in Figure 3 we have plotted N_e versus the range $\left[1 - K \cdot \frac{\rho}{N_e}, K \cdot \frac{N_e}{\rho}\right]$ for $K = 2$. We set N_e to vary on a logarithmic scale for clarity. For example, if N_e exceeds the threshold ρ by a factor of 2, there is no restriction on the oracle output, whereas if N_e exceeds ρ by a factor of 4, then the oracle must classify the edge as heavy with probability at least 0.5. In contrast, the blue piece-wise line shows the probability $\Pr[O_\rho(e) = \text{HEAVY}]$ if the oracle O_ρ is a perfect oracle.

Proof of Theorem C.1. We follow the proof of Theorem 4.1 from the main paper. Let $\mathcal{T}_1$ be the set of triangles such that their first two edges in the stream are light according to the oracle. Let $\mathcal{T}_2$ be the set of triangles such that their first two edges in the stream are determined light and heavy according to the oracle. Finally, let $\mathcal{T}_3$ be the set of triangles for which their first two edges are

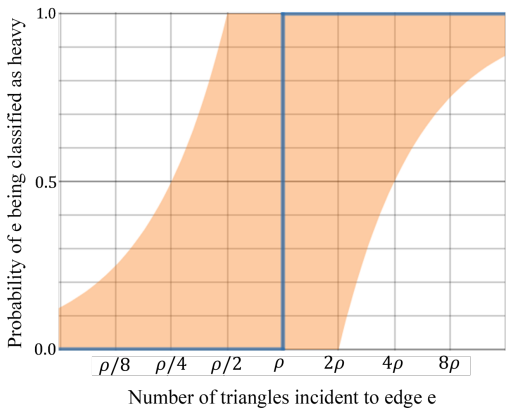


Figure 3: Plot of N_e, the number of triangles containing edge e, versus the allowed oracle probability range $\Pr[O_\rho(e) = \text{HEAVY}]$, shaded in orange. The blue piece-wise line shows the probability $\Pr[O_\rho(e) = \text{HEAVY}]$ if the oracle O_ρ is a perfect oracle.

determined heavy by the oracle. Furthermore, we define χ_i for each triangle t_i, $t(e)$ for each edge e, and $\mathcal{E}(i), E_i$ for each $i \geq 1$ as done in Theorem 4.1. Finally, we define $\mathcal{L}(i)$ as the set of deemed light edges that are part of exactly i triangles, and $L(i) = |\mathcal{L}(i)|$.

First, note that the expectation (over the randomness of O_ρ) of $L(i)$ is at most $E(i) \cdot K \cdot \frac{\rho}{i}$, since our assumption on $\Pr[O_\rho(e) = \text{heavy}]$ tells us that $\Pr[O_\rho(e) = \text{light}] \leq K \cdot \frac{\rho}{i}$ if $t(e) = i$. Therefore, by the same computation as in Theorem 4.1, we have that, conditioning on the oracle, $\mathbf{Ex}[\ell_1|O_\rho] = p^2|\mathcal{T}_1]$ and

$$\begin{aligned}
\mathbf{Var}[\ell_1|O_\rho] &= \sum_{t_i \in \mathcal{T}_1} p^2 + \sum_{\substack{t_i, t_j \in \mathcal{T}_1 \\ t_i^{12} \cap t_j^{12} \neq \emptyset}} p^3 \\
&\leq p^2|\mathcal{T}_1| + p^3 \sum_{e \text{ light}} t(e)^2 \\
&= p^2|\mathcal{T}_1| + p^3 \cdot \sum_{t \geq 1} t^2 \cdot L(t)
\end{aligned}$$

But since $\mathbf{Ex}_{O_\rho}[L(t)] \leq \frac{K \cdot \rho}{t} \cdot E(t)$ for all t, we have that

$$\begin{aligned}
\mathbf{Ex}_{O_\rho}[\mathbf{Var}[\ell_1|O_\rho]] &\leq \mathbf{Ex}_{O_\rho}\left[p^2 \cdot |\mathcal{T}_1| + p^3 \cdot \sum_{t \geq 1} t^2 L(t)\right] \\
&= p^2|\mathcal{T}_1| + K\rho \cdot p^3 \sum_{t \geq 1} t \cdot E(t) \\
&\leq (p^2 + 3Kp^3 \cdot \rho) \cdot T.
\end{aligned}$$

A similar analysis to the above shows that $\mathbf{Ex}[\ell_2] = p \cdot |\mathcal{T}_2|$ and that

$$\begin{aligned}
\mathbf{Ex}_{O_\rho}[\mathbf{Var}[\ell_2|O_\rho]] &\leq \mathbf{Ex}_{O_\rho}\left[p \cdot |\mathcal{T}_2| + p \cdot \sum_{t \geq 1} t^2 L(t)\right] \\
&= p|\mathcal{T}_2| + K\rho \cdot p \sum_{t \geq 1} t \cdot E(t) \\
&\leq (p + 3Kp \cdot \rho) \cdot T.
\end{aligned}$$

Hence, as in Theorem 4.1, we have $\mathbf{Ex}[\ell|O_\rho] = T$ and $\mathbf{Ex}_{O_\rho}[\mathbf{Var}[\ell|O_\rho]] \leq 4T \cdot (p^{-2} + 3Kp^{-1}\rho)$. Thus, $\mathbf{Ex}[\ell] = \mathbf{Ex}[\mathbf{Ex}[\ell|O_\rho]] = T$ and $\mathbf{Var}[\ell] = \mathbf{Ex}_{O_\rho}[\mathbf{Var}[\ell|O_\rho]] + \mathbf{Var}_{O_\rho}[\mathbf{Ex}[\ell|O_\rho]] \leq 4T \cdot (p^{-2} + 3Kp^{-1}\rho)$ by the laws of total expectation/variance. Therefore, since K is a constant, the variance is

the same as in Theorem 4.1, up to a constant. Therefore, we still have that $\ell \in (1 \pm O(\epsilon)) \cdot T$ with probability at least $2/3$.

Finally, the expected space complexity is bounded by

$$\begin{aligned} mp + \sum_{e \in m} Pr[O_\rho(e) = \textsf{heavy}] &\leq mp + \sum_{t \geq 1} E(t) \cdot K \cdot \frac{t}{\rho} \\ &= mp + \frac{K}{\rho} \cdot \sum_{t \geq 1} E(t) \cdot t \\ &= O(mp + T/\rho), \end{aligned}$$

since $\sum_{t \geq 1} E(t) \cdot t = 3T$ and $K = O(1)$. Hence, setting ρ and p as in the proof of Theorem 4.1 implies that the returned value is a $(1 \pm \epsilon)$-approximation of T, and the the space complexity is $O(\epsilon^{-1}(m/\sqrt{T} + \sqrt{m}))$ as before. □

C.3 Lower Bound

In this section, we prove a lower bound for algorithms in the arbitrary order model that have access to a perfect heavy edge oracle. Here heavy means $N_{uv} \geq T/c$ for a pre-determined threshold c, and we assume $c = o(m)$ and $T/c > 1$, as otherwise the threshold will be too small or too close to the average to give an accurate prediction. The following theorem shows an $\Omega(\min(m/\sqrt{T}, m^{3/2}/T))$ lower bound even with such an oracle.

Theorem C.2. *Suppose that the threshold of the oracle $c = O(m^q)$, where $0 \leq q < 1$. Then for any T and m, $T \leq m$, there exists $m' = \Theta(m)$ and $T' = \Theta(T)$ such that any one-pass arbitrary order streaming algorithm that distinguishes between m' edge graphs with 0 and T' triangles with probability at least $2/3$ requires $\Omega(m/\sqrt{T})$ space. For any T and m, $T = \Omega(m^{1+\delta})$ where $\max(0, q - \frac{1}{2}) \leq \delta < \frac{1}{2}$, there exists $m' = \Theta(m)$ and $T' = \Theta(T)$ such that any one-pass arbitrary order streaming algorithm that distinguishes between m' edge graphs with 0 and T' triangles with probability at least $2/3$ requires $\Omega(m^{3/2}/T)$ space.*

When $T \leq m$, we consider the hubs graph mentioned in Kallaugher & Price (2017).

Definition C.3 (Hubs Graph, Kallaugher & Price (2017))**.** *The hubs graph $H_{r,d}$ consists of a single vertex v with $2rd$ incident edges, and d edges connecting disjoint pairs of v's neighbors to form triangles.*

It is easy to see that in $H_{r,d}$, each edge has at most one triangle. Hence, there will not be any heavy edges of this kind in the graph. In Kallaugher & Price (2017), the authors show an $\Omega(r\sqrt{d})$ lower bound for the hubs graph.

Lemma C.4 (Kallaugher & Price (2017))**.** *Given r and d, there exist two distributions $\mathcal{G}_1, \mathcal{G}_2$ on subgraphs G_1 and G_2 of $H_{r,d}$, such that any algorithm which distinguishes them with probability at least $2/3$ requires $\Omega(r\sqrt{d})$ space, where G_i has $\Theta(rd)$ edges and $T(G_1) = d, T(G_2) = 0$.*

Now, given T and m, where $T \leq m$, we let $r = \Theta(m/T)$ and $d = T$. We can see $H_{r,d}$ has $\Theta(m)$ edges and T triangles, and we need $\Omega(r\sqrt{d}) = \Omega(m/\sqrt{T})$ space.

When $T > m$, we consider the following $\Omega(m^{3/2}/T)$ lower bound in Bera & Chakrabarti (2017).

Let H be a complete bipartite graph with b vertices on each side (we denote the two sides of vertices by A and B). We add an additional N vertex blocks $V_1, V_2, ...V_N$ with each $|V_i| = d$. Alice has an N-dimensional binary vector x. Bob has an N-dimensional binary vector y. Both x and y have exactly $N/3$ coordinates that are equal to 1. Then, we define the edge sets

$$E_{\text{Alice}} = \bigcup_{i : x_i = 1} \{\{u, v\}, u \in A, v \in V_i\}$$

and

$$E_{\text{Bob}} = \bigcup_{i : y_i = 1} \{\{u, v\}, u \in B, v \in V_i\} .$$

Let the final resulting graph be denoted by $G = (V, E)$ where $V = V_H \cup V_1 \cup ... \cup V_N$ and $E = E_H \cup E_{\text{Alice}} \cup E_{\text{Bob}}$.

In Bera & Chakrabarti (2017), the authors show by a reduction to the $\text{DISJ}_N^{N/3}$ problem in communication complexity, that we need $\Omega(N)$ space to distinguish the case when G has 0 triangles from the case when G has at least $b^2 d$ triangles.

Given m and T, $T = \Theta(m^{1+\delta})$ where $1 \leq \delta < 1/2$, as shown in Bera & Chakrabarti (2017), we can set $b = N^s$ and $d = N^{s-1}$ where $s = 1/(1-2\delta)$. This will make $m = \Theta(N^{2s})$ and $T = \Theta(N^{3s-1})$, which will make the $\Omega(m^{3/2}/T)$ lower bound hold. Note that at this time each edge will have an $O(\frac{1}{m})$-fraction of triangles or an $O(\frac{1}{m^{1-\frac{1}{2s}}})$-fraction of triangles. Assume the threshold of the oracle $c = O(m^q)$. When $\delta \geq \max(0, q - \frac{1}{2})$, there will be no heavy edges in the graph.

Theorem C.2 follows from the above discussion.

D Four-Cycle Counting in the Arbitrary Order Model

In the 4-cycle counting problem, for each $xy \in E(G)$, define $N_{xy} = |z, w : \{x, y, z, w\} \in \square|$ as the number of 4-cycles attached to edge xy.

In this section, we present the one-pass $\widetilde{O}(T^{1/3} + \epsilon^{-2} m/T^{1/3})$ space algorithm behind Theorem 1.3 and the proof of the theorem.[6]

We begin with the following basic idea.

- Initialize: $C \leftarrow 0, S \leftarrow \emptyset$. On the arrival of edge uv:
  - With probability p, $S \leftarrow \{uv\} \cup S$.
  - $C \leftarrow C + |\{\{w, z\} : uw, wz \text{ and } zv \in S\}|$.
- Return C/p^3.

Following the analysis in Vorotnikova (2020), we have

$$\textbf{Var}[C] \leq O(Tp^3 + T\Delta_E p^5 + T\Delta_W p^4) ,$$

where Δ_E and Δ_W denote the maximum number of 4-cycles sharing a single edge or a single wedge.

There are two important insights from the analysis: first, this simple sampling scheme can output a good estimate to the number of 4-cycles on graphs that do not have too many heavy edges and heavy wedges (the definitions of heavy will be shown later). Second, assuming that we can store all the heavy edges, then at the end of the stream we can also estimate the number of 4-cycles that have exactly one heavy edge but do not have heavy wedges.

For the 4-cycles that have heavy wedges, we use an idea proposed in McGregor & Vorotnikova (2020); Vorotnikova (2020): for any node pair u and v, if we know their number of common neighbors (denoted by k), we can compute the exact number of 4-cycles with u, v as a diagonal pair (i.e., the four cycle contains two wedges with the endpoints u, v), and this counts to $\binom{k}{2}$. Furthermore, if k is large (in this case we say all the wedges with endpoints u, v are heavy), we can detect it and obtain a good estimation to k by a similar vertex sampling method mentioned in Section 3. We can then estimate the total number of 4-cycles that have heavy wedges with our samples.

At this point, we have not yet estimated the number of 4-cycles having more than one heavy edge but without heavy wedges. However, McGregor & Vorotnikova (2020) shows this class of 4-cycles only takes up a small fraction of T, and hence we can get a $(1 \pm \epsilon)$-approximation to T even without counting this class of 4-cycles.

The reader is referred to Algorithm 5 for a detailed version of our one-pass, 4-cycle counting algorithm. Before the stream, we randomly select a node set S, where each vertex is in S with probability

[6] We assume $\frac{1}{T^{1/6}} \leq O(\epsilon)$, which is the same assumption as in previous work

Algorithm 5 Counting 4-cycles in the Arbitrary Order Model

Input: Arbitrary Order Stream and an oracle that outputs TRUE if $N_{xy} \geq T/\rho$ and FALSE otherwise.
Output: An estimate of the number T of 4-cycles.
Initialize $A_l \leftarrow 0$, $A_h \leftarrow 0$, $A_w \leftarrow 0$ $E_L \leftarrow \emptyset$ $E_H \leftarrow \emptyset$, $E_S \leftarrow \emptyset$, and $W \leftarrow \emptyset$. Set $\rho \leftarrow T^{1/3}$, $p \leftarrow \alpha\epsilon^{-2}\log n/\rho$ for a sufficiently large α, and Let S be a random subset of nodes such that each node is in S with probability p.
while seeing edge uv in the stream **do**
  if $u \in S$ or $v \in S$ **then**
    $E_S \leftarrow \{uv\} \cup E_S$.
  if the oracle outputs FALSE **then**
    With probability p, $E_L \leftarrow \{uv\} \cup E_L$.
    Find pair (w, z) such that $uw, wz,$ and $zv \in E_L$, let $D \leftarrow D \cup \{(u, w, z, v)\}$.
  else
    $E_H \leftarrow \{uv\} \cup E_H$.
for each node pair (u, v) **do**
  let $q(u, v)$ be the number of wedges with center in S and endpoints u and v.
  if $q(u, v) \geq pT^{1/3}$ **then**
    $A_w \leftarrow A_w + \binom{q(u,v)}{2}$.
    $W \leftarrow W \cup \{(u, v)\}$.
for each 4-cycle d in D **do**
  if the end points of wedges in d are not in W **then**
    $A_l \leftarrow A_l + 1$
for each edge uv in E_H **do**
  for each 4-cycle d formed with uv and $e \in E_L$ **do**
    if the end points of wedges in d are not in W **then**
      $A_h \leftarrow A_h + 1$
return: $A_l/p^3 + A_h/p^3 + A_w$.

p independently, and we later store all edges that are incident to S during the stream. In the meantime, we define the edge uv to be *heavy* if $N_{uv} \geq T^{2/3}$ and train the oracle to predict whether uv is heavy or not when we see the edge uv during the stream. Let $p = \widetilde{O}(\epsilon^{-2}/T^{1/3})$. We store all the heavy edges and sample each light edge with probability p during the stream. Upon seeing see a light edge uv, we look for the 4-cycles that are formed by uv and the light edges that have been sampled before, and then record them in set D. At the end of the stream, we first find all the node pairs that share many common neighbors in S and identify them as heavy wedges. Then, for each 4-cycle $d \in D$, we check if d has heavy wedges, and if so, remove it from D. Finally, for each heavy edge uv indicated by the oracle, we compute the number of 4-cycles that are formed by uv and the sampled light edges, and without heavy wedges. This completes all parts of our estimation procedure.

We now prove Theorem 1.3, restated here for the sake of convenience.

Theorem 1.3. *There exists a one-pass algorithm, Algorithm 5, with space complexity* $\widetilde{O}(T^{1/3} + \epsilon^{-2}m/T^{1/3})$ *in the arbitrary order model that, using a learning-based oracle, returns a* $(1 \pm \epsilon)$*-approximation to the number* T *of four cycles with probability at least* $7/10$.

Proof. From the Lemma 3 in Vorotnikova (2020), we know that with probability at least $9/10$, we can get an estimate A_w such that

$$A_w = T_w \pm \epsilon T.$$

Where T_w is the number of the 4-cycles that have at least one heavy wedge. We note that if a 4-cycle has two heavy wedges, it will be counted twice. However, Vorotnikova (2020) shows that this double counting is at most $O(T^{2/3}) = O(\epsilon)T$.

For the edge sampling algorithm mentioned in D, from the analysis in Vorotnikova (2020), we have

$$\mathbf{Var}[A_l/p^3] \leq O(T/p^3 + T\Delta_E/p + T\Delta_W/p^2) .$$

Notice that in our algorithm, the threshold of the heavy edges and heavy wedges are $N_{uv} \geq T^{2/3}$ and $N_w \geq T^{1/3}$, respectively, which means $\mathbf{Var}[\frac{A_l}{p^3}] = O(\epsilon^2 T^2)$. Hence, we can get an estimate A_l such that with probability at least $9/10$,

$$A_l = T_l \pm \epsilon T \ ,$$

where T_l is the number of 4-cycles that have no heavy edges. Similarly, we have with probability at least $9/10$,

$$A_h = T_h \pm \epsilon T \ ,$$

where T_h is the number of 4-cycles that have at most one heavy edge.

One issue is that the algorithm has not yet estimated the number of 4-cycles having more than one heavy edge, but without heavy wedges. However, from Lemma 5.1 in McGregor & Vorotnikova (2020) we get that the number of this kind of 4-cycles is at most $O(T^{5/6}) = O(\epsilon)T$. Hence putting everything together, we have

$$|A_l/p^3 + A_h/p^3 + A_w - T| \leq O(\epsilon)T \ ,$$

which is what we need.

Now we analyze the space complexity. The expected number of light edges we sample is $O(mp) = \widetilde{O}(\epsilon^{-2}m/T^{1/3})$ and the expected number of nodes in S and edges in E_S is $O(2mp) = \widetilde{O}(\epsilon^{-2}m/T^{1/3})$. The expected number of 4-cycles we store in D is $O(Tp^3) = \widetilde{O}(\epsilon^{-6})$. We store all the heavy edges, and the number of heavy edges is at most $O(T^{1/3})$. Therefore, the expected total space is $\widetilde{O}(T^{1/3} + \epsilon^{-2}m/T^{1/3})$. □

E Runtime Analysis

In this section, we verify the runtimes of our Algorithms 1, 2, 3, 4, and 5.

Proposition E.1. *Algorithm 1 runs in expected time at most* $\widetilde{O}\left(\min(\epsilon^{-2}m^{5/3}/T^{1/3}, \epsilon^{-1}m^{3/2})\right)$ *in the setting of Theorem 1.1, or* $\widetilde{O}\left(\min(\epsilon^{-2}m^{5/3}/T^{1/3}, K \cdot \epsilon^{-1}m^{3/2})\right)$ *in the setting of Theorem 1.4.*

Proof. For each edge $ab \in S_L \cup S_M$, we check whether we see both va and vb (lines 9-10) when looking at edges adjacent to v. So, this takes time $|S_L| + |S_M|$ per edge in the stream. We similarly check for each edge in S_{aux} (lines 11-12), so this takes time $|S_{aux}|$. Finally, for each edge vu (line 13), we note that lines 14-17 can trivially be implemented in $O(1)$ time, along with 1 call to the oracle. The remainder of the oracle simply involves looking through the set S_{aux} and S_M to search or count for elements. Thus, the overall runtime is $O(|S_L| + |S_M| + |S_{aux}|)$ per stream element, so the runtime is $O\left(m \cdot (|S_L| + |S_M| + |S_{aux}|)\right)$.

This is at most $O(m \cdot S)$, where S is the space used by the algorithm. So, the runtime is $\widetilde{O}\left(\min(\epsilon^{-2}m^{5/3}/T^{1/3}, \epsilon^{-1}m^{3/2}\right)$ in the setting of Theorem 1.1, and is at most $\widetilde{O}\left(\min(\epsilon^{-2}m^{5/3}/T^{1/3}, K \cdot \epsilon^{-1}m^{3/2})\right)$ in the setting of Theorem 1.4. □

Proposition E.2. *Algorithm 2 runs in time* $\widetilde{O}(\epsilon^{-2}(\alpha + m\beta/T)m)$.

Proof. For each edge $ab \in S^i$ for each $0 \leq i \leq c\epsilon^{-2}$, we check whether we see both ya and yb in the stream when looking at edges adjacent to y. So, lines 7-8 take time $O(\sum |S^i|)$ for each edge we see in the stream. By storing each S^i (and each Q^i) in a balanced binary search tree or a similar data structure, we can search for $xy \in S^i$ in time $O(\log n)$ in line 11, and it is clear that lines 12-16 take time $O(\log n)$ (since insertion into the data structure for Q^i, S^i can take time $O(\log n)$). Since we do this for each i from 0 to $c\epsilon^{-2}$ and for each incident edge yx we see, in total we spend $O(\epsilon^{-2}\log n + \sum |S^i|)$ time up to line 16. The remainder of the lines take time $O(\epsilon^{-2} \cdot m \log n)$, since the slowest operation is potentially deleting an edge from each S^i and a value from each Q^i up to m times (for each edge).

Overall, the runtime is $\widetilde{O}(\epsilon^{-2} \cdot m + m \cdot \sum |S^i|) = \widetilde{O}(m \cdot S)$, where S is the total space used by the algorithm. Hence, the runtime is $\widetilde{O}(\epsilon^{-2}(\alpha + m\beta/T)m)$. □

Proposition E.3. *Algorithm 3 runs in time* $\widetilde{O}(K\epsilon^{-2}(\alpha + m\beta/T)m)$.

Proof. For each edge $ab \in S_i^j$ for each $1 \leq i \leq O(\log n), 1 \leq j \leq O(\log\log n)$, we check whether we see both ya and yb in the stream when looking at edges adjacent to y. So, line 5 takes time $O(\sum |S_i^j|)$ for each edge we see in the stream. By storing each S_i^j in a balanced binary search tree or a similar data structure, we can search for $xy \in S^i$ in time $O(\log n)$ in line 7, and it is clear that lines 8-12 take time $O(\log n)$ (since insertion into the data structure for Q_i^j can take time $O(\log n)$). Since we do this for each i from 0 to $c\epsilon^{-2}$ and for each incident edge yx we see, in total we spend poly $\log n \cdot O(\sum |S^i|)$ time up to line 12. Finally, lines 13-15 take time poly $\log n$.

Overall, the runtime is $\widetilde{O}(m \cdot \sum |S^i|) = \widetilde{O}(m \cdot S)$, where S is the total space used by the algorithm. Hence, the runtime is $\widetilde{O}(K\epsilon^{-2}(\alpha + m\beta/T)m)$. □

Proposition E.4. *Algorithm 4 runs in time* $\widetilde{O}\left(\epsilon^{-1} \cdot m^2/\sqrt{T} + \epsilon^{-1} \cdot m^{3/2}\right)$.

Proof. For each edge e in the stream, first we check the oracle, and we either add e to H or to S_L with probability p (lines 6-9), which take $O(1)$ time. The remaining lines (lines 10-18) involve operations that take $O(1)$ time for each w which represents a pair (e_1, e_2) of edges where $e_1, e_2 \in S_L \cup H$ and (e, e_1, e_2) forms a triangle. So, the runtime is bounded by the amount of time it takes to find all $e_1, e_2 \in S_L \cup H$ such that (e, e_1, e_2) forms a triangle. If the edge $e = (u, v)$, we just find all edges in $S_L \cup H$ adjacent to u, and all edges in $S_L \cup H$ adjacent to v. Then, we sort these edges based on their other endpoint and match the edges if they form triangles. So, the runtime is $\widetilde{O}(|S_L| + |H|)$ per edge e in the stream. Finally, line 19 takes O(1) time.

Overall, the runtime is $\widetilde{O}(m \cdot (|S_L| + |H|)) = \widetilde{O}(m \cdot S)$ where S is the total space of the algorithm. So, the runtime is $\widetilde{O}\left(\epsilon^{-1} \cdot m^2/\sqrt{T} + \epsilon^{-1} \cdot m^{3/2}\right)$. □

Proposition E.5. *Algorithm 5 runs in time* $\widetilde{O}\left(\epsilon^{-2}/T^{1/3} \cdot (n^3 + m^2)\right)$.

Proof. We note that for each edge uv in the stream (line 4), the code in the loop (lines 5-11) can be implemented using 1 oracle call and $\widetilde{O}(|E_L|)$ time. The only nontrivial step here is to find pairs (w, z) such that uw, wz, zv are all in E_L. However, by storing E_L in a balanced binary search tree or similar data structure, one can enumerate through each edge $wz \in E_L$ and determine if uw and zv are in E_L in $O(\log |E_L|)$ time. So, lines 4-11 take time $\widetilde{O}(|E_L|)$ per stream element.

Next, lines 12-16 can easily be implemented in time $O(n^2 \cdot |S|)$, lines 17-19 can be easily implemented in time $\widetilde{O}(|D|)$ if the vertices in W are stored properly, and lines 20-23 can be done in $\widetilde{O}(|E_H| \cdot |E_L|)$ time. The last part is true since we check each $uv \in E_H$ and $e = (u', v') \in E_L$, and then check if u, u', v', v form a 4-cycle by determining if u, u' and v, v' are in $E_L \cup E_H$ (which takes time $O(\log |E_L| + \log |E_H|)$ time assuming E_L, E_H are stored in search tree or similar data structure).

Overall, we can bound the runtime as $\widetilde{O}(m \cdot |E_L| + n^2 \cdot |S| + |D| + |E_H| \cdot |E_L|) = \widetilde{O}(m \cdot |E_L| + n^2 \cdot |S|)$. As each edge is in E_L with probability at most p and each edge is in S with probability at most p, the total runtime is $\widetilde{O}\left((m^2 + n^3) \cdot p\right) = \widetilde{O}\left(\epsilon^{-2}/T^{1/3} \cdot (n^3 + m^2)\right)$. □

F Additional Experimental Results

All of our graph experiments were done on a CPU with i5 2.7 GHz dual core and 8 GB RAM or a CPU with i7 1.9 GHz 8 core and 16GB RAM. The link prediction training was done on a single GPU.

F.1 Dataset descriptions

- **Oregon:** 9 graphs sampled over 3 months representing a communication network of internet routers Leskovec & Krevl (2014); Leskovec et al. (2005).

- **CAIDA:** 98 graphs sampled approximately weekly over 2 years, representing a communication network of internet routers Leskovec & Krevl (2014); Leskovec et al. (2005).
- **Reddit Hyperlinks:** Network where nodes represent sub communities (subreddits) and edges represent posts that link two different sub communities Leskovec & Krevl (2014); Kumar et al. (2018).
- **WikiBooks:** Network representing Wikipedia users and pages, with editing relationships between them Rossi & Ahmed (2015).
- **Twitch:** A user-user social network of gamers who stream in a certain language. Vertices are the users themselves and the links are mutual friendships between them. Rozemberczki et al. (2019)
- **Wikipedia:** This is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. Mahoney (2011)
- **Synthetic Power law:** Power law graph sampled from the Chung-Lu-Vu (CLV) model with the expected degree of the ith vertex proportional to $1/i^2$ Chung et al. (2003). To create this graph, the vertices are 'revealed' in order. When the jth vertex arrives, the probability of forming an edge between the jth vertex and ith vertex for $i < j$ is proportional to $1/(ij)^2$.

F.2 Details on link prediction oracle

Our oracle for the Twitch and Wikipedia graphs is based on Link Prediction, where the task is to estimate the likelihood of the existence of edges or to find missing links in the network. Here we use the method and code proposed in Zhang & Chen (2018). For each target link, it will extract a local enclosing subgraph around a node pair, and use a Graph Neural Network to learn general graph structure features for link prediction. For the Twitch network, we use all the training links as the training data for the graph neural network, while for the Wikipedia network, we use 30% of the training links as the training data due to memory limitations. For the Twitch network, our set of links that we will try to predict are between two nodes that have a common neighbor in the training network, but do not form a link in the training network. This is about 3.8 million pairs, and we call this the candidate set. We do this for memory considerations. For the remaining node pairs, we set the probability they form a link to be 0. For the Wikipedia network, we randomly select a link candidate set of size 20 times the number of edges (about 1 million pairs) from the entire set of testing links. These link candidate sets will be used by the oracle to determine heaviness. Then, we use this network to do our prediction for all links in our candidate sets for the two networks.

Now we are ready to build our heavy edge oracle. For the adjacency list model, when we see the edge uv in the stream, we know all the neighbors of u, and hence, we only need to predict the neighbors of v. Let $\deg(v)$ be the degree of v in the training graph. The training graph and testing graph are randomly split. Hence, we can use the training set to provide a good estimation to the degree of v. Next, we choose the largest $\deg(v)$ edges incident to v from the candidate set, in terms of their predicted likelihood given by the the neural network, as our prediction of $N(v)$, which leads to an estimate of R_{uv}. For the arbitrary order model, we use the same technique as above to predict $N(u)$ and $N(v)$, and estimate N_{uv}.

F.3 Parameter Selection for Algorithm 2

We need to set two parameters for Algorithm 2: p, which is the edge sampling probability, and θ, which is the heaviness threshold. In our theoretical analysis, we assume knowledge of a lower bound on T in order to set p and θ, as is standard in the theoretical streaming literature. However, in practice, such an estimate may not be available; in most cases, the only parameter we are given is a space bound for the number of edges that can be stored. To remedy this discrepancy, we modify our algorithm in experiments as follows.

First, we assume we only have access to the stream, a space parameter Z indicating the maximum number of edges that we are allowed to store, and an estimate of m, the number of edges in the stream. Given Z, we need to designate a portion of it for storing heavy edges, and the rest for storing light edges. The trade-off is that the more heavy edges we store, the smaller our sampling probability of light edges would be. We manage this trade off in our implementation by reserving $0.3 \cdot Z$ of the edge 'slots' for heavy edges. The constant 0.3 is *fixed* throughout all our experiments.

We then set $p = 0.7 \cdot Z/m$. To improve the performance of our algorithm, we optimize for space usage by *always* insuring that we are storing exactly Z space (after observing the first Z edges of the stream). To do so and still maintain our theoretical guarantees, we perform the following procedure. Call the first Z edges of the stream the early phase and the rest of the edges the late phase.

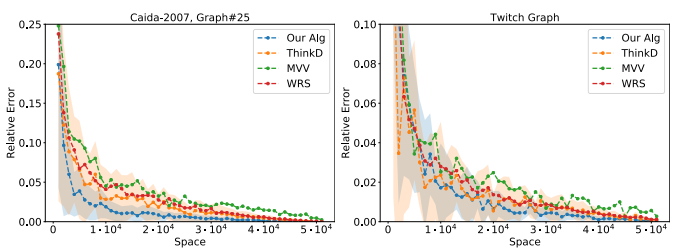
We always keep edges in the early phase to use our space allocation Z and also keep track of the 0.3-fraction of the heaviest edges. After the early phase is over, i.e., more than Z edges have passed in the stream, if a new incoming edge is heavier than the lightest of the stored heavy edges, we replace the least heavy stored edge with the new arriving edge and re-sample the replaced edge with probability p. Otherwise, if the new edge is not heavier than the lightest stored edge, we sample the new incoming edge with probability p. If we exceed Z, the space threshold, we replace one of the light edges sampled in the early phase. Then similarly as before, we re-sample this replaced edge with probability p and continue this procedure until one of the early light edges has been evicted. In the case that there are no longer any of the light edges sampled in the early phase stored, we replace a late light edge and then any arbitrary edge (again performing the re-sampling procedure for the evicted edge).

Note that in our modification, we only require our predictor be able to compare the heaviness between two edges, i.e., we do not need the predictor to output an estimate of the number of triangles on an edge. This potentially allows for more flexibility in the choice of predictors.

If the given space bound meets the space requirements of Theorem C.1, then the theoretical guarantees of this modification simply carry over from Theorem 1.2: we always keep the heaviest edges and always sample light edges with probability at least p. In case the space requirements are not met, the algorithm stores the most heavy edges as to reduce the overall variance.

F.4 Additional Figures from Arbitrary Order Triangle Counting Experiments

Additional figures for our arbitrary order triangle counting experiments are given in Figures 4 and 5.



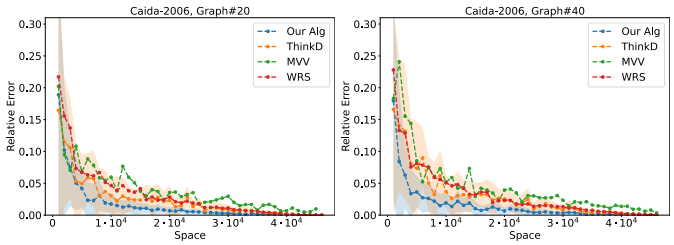
(a) CAIDA 2007 (b) Twitch

Figure 4: Error as a function of space for various graph datasets.

F.5 Experimental Design for Adjacency List Experiments

We now present our adjacencly list experiments. At a high level overview, similarly to the arbitrary order experiments, for our learning-based algorithm, we reserve the top 10% of the total space for storing the heavy edges. To do this in practice, we can maintain the heaviest edges currently seen so far and evict the smallest edge in the set when a new heavy edge is predicted by the oracle and we no longer have sufficient space. We also consider a multi-layer sub-sampling version of the algorithm in Section B.2. Here we use more information from the oracle by adapting the sub-sampling rates of edges based on their predicted value. For more details, see Section F.5. Our results are presented in Figure 6 (with additional plots given in Figure 7). Our algorithms soundly outperform the MVV baseline for most graph datasets. We only show the error bars for the multi-layer algorithm and MVV for clarity. Additional details follow.

We use the same predictor for N_{xy} in Section 4 as a prediction for R_{xy}. The experiment is done under a random vertex arrival order. For the learning-based algorithm, suppose Z is the maximum number of edges that we are allowed to store. we set the $k = Z/10$ edges with the highest predicted R_{uv} values to be the heavy edges, and store them during the stream (i.e., we use 10% of the total



(a) CAIDA 2006, Graph #20 (b) CAIDA 2006, Graph #40

(c) Oregon, Graph #3 (d) Oregon, Graph #7

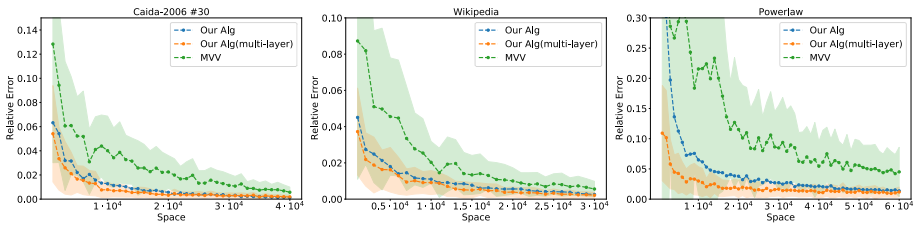
Figure 5: Error as a function of space for various graph datasets.

space for storing the heavy edges). For the remaining edges, we use a similar sampling-based method. Note that it is impossible to know the k-heaviest edges before we see all the edges in the stream. However, in the implementation we can maintain the k heaviest edges we currently have seen so far, and evict the smallest edge in the set when a new heavy edge is predicted by the oracle.

We also consider the multi-layer sub-sampling algorithm mentioned in Section B.2, which uses more information from the oracle. We notice that in many graph datasets, most of the edges having very few number of triangles attached to. Taking an example of the Oregon and CAIDA graph, only about 3%-5% of edges will satisfy $R_e \geq 5$ under a random vertex arrival order. Hence, for this edges, intuitively we can estimate them using a slightly smaller space.

For the implementation of this algorithm(we call it multi-layer version), we use 10% of the total space for storing the top $k = Z/10$ edges, and 70% of the space for sub-sampling the edges that the oracle predict value is very tiny(the threshold may be slightly different for different datasets, like for the Oregon and CAIDA graph, we set the threshold to be 5). Then, we use 20% of the space for sub-sampling the remaining edges, for which we call them the medium edges.

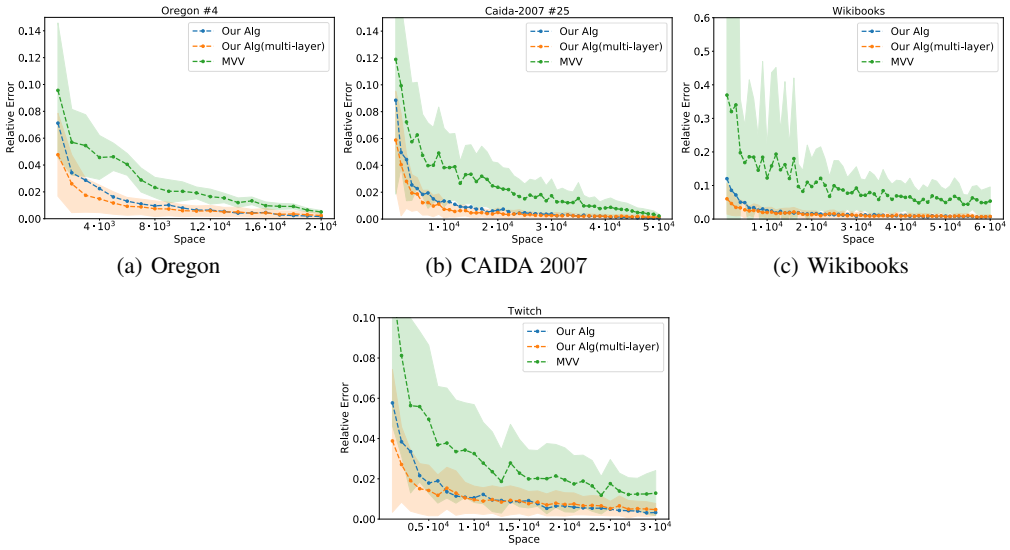
F.6 Figures from Adjacency List Triangle Counting Experiments



(a) CAIDA 2006 (b) Wikipedia (c) Powerlaw

Figure 6: Error as a function of space in the adjacency list model.

Additional figures from the adjacency list triangle counting experiments are shown in Figure 7. They are qualitatively similar to the results presented in Figure 6 as the multi-layer sampling algorithm is superior over the MVV baseline for all of our datasets.



(a) Oregon (b) CAIDA 2007 (c) Wikibooks

(d) Twitch

Figure 7: Error as a function of space for various graph datasets.

F.7 Accuracy of the Oracle

In this section, we evaluate the accuracy of the predictions the oracle gives in our experiments.

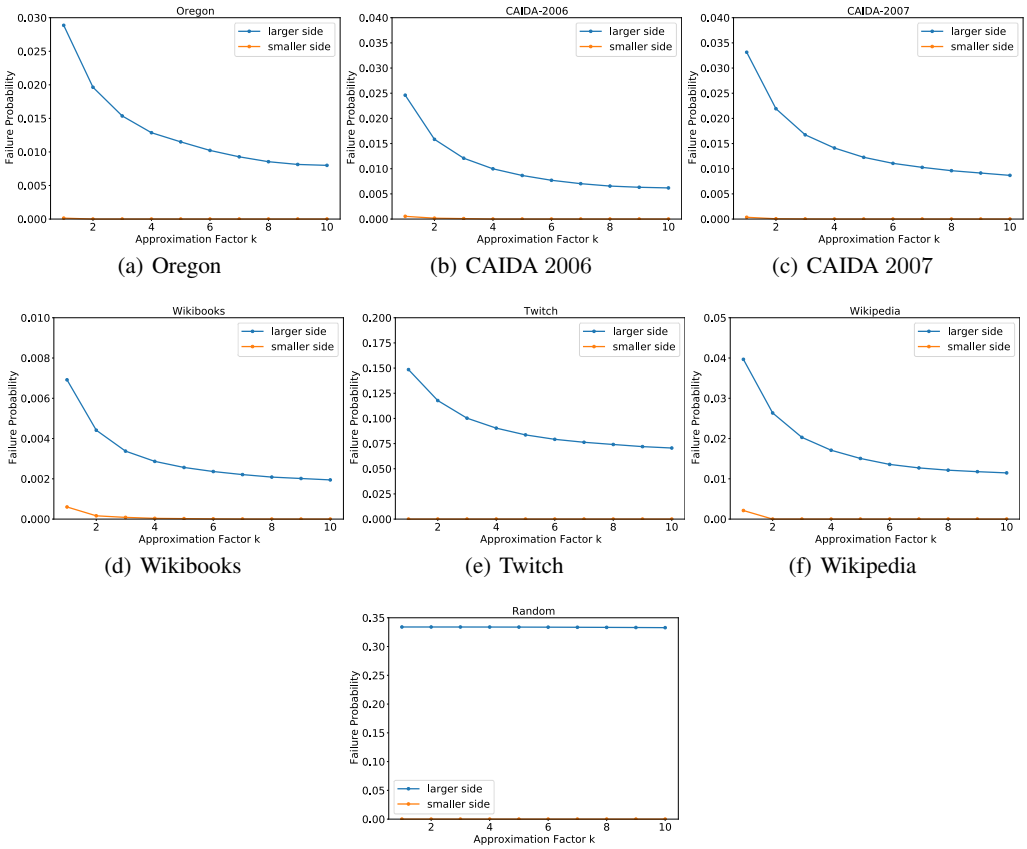
Value Prediction Oracle : We use the prediction of N_{xy} in Section 4 as a value prediction for $R_{xy}(x \leq_s y)$, under a fixed random vertex arrival order. The results are shown in Figure 8. For a fixed approximation factor k, we compute the failure probability δ of the value prediction oracle as follows: $\delta = \#/m$, where m is the number of total edges and $\#$ equals to the number of edges e that $p(e) \geq k\alpha R_e + \beta$ or $p(e) \leq \frac{1}{k}R_e - \beta$, respectively. Here we set $\alpha = 1, \beta = 10$ for all graph datasets.

We can see for the smaller side, there are very few edges e such that $p(e) \leq \frac{1}{k}R_e - \beta$. This meets the assumption of the exponential decay tail bound of the error. For the larger side, it also meets the assumption of the linear decay tail bound on most of the graph datasets.

F.8 Details on Oracle Training Overhead

The overhead of the oracles used in our experiments vary from task to task. For the important use case illustrated by the Oregon and CAIDA datasets in which we are interested in repeatedly counting triangles over many related streams, we can pay a small upfront cost to create the oracle which can be reused over and over again. Thus, the time complexity of building the oracle can be amortized over many problem instances, and the space complexity of the oracle is relatively small as we only need to store the top 10% of heavy edges from the first graph (a similar strategy is used in prior work on learning-augmented algorithms in Hsu et al. (2019b)). To give more details, for this snapshot oracle, we simply calculate the N_e values for all edges only in the first graph. We then keep the heaviest edges to form our oracle. Note that this takes polynomial time to train. The time complexity of using this oracle in the stream is as follows: when an edge in the stream comes, we simply check if it's among the predicted heavy edges in our oracle. This is a simple lookup which can even be done in constant time using hashing.

For learning-based oracles like the linear regression predictor, we similarly need to pay an upfront cost to train the model, but the space required to store the trained model depends on the dimension of the edge features. For the Reddit dataset with ~ 300 features, this means that the storage cost for the oracle is a small fraction of the space of the streaming algorithm. Training of this oracle can be done in polynomial time and can even be computed in a stream via sketching and sampling



(a) Oregon (b) CAIDA 2006 (c) CAIDA 2007

(d) Wikibooks (e) Twitch (f) Wikipedia

(g) Powerlaw

Figure 8: Failure probability as a function of approximation factor k for various graph datasets.

techniques which reduce the number of constraints from m (number of edges) to roughly linear in the number of features.

Our expected value oracle exploits the fact that our input graph is sampled from the CLV random graph model. Given this, we can explicitly calculate the expected value of N_e for every edge which requires no training time and nearly constant space. For training details for our link prediction model, see Section F.2. Note that in general, there is a wide and rich family of predictors which can be trained using sublinear space or in a stream such as regression Woodruff (2014), classification for example using SVMs Andoni et al. (2020); Rai et al. (2009) and even deep learning modelsGomes et al. (2019).

G Implicit Predictor in Prior Works

We prove that the first pass of the two pass triangle counting Algorithm given in Section 3.1 of McGregor et al. (2016), satisfies the conditions of the K-noisy oracle in Definition 1.1. Therefore, our work can be seen as a generalization of their approach when handling multiple related data sets, where instead of performing two passes on each data set and using the first of which to train the heavy edge oracle, we perform the training once according to the first related dataset, and we get a one pass algorithm for all remaining sets.

We first recall the first pass of (McGregor et al., 2016, Section 3.1):

1. Sample each node z of the graph with probability $p = C\epsilon^{-2}\log m/\rho$ for some large constant $C > 0$. Let Z be the set of sampled nodes.

2. Collect all edges incident on the set Z.
3. For any edge $e = \{u, v\}$, let $\widetilde{t}(e) = |\{z \in Z : u, v \in N(z)\}|$ and define the oracle as

$$\text{oracle}(e) = \begin{cases} \text{LIGHT} & \text{if } \widetilde{t}(e) < p \cdot \rho \\ \text{HEAVY} & \text{if } \widetilde{t}(e) \geq p \cdot \rho. \end{cases}$$

Lemma G.1. *For any edge* $e = (u, v)$, $\text{oracle}(e) = \text{LIGHT}$ *implies* $N_e \leq 2\rho$ *and* $\text{oracle}(e) = \text{HEAVY}$ *implies* $N_e > \rho/\sqrt{2}$ *with failure probability at most* $1/n^{10}$.

Proof. For any edge e, it follows that $\widetilde{t}(e) \sim \text{Bin}(N_e, p)$. Therefore if $N_e > 2\rho$,

$$\Pr[\widetilde{t}(e) < p \cdot \rho] \leq \exp(-\Omega(p \cdot \rho)) \leq 1/n^{10}$$

by picking C large enough in the definition of p. The other case follows similarly. □

Lemma G.2. *The expected space used by the above oracle is* $O(pm)$.

Proof. Each vertex is sampled in Z with probability p and we keep all of its incident edges. Therefore, the expected number of edges saved is $O(p \sum_v d_v) = O(pm)$. □

Note that the oracle satisfies the conditions of the K-noisy oracle in Definition 1.1. For example, if $N_e \geq C'\rho$ for $C' \gg 1$, Definition 1.1 only assumes that we incorrectly classify e with probability $1/C'$, whereas the oracle presented above incorrectly classifies e with probability $\exp(-C')/n^{10}$ which is much smaller than the required $1/C'$.

H Learnability Results

In this section, we give formal learning bounds for efficient predictor learning for edge heaviness. In particular, we wish to say that a good oracle or predictor for edge heaviness and related graph parameters can be learned efficiently using few samples if we observe graph instances drawn from a distribution. We can view the result of this section, which will be derived via the PAC learning framework, as one formalization of data driven algorithm design. Our results are quite general but we state simple examples throughout the exposition for clarity. Our setting is formally the following.

Suppose there is an underlying distribution $\mathcal{D}$ which generates graph instances $H_1, H_2, \cdots$ all on n vertices. Note that this mirrors some of our experimental setting, in particular our graph datasets which are similar snapshots of a dynamic graph across time.

Our goal is to efficiently learn a good predictor f among some family of functions $\mathcal{F}$. The input of each f is a graph instance H and the output is a feature vector in k dimensions. The feature vector represents the prediction of the oracle and can encapsulate a variety of different meanings. One example is when $k = |E|$ and f outputs an estimate of edge heaviness for all edges. Another is when $k << |E|$ and f outputs the id's of the k heaviest edges. We also think of each input instance H as encoded in a vector in $\mathbb{R}^p$ for $p \geq \binom{n}{2}$ (for example, each instance is represented as an adjacency matrix). Note that we allow for $p > \binom{n}{2}$ if for example, each edge or vertex for $H \sim \mathcal{D}$ also has an endowed feature vector.

To select the 'best' f, we need to precisely define the meaning of best. Note that in many settings, this involves a loss function which captures the quality of a solution. Indeed, suppose we have a loss function $L : f \times H \to \mathbb{R}$ which represents how well a predictor f performs on some input H. An example of L could be squared error from the output of f to the true edge heaviness values of edges in H. Note that such a loss function clearly optimizes for predicting the heavy edges well.

Our goal is to learn the best function $f \in \mathcal{F}$ which minimizes the following objective:

$$\mathbb{E}_{H \sim D}[L(f, H)]. \tag{1}$$

Let f^* be such the optimal $f \in \mathcal{F}$, and assume that for each instance H and each $f \in F$, $f(H)$ can be computed in time $T(p, k)$. For example, suppose graphs drawn from $\mathcal{D}$ possess edge features in $\mathbb{R}^d$, and that our family $\mathcal{F}$ is parameterized by a single vector $\theta \in \mathbb{R}^d$ and represents linear functions

which outputs the dot product of each edge feature with θ. Then it is clear that $T(p,k)$ is a (small) polynomial in the relevant parameters.

Our main result is the following.

Theorem H.1. *There is an algorithm which after poly$(T(p,k), 1/\epsilon)$ samples, returns a function $\hat{f}$ that satisfies*

$$\mathbb{E}_{H\sim D}[L(\hat{f}, H)] \leq \mathbb{E}_{H\sim D}[L(f^*, H)] + \epsilon$$

with probability at least $9/10$.

We remark that one can boost the probability of success to $1-\delta$ by taking additional $\log(1/\delta)$ multiplicative samples.

The above theorem is a PAC-style bound which shows that only a small number of samples are needed in order to ensure a good probability of learning an approximately-optimal function $\hat{f}$. The algorithm to compute $\hat{f}$ is the following: we simply minimize the empirical loss after an appropriate number of samples are drawn, i.e., we perform empirical risk minimization. This result is proven by Theorem H.3. Before introducing it, we need to define the concept of pseudo-dimension for a function class which is the more familiar VC dimension, generalized to real functions.

Definition H.2 (Pseudo-Dimension, Definition 9 Lucic et al. (2018))**.** *Let $\mathcal{X}$ be a ground set and $\mathcal{F}$ be a set of functions from $\mathcal{X}$ to the interval $[0,1]$. Fix a set $S = \{x_1, \cdots, x_n\} \subset \mathcal{X}$, a set of real numbers $R = \{r_1, \cdots, r_n\}$ with $r_i \in [0,1]$ and a function $f \in \mathcal{F}$. The set $S_f = \{x_i \in S \mid f(x_i) \geq r_i\}$ is called the induced subset of S formed by f and R. The set S with associated values R is shattered by $\mathcal{F}$ if $|\{S_f \mid f \in \mathcal{F}\}| = 2^n$. The* pseudo-dimension *of $\mathcal{F}$ is the cardinality of the largest shattered subset of $\mathcal{X}$ (or ∞).*

The following theorem relates the performance of empirical risk minimization and the number of samples needed, to the notion of pseudo-dimension. We specialize the theorem statement to our situation at hand. For notational simplicity, we define $\mathcal{A}$ be the class of functions in f composed with L:

$$\mathcal{A} := \{L \circ f : f \in \mathcal{F}\}.$$

Furthermore, by normalizing, we can assume that the range of L is equal to $[0,1]$.

Theorem H.3 (Anthony & Bartlett (1999))**.** *Let $\mathcal{D}$ be a distribution over graph instances and $\mathcal{A}$ be a class of functions $a : H \to [0,1]$ with pseudo-dimension $d_{\mathcal{A}}$. Consider t i.i.d. samples $H_1, H_2, \ldots, H_t$ from $\mathcal{D}$. There is a universal constant c_0, such that for any $\epsilon > 0$, if $t \geq c_0 \cdot d_{\mathcal{A}}/\epsilon^2$, then we have*

$$\left| \frac{1}{t} \sum_{i=1}^{t} a\left(H_i\right) - \mathbb{E}_{H\sim\mathcal{D}}\, a(H) \right| \leq \epsilon$$

for all $a \in \mathcal{A}$ with probability at least $9/10$.

The following corollary follows from the triangle inequality.

Corollary H.4. *Consider a set of t independent samples $H_1, \ldots, H_t$ from $\mathcal{D}$ and let $\hat{a}$ be a function in $\mathcal{A}$ which minimizes $\frac{1}{t}\sum_{i=1}^{t} a(H_i)$. If the number of samples t is chosen as in Theorem H.3, then*

$$\mathbb{E}_{H\sim D}[\hat{a}(H)] \leq \mathbb{E}_{H\sim D}[a^*(H)] + 2\epsilon$$

holds with probability at least $9/10$.

The main challenge is to bound the pseudo-dimension of our given function class $\mathcal{A}$. To do so, we first relate the pseudo-dimension to the VC dimension of a related class of threshold functions. This relationship has been fruitful in obtaining learning bounds in a variety of works such as Lucic et al. (2018); Izzo et al. (2021).

Lemma H.5 (Pseudo-dimension to VC dimension, Lemma 10 in Lucic et al. (2018))**.** *For any $a \in \mathcal{A}$, let B_a be the indicator function of the region on or below the graph of a, i.e., $B_a(x,y) = sgn(a(x) - y)$. The pseudo-dimension of $\mathcal{A}$ is equivalent to the VC-dimension of the subgraph class $B_{\mathcal{A}} = \{B_a \mid a \in \mathcal{A}\}$.*

Finally, the following theorem relates the VC dimension of a given function class to its computational complexity, i.e., the complexity of computing a function in the class in terms of the number of operations needed.

Lemma H.6 (Theorem 8.14 in Anthony & Bartlett (1999)). *Let $w : \mathbb{R}^\alpha \times \mathbb{R}^\beta \to \{0, 1\}$, determining the class*

$$\mathcal{W} = \{x \to w(\theta, x) : \theta \in \mathbb{R}^\alpha\}.$$

Suppose that any w can be computed by an algorithm that takes as input the pair $(\theta, x) \in \mathbb{R}^\alpha \times \mathbb{R}^\beta$ and returns $w(\theta, x)$ after no more than r of the following operations:

- *arithmetic operations $+, -, \times$, and $/$ on real numbers,*
- *jumps conditioned on $>, \geq, <, \leq, =$, and $=$ comparisons of real numbers, and*
- *output $0, 1$,*

then the VC dimension of $\mathcal{W}$ is $O(\alpha^2 r^2 + r^2 \alpha \log \alpha)$.

Combining the previous results allows us prove Theorem H.1. At a high level, we are instantiating Lemma H.6 with the complexity of *computing* any function in the function class $\mathcal{A}$.

Proof of Theorem H.1. First by Theorem H.3 and Corollary H.4, it suffices to bound the pseudo-dimension of the class $\mathcal{A} = L \circ \mathcal{F}$. Then from Lemmas H.5, the pseudo-dimension of $\mathcal{A}$ is the VC dimension of threshold functions defined by $\mathcal{A}$. Finally from Lemma H.6, the VC dimension of the appropriate class of threshold functions is polynomial in the complexity of computing a member of the function class. In other words, Lemma H.6 tells us that the VC dimension of $B_{\mathcal{A}}$ as defined in Lemma H.5 is polynomial in the number of arithmetic operations needed to compute the threshold function associated to some $a \in \mathcal{A}$. By our definition, this quantity is polynomial in $T(p, k)$. Hence, the pseudo-dimension of $\mathcal{G}$ is also polynomial in $T(p, k)$ and the result follows. □

Note that we can consider initializing Theorem H.1 with specific predictions. If the family of oracles we are interested in is efficient to compute (which is the case of the predictors we employ in our experiments), then Theorem H.1 assures us that only polynomially many samples are required (in terms of the computational complexity of our function class), to be able to learn a nearly optimal oracle. Furthermore, computing the empirical risk minimizer needed in Theorem H.1 is also efficient for a wide verity of function classes. For example in practice, we can simply use gradient descent or stochastic gradient descent for a range of predictor models, such as regression or even general neural networks.

We remark that our learnability result is in similar in spirit to one given in the recent learning-augmented paper Dinitz et al. (2021). There, they derive sample complexity learning bounds for the different algorithmic problem of computing matchings in a graph (not in a stream). Since they specialize their analysis to a specific function class and loss function, their bounds are tighter compared to the possibly loose polynomial bounds we have stated. However, our analysis above is more general as it allows for a variety of predictors and loss functions to measure the quality of the predictions.

Knowledge and Information Systems (2024) 66:1917–1954
https://doi.org/10.1007/s10115-023-02001-6

REGULAR PAPER

ALICE and the Caterpillar: A more descriptive null model for assessing data mining results

Giulia Preti[1] · **Gianmarco De Francisci Morales**[1] · **Matteo Riondato**[2]

Received: 20 January 2023 / Revised: 15 May 2023 / Accepted: 2 October 2023 /
Published online: 2 November 2023
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2023

Abstract
We introduce novel null models for assessing the results obtained from observed binary transactional and sequence datasets, using statistical hypothesis testing. Our null models maintain more properties of the observed dataset than existing ones. Specifically, they preserve the Bipartite Joint Degree Matrix of the bipartite (multi-)graph corresponding to the dataset, which ensures that the number of caterpillars, i.e., paths of length three, is preserved, in addition to other properties considered by other models. We describe ALICE, a suite of Markov chain Monte Carlo algorithms for sampling datasets from our null models, based on a carefully defined set of states and efficient operations to move between them. The results of our experimental evaluation show that ALICE mixes fast and scales well, and that our null model finds different significant results than ones previously considered in the literature.

Keywords Hypothesis testing · Markov Chain Monte Carlo methods · Sequence datasets · Significant pattern mining · Swap randomization · Transactional datasets

1 Introduction

Binary transactional datasets and sequence datasets are the object of study in several areas, from marketing to network analysis, to finance modeling, processing of satellite images, and

"One side will make you grow taller, and the other side will make you grow shorter." — The Caterpillar, *Alice in Wonderland*.

✉ Giulia Preti
giulia.preti@centai.eu

Gianmarco De Francisci Morales
gdfm@acm.org

Matteo Riondato
mriondato@amherst.edu

1 CENTAI, Corso Inghilterra 3, 10138 Turin, TO, Italy

2 Department of Computer Science, Amherst College, 25 East Drive, Amherst, MA 01002, USA

more. In genomics, for example, transactions represent individuals and the items in a transaction represent their gene mutations. Many fundamental data mining tasks can be defined on them, such as frequent itemset/sequence mining, clustering, and anomaly detection.

The goal of knowledge discovery from a dataset is not simply to analyze the dataset, but to obtain *new understanding* of the stochastic, often noisy, *process that generated the dataset*. Such novel insights can only be obtained by subjecting the results of the analysis to a rigorous validation, which allows to separate those results that give new information about the process from those that are due to the randomness of the process itself. This kind of validation is actually necessary in many scientific fields, for example in microbiology and genomics, when the observed dataset represents individuals with their gene mutations, or protein interactions [15, 49, 53].

The *statistical hypothesis testing* framework [32] is a very rigorous validation process for the results obtained from an observed dataset. Hypotheses about the results are formulated and then tested by comparing the results (or appropriate statistics about them) to their distribution over the *null model*, i.e., a set of datasets enriched with a user-specified probability distribution (see Sect. 3.2), which contains all and only the datasets that preserve a user-specified subset of the properties of the observed dataset (e.g., the size, or some cumulative statistics). The testing of hypotheses requires, in *resampling-based methods* [66], to be able to efficiently draw multiple datasets from the null model. These samples are then used to obtain an approximation of the distribution of results from the null model, to which the actually observed results are compared. When the probability of obtaining results as or more extreme than those observed is low, the observed results are deemed *statistically significant*, i.e., they are deemed to give previously unknown information about the data-generating process.

Informally, the properties preserved by the null model, and the sampling distribution, capture the existing or assumed knowledge about the process that generated the observed dataset. Testing the hypotheses can be understood as trying to ascertain whether the observed results can be explained by the existing knowledge. The choice of the null model must be made by the user, based on their domain knowledge, and should be deliberate. Null models that capture more properties of the observed dataset are usually more descriptive and therefore to be preferred. The challenge in using such models is the need for efficient computational procedures to draw datasets from the null model according to the user-specified distribution, as many such sampled datasets are necessary to test complex or multiple hypotheses.

Contributions

We study the problem of assessing results obtained from an observed binary[1] transactional or sequence dataset by performing statistical hypothesis tests via resampling methods from a descriptive null model. Specifically, our contributions are the following.

- We introduce novel null models (Sects. 4 and 6.2) that preserve additional properties of the observed dataset than those preserved by existing null models [17, 60]. Specifically, all datasets in our null models have the same *Bipartite Joint Degree Matrix (BJDM)* of the bipartite (multi-)graph corresponding to the observed dataset (Sects. 4.1 and 4.2). Maintaining the BJDM captures additional "structure" of the observed dataset: e.g., on transactional datasets, in addition to dataset size, transaction lengths, and item or itemset supports, the number of *caterpillars* in the observed dataset is also preserved (Lemma 3). We also explain why more natural properties, such as the supports of itemsets of length two on transactional datasets, are not as informative as one may think.

[1] In the rest of the work, we drop the attribute "binary": all datasets we refer to are binary.

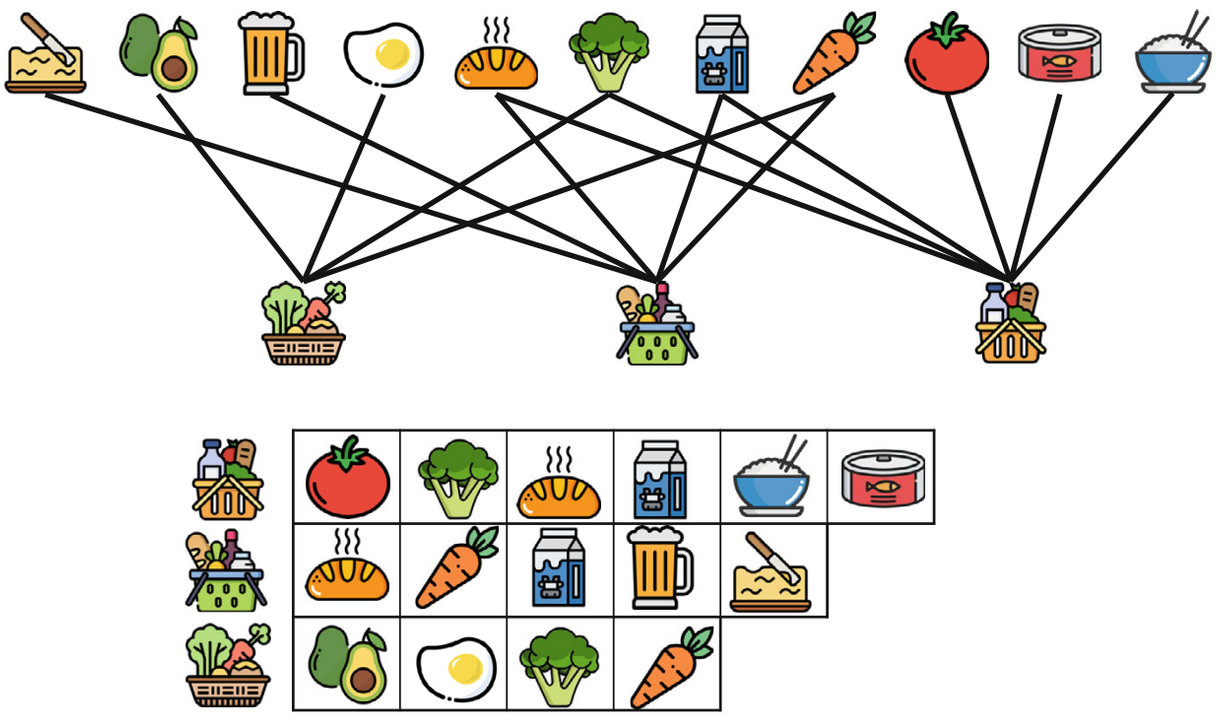


Fig. 1 A dataset of shopping baskets (lower) and the respective bipartite graph (upper)

- We present ALICE,[2] a suite of Markov chain Monte Carlo algorithms for sampling datasets from our null models according to a user-specified distribution. ALICE- A (Sect. 5.1) is based on *Restricted Swap Operations (RSOs)* on biadjacency matrices, which preserve the BJDM. Our contributions include a sampling algorithm to draw such RSOs much more efficiently than with the natural rejection sampling approach. A second algorithm, ALICE- B, (Sect. 5.2) adapts the CURVEBALL approach [11, 62] to RSOs, to essentially perform multiple RSOs at every step, thus leading to faster mixing. Finally, ALICE- S samples from the null model for sequence datasets, using Metropolis-Hastings and a variant of RSOs, to take into account the fact that the bipartite graph corresponding to a sequence dataset is a *multi-graph*.
- The results of our experimental evaluation show that ALICE mixes fast, it is scalable as the dataset grows, and that our new null model differs from previous ones, as it marks different results as significant.

The present article extends the conference version [48] in multiple ways, including:

- The extension to sequence datasets and the development of ALICE- S (Sect. 6) is entirely new. In addition to introducing a novel null model and algorithm, to the best of our knowledge, our work is the first to look at sequence datasets as bipartite multi-graphs, which is a generic representation that can be used in other works.
- We give an explicit counterexample (Fig. 2) showing that preserving the number of caterpillars and other fundamental properties is not sufficient to preserve the BJDM, while the opposite is true (Sect. 4.2).
- We include a discussion of Gram mates [29, 30], to explain why a model preserving the supports of itemsets of length two may not be very interesting.
- We add examples and figures to help the understanding of important concepts.

[2] Like the eponymous character of *Alice in Wonderland* our algorithms explore a large strange world, and interact with caterpillars.

Outline After discussing related work in Sect. 2, we focus the presentation on binary transactional datasets, with preliminaries (Sect. 1) also covering statistical hypothesis testing. Then we describe the null model for transactional datasets (Sect. 4), and then the two algorithms to sample datasets from this null model (Sect. 5). Covering first only transactional datasets allows us to discuss sequence datasets, the null model, and the specific algorithm for this case in Sect. 6. Our experimental evaluation and its results are presented in Sect. 7.

2 Related work

The need for statistically validating results from transactional datasets was understood immediately after the first efficient algorithm for obtaining these results was introduced [10, 36]. A long line of works also studies how to filter out uninteresting patterns, or directly mine *interesting* ones [64]. This direction is orthogonal to the study of the *statistical validity* of the results, which is our focus.

Many works concentrate on the case of *labeled* transactional datasets [14, 22, 31, 34, 37, 43, 44, 46, 56–58, 67], where each transaction comes with a binary label. Most of these works use resampling-based approaches, as we do, but the very different nature of the studied tasks and data, as we study the *unlabeled* case, makes them inapplicable to our problems. We refer to the tutorial by Pellegrina et al. [45] for a detailed survey of the work done in *unlabeled datasets*, including resampling methods. The different nature of the data makes these approaches inapplicable to our case.

Most work has been on mining *significant frequent itemsets*, tiles, or association rules [21, 33, 65]. The survey by by Hämäläinen and Webb [23] presents many of these works in depth. The most relevant to ours are those by Gionis et al. [17] and Hanhijärvi [24], who present resampling methods for drawing transactional datasets from a null model which preserves the number of transactions, the transaction lengths, and the item supports as in an observed dataset. These approaches, like ours, can be used for testing any result from transactional datasets, not just for significant pattern mining. We present a null model that is more descriptive than the ones studied in these works, because it preserves additional properties of the observed dataset. Bie [6] proposes a method to *uniformly* sample datasets from a null model that preserves, *in expectation*, the same constraints. While it can partially be extended to preserve the constraints exactly, it cannot be used to sample according to any user-specified distribution, which we believe to be a fundamental ingredient of the null model, as it includes already available knowledge of the data-generating process *in addition to* the constraints.

Assessing results obtained from sequence datasets has also generated interest [27, 47, 60]. We refer the interested reader for an in-depth discussion of related work in this area to [27, Sect. 2]. To the best of our knowledge, we are the first to look at sequence datasets as bipartite *multi*-graphs, and to propose a null model that explicitly preserves properties of such multi-graphs. Our null model for sequence datasets preserves additional properties than the one introduced by Tonon and Vandin [60], similarly to how our null model for transactional datasets preserves additional properties than the one by Gionis et al. [17], as indeed the Tonon and Vandin's model is essentially an adaptation of the Gionis et al's model to sequence datasets. Tonon and Vandin [60] and Jenkins et al. [27] present other null models for sequence datasets. Extending these models to preserve the additional properties we consider is an interesting direction for future work.

Beyond binary transactional and sequence datasets, resampling methods for assessing data mining results have been proposed for graphs [20, 25, 54, 55], real-valued and mixed-valued matrices [40], and database tables [41]. None of these works proposes a null model similar to the one we introduce, nor presents similar sampling algorithms. Our approach can be a starting point to develop more descriptive null models for these richer types of data.

ALICE, our algorithm for sampling from a null model of datasets, can also be seen as sampling from the set of bipartite graphs with a prescribed BJDM, according to a desired sampling distribution. In this sense, our contributions belong to a long line of works that studies how to generate (bipartite) graphs with prescribed properties and according to a desired probability distribution [3, 4, 8, 12, 16, 19, 28, 42, 50, 52, 54, 59, 61]. The surveys by Cimini et al. [12], Bonifati et al. [8], and Greenhill [19] give complete coverage of this field. These approaches have been studied in the context of complex networks, while we use *bipartite* graphs to represent transactional datasets, and our main goal is to statistically assess results obtained from such datasets, not to study the properties of the graphs.

No previous work on sampling bipartite graphs deals with the question we study. Saracco et al. [52] presented a configuration model to sample bipartite networks that, *in expectation*, have the same degree sequences as a prescribed one. ALICE *exactly* maintains the BJDM, which preserves the exact degree sequences, and also other additional properties (see Sect. 4); thus, our null model preserves more characteristics of the observed dataset. Aksoy et al. [4] proposed a method to generate bipartite networks that preserve also the clustering coefficient, which is not related to the BJDM. Amanatidis et al. [5] give necessary and sufficient conditions for a matrix to be the BJDM of a bipartite graph. We always start from such a matrix, so we do not have to address its realizability. The concept of *Restricted Swap Operation (RSO)* was introduced by Czabarka et al. [13], but not for the purpose used in ALICE. Boroojeni et al. [9] present randomized algorithms to generate a bipartite graph from a BJDM, but there is no proof that their approaches can generate all possible graphs with that BJDM nor there is an analysis on the probability that such a graph is generated. Both aspects are important in order to use the samples for statistical hypothesis testing (see Sect. 3.2), and ALICE achieves these goals.

We are interested in sampling graphs (but really, datasets) from a set of graphs that preserve the same properties as some observed graph (i.e., dataset). This task is different from the problem of generating a graph from a random family, such as Erdős-Rényi graphs, stochastic block models, Kronecker graphs, preferential attachment graphs, and others, or fitting the parameters of such a family on the basis of one or more observed graphs.

3 Preliminaries

We now define the key concepts and notation used in this work. Table 1 summarizes the most important notation. Preliminaries for sequence datasets are deferred to Sect. 6.1.

3.1 Transactional datasets

Let $\mathcal{I} \doteq \{a_1, \ldots, a_{|\mathcal{I}|}\}$ be a finite alphabet of *items*. W.l.o.g., we can assume $\mathcal{I} = \{1, \ldots, |\mathcal{I}|\}$. Any $A \subseteq \mathcal{I}$ is an *itemset*. A *transactional dataset*[3] $\mathcal{D}$ is a finite bag of itemsets, which are known also as *transactions* when considered as the elements of a dataset. The *size* $|\mathcal{D}|$ of

[3] From here to the end of Sect. 5, we only discuss *transactional* datasets, so we drop the attribute and just refer to them as "datasets".

Table 1 Table of symbols

	Symbol	Description
Dataset	$\mathcal{I}$	Set of items
	S	Ordered list of itemsets
	$\mathcal{D}$	Dataset (bag of itemsets in the transactional case)
		(bag of sequences in the sequence case)
	$M_{\mathcal{D}}$	Binary matrix associated to the transactional dataset $\mathcal{D}$
	$\mathsf{mat}(\mathcal{D})$	Set of binary matrices associated to the transactional dataset $\mathcal{D}$
	$\mathsf{dat}(M)$	Transactional dataset whose binary matrix is M
	$\mathring{\mathcal{D}}$	Observed dataset
Bipartite (multi-)Graph	G	Bipartite (multi-)graph
	$L \cup R$	Set of left (L) and right (R) vertices of G
	E	Set of (multi-)edges of G
	$\mathcal{G}$	Set of bipartite multi-graphs
	$\Gamma(v)$	Set of nodes connected to v in G
	J_G	Bipartite Joint Degree Matrix (BJDM) of G
	$\mathsf{z}(G)$	Number of simple paths of length 3 (caterpillars) in G
	$\mathcal{M}$	Set of binary matrices of graphs with the same BJDM
Null model	Π	Null model
	$\mathcal{Z}$	Set of datasets sharing some properties of $\mathring{\mathcal{D}}$
	π	Probability distribution over $\mathcal{Z}$
	$p_{\mathring{\mathcal{D}},H_0}$	p value of a null hypothesis H_0 involving Π and $\mathring{\mathcal{D}}$

the dataset is the number of transactions it contains. The *length* $|t|$ of a transaction $t \in \mathcal{D}$ is the number of items in it. Figure 1 (lower) shows a dataset of shopping baskets with three baskets (transactions) of length 6, 5, and 4, respectively.

For any itemset $A \subseteq \mathcal{I}$, the *support* $\sigma_{\mathcal{D}}(A)$*of* A *in* $\mathcal{D}$ is the number of transactions of $\mathcal{D}$ which contain A:

$$\sigma_{\mathcal{D}}(A) \doteq |\{t \in \mathcal{D} : A \subseteq t\}| \ .$$

The support is a natural (albeit not without drawbacks) measure of interestingness. A foundational knowledge discovery task requires to find, given a *minimum support threshold* $\theta \in [0, |\mathcal{D}|]$, the collection $\mathsf{Fl}_\theta(\mathcal{D})$ of *Frequent Itemsets (FIs) in* $\mathcal{D}$*w.r.t.* θ: $\mathsf{Fl}_\theta(\mathcal{D}) \doteq \{A \subseteq \mathcal{I} : \sigma_{\mathcal{D}}(A) \geq \theta\}$ [2]. Given $\theta = 2$, for $\mathcal{D}$ in Fig. 1 (lower), $\mathsf{Fl}_\theta(\mathcal{D})$ contains the itemsets { carrot }, { broccoli }, { bread }, { milk }, and { bread , milk }.

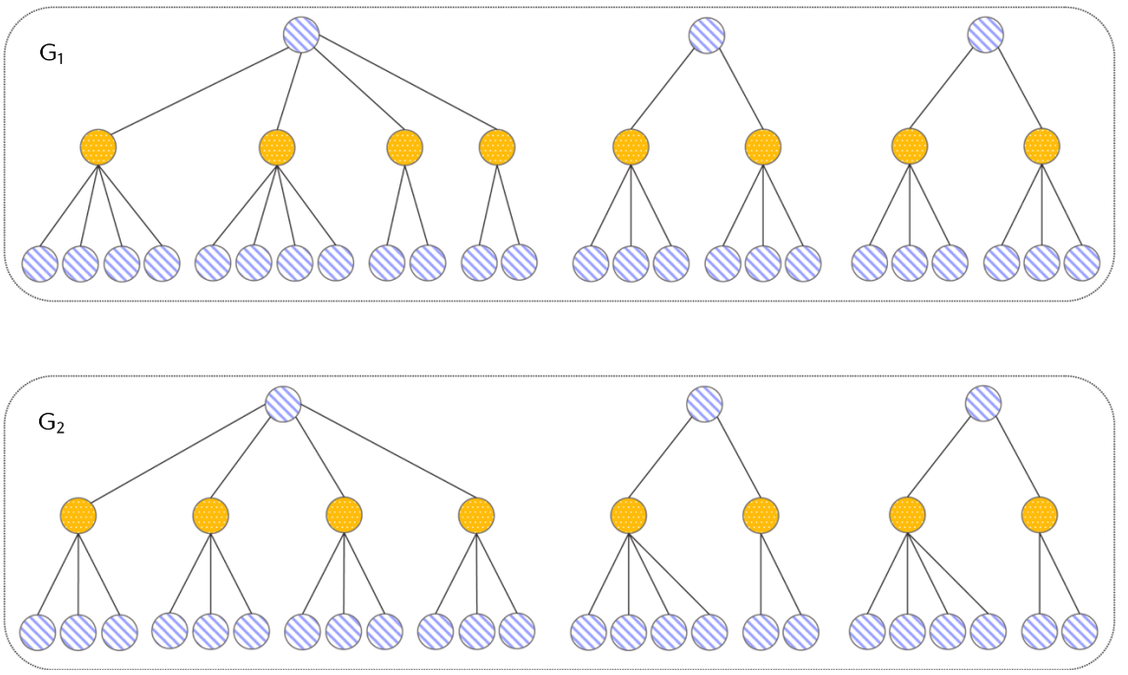


Fig. 2 Two bipartite graphs with the same degree distributions and the same number of caterpillars, but different BJDMs

3.2 Null models and hypothesis testing

The statistical hypothesis testing framework [32] allows to rigorously understand whether the results obtained from an *observed dataset* $\mathring{\mathcal{D}}$ (e.g., the collection of frequent itemsets, or its size, among many others) are actually interesting or are just due to randomness in the (unknown, at least partially) data generation process. Informally, the observed results are compared to the distribution of results that would be obtained from a *null model* (see below); if results as or more extreme than the observed ones are sufficiently unlikely, the observed results are deemed *statistically significant*.

A *null model* $\Pi = (\mathcal{Z}, \pi)$ is a pair where $\mathcal{Z}$ is a set of datasets, and π is a (user-specified) probability distribution over $\mathcal{Z}$. The datasets in $\mathcal{Z}$ are all and only those that share some descriptive characteristics with an *observed dataset* $\mathring{\mathcal{D}}$, which also belongs to $\mathcal{Z}$.[4] Null models in previous works [6, 17] preserve the following two *fundamental properties*:

- the distribution of the transaction lengths, i.e., for any possible transaction length $\ell \in [1, |\mathcal{I}|]$, $\mathcal{D} \in \mathcal{Z}$ contains the same number of transactions of length ℓ as $\mathring{\mathcal{D}}$[5]; and
- the support of the items, i.e., for any $i \in \mathcal{I}$ and $\mathcal{D} \in \mathcal{Z}$, $\sigma_{\mathcal{D}}(i) = \sigma_{\mathring{\mathcal{D}}}(i)$.

The intuition behind wanting to preserve some properties of $\mathring{\mathcal{D}}$ is that these properties, together with π, capture what is known or assumed about the process that generated the data, and the goal is to understand whether the results obtained from $\mathring{\mathcal{D}}$ are, informally, "typical" for datasets with these characteristics. Formally, given $\mathring{\mathcal{D}}$ and a null model $\Pi = (\mathcal{Z}, \pi)$, one formulates a *null hypothesis* H_0 involving Π and a result $R_{\mathring{\mathcal{D}}}$ obtained from $\mathring{\mathcal{D}}$. For example, let $R_{\mathring{\mathcal{D}}} = |\mathsf{Fl}_\theta(\mathring{\mathcal{D}})|$, and

$$H_0 \doteq \text{“} \mathop{\mathbb{E}}_{\mathcal{D} \sim \pi} [|\mathsf{Fl}_\theta(\mathcal{D})|] = R_{\mathring{\mathcal{D}}} \text{”}.^{6} \tag{1}$$

[4] Thus, Π depends on $\mathring{\mathcal{D}}$, but we hide it in the notation to keep it light.

[5] This property implies that the size of the dataset is preserved as well, i.e., $|\mathcal{D}| = |\mathring{\mathcal{D}}|$ for any $\mathcal{D} \in \mathcal{Z}$.

[6] This hypothesis is just one simple example of many possible different hypotheses that could be tested.

The hypothesis is then tested by computing the *p-value* $p_{\mathring{\mathcal{D}},H_0}$ *of* H_0, defined as the probability that, in a dataset $\mathcal{D}'$ sampled from $\mathcal{Z}$ according to π, the results $R_{\mathcal{D}'}$ (e.g., $|\mathsf{Fl}_\theta(\mathcal{D}')|$) are *more extreme* (e.g., larger) than $R_{\mathring{\mathcal{D}}}$, i.e.,

$$p_{\mathring{\mathcal{D}},H_0} \doteq \Pr_{\mathcal{D}'\sim\pi}(R_{\mathcal{D}'} \text{ more extreme than } R_{\mathring{\mathcal{D}}}) \ . \tag{2}$$

The notion of "more extreme" depends on the nature of $R_{\mathring{\mathcal{D}}}$. When $p_{\mathring{\mathcal{D}},H_0}$ is *not larger* than a user-specified *critical value* α, then the observed results $R_{\mathring{\mathcal{D}}}$ are deemed to be *statistically significant*, i.e., unlikely to be due to random chance (in other words, the null hypothesis H_0 is rejected as not sufficiently supported by the available data).

Computing the p-value $p_{\mathring{\mathcal{D}},H_0}$ from (2) exactly is often essentially impossible. E.g., for statistically sound knowledge discovery tasks on sequence datasets, the exact distribution of test statistics is known only in very restricted cases [47], while all other approaches use resampling [27, 60]. Thus, an empirical estimate $\tilde{p}_{\mathring{\mathcal{D}},H_0}$ is obtained as follows and used in place of $p_{\mathring{\mathcal{D}}}$ when testing the hypothesis [66]. Let $\mathcal{D}_1, \ldots, \mathcal{D}_T$ be T datasets *independently sampled* from $\mathcal{Z}$ according to π, then

$$\tilde{p}_{\mathring{\mathcal{D}},H_0} \doteq \frac{1 + |\{\mathcal{D}_i : R_{\mathcal{D}_i} \text{ is more extreme than } R_{\mathring{\mathcal{D}}}\}|}{1+T} \ . \tag{3}$$

Such *resampling methods*, of which the well-known bootstrap is also an instance, are often to be preferred to the explicit derivation of the statistics for multiple reasons:

- they are, in some sense, independent from the test being conducted, as the test statistic distribution (or better, the p-value) is estimated from the sampled datasets, as in (3);
- they leverage data-dependent distributional characteristics, which tend to result in higher statistical power; and
- they scale to high-dimensional settings.

In many knowledge discovery tasks, and in many applications such as during clinical trials for drug approvals [26], or in genomics studies [18], one is interested in testing *multiple hypotheses*. For example, *significant itemset mining* (see Sect. 2) requires testing one hypothesis

$$H_0^A \doteq \text{“} \mathop{\mathbb{E}}_{\mathcal{D}\sim\pi}[\sigma_{\mathcal{D}}(A)] = \sigma_{\mathring{\mathcal{D}}}(A)\text{”}$$

for each itemset A.[7] When testing multiple hypotheses, i.e., all hypotheses in a class $\mathcal{H}$, one is interested in ensuring that the *Family-Wise Error Rate*, i.e., the probability of making *any* false discovery, is at most a user-specified acceptable threshold δ. Classic methods for controlling the FWER, such as the Bonferroni correction [7], lack the *statistical power* to be useful in knowledge discovery settings, i.e., the probability that a *true* significant discovery is marked as such is very low, due to the large number $|\mathcal{H}|$ of hypotheses. *Resampling-based methods* [66] perform better for these tasks because they empirically estimate the distribution of the minimum p-value of the hypotheses in $\mathcal{H}$ by *sampling datasets from* $\mathcal{Z}$, and use this information to compute an *adjusted critical value* $\hat{\alpha}$.

For example, the Westfall–Young approach works as follows. Let $\mathcal{D}'_1, \ldots, \mathcal{D}'_T$ be T datasets *sampled independently* from $\mathcal{Z}$ according to π, and let

$$\check{p}_i \doteq \min_{h\in\mathcal{H}} p_{\mathcal{D}'_i,h} \tag{4}$$

[7] This hypothesis is one of many kinds of hypotheses that can be tested by using the support as the test statistic.

be the minimum p-value, on $\mathcal{D}'$, of any hypothesis $h \in \mathcal{H}$. The *adjusted critical value* $\hat{\alpha}$ to which the p-values of the hypotheses are compared is

$$\hat{\alpha} \doteq \max \left\{ \alpha : \frac{|\{\mathcal{D}'_i : \check{p}_i \leq \alpha\}|}{T} \leq \delta \right\} .$$

That is, $\hat{\alpha}$ is the largest $\alpha \in [0, 1]$ such that the fraction of the T datasets $\mathcal{D}'_i$ whose minimum p-value $\check{p}$ is at most α is not greater than δ. Estimates computed as in (3) are used in place of the exact p-values in the r.h.s. of (4). Comparing the (estimated) p-value of each hypothesis in $\mathcal{H}$ to $\hat{\alpha}$ guarantees that the FWER is at most δ. Thus, efficiently drawing random datasets from $\mathcal{Z}$ according to π plays a key role in statistical hypothesis testing. Our goal in this work is to develop efficient methods to sample a dataset from $\mathcal{Z}$ according to π where $\mathcal{Z}$ is the set of datasets that, in addition to preserving the aforementioned three properties from $\mathring{\mathcal{D}}$, also preserve an additional important characteristic property that we describe in Sect. 4.2.

3.3 Markov chain Monte Carlo methods

ALICE follows the *Markov chain Monte Carlo (MCMC) method*, and uses the *Metropolis–Hastings (MH) algorithm* [38, Ch. 7 and 10]. Next is an introduction tailored to our work.

Let $G = (V, E)$ be a directed, weighted, strongly connected, aperiodic graph, potentially with self-loops. The vertices V are known as *states* in this context. W.l.o.g., we can assume $V = \{1, 2, \ldots, |V|\}$. For any state v, let $\Gamma(v)$ be the set of (out-)neighbors of v, i.e., the set of states u such that $(v, u) \in E$ (it holds $v \in \Gamma(v)$ if there is a self-loop). For any neighbor $u \in \Gamma(v)$, the weight $\mathsf{w}(v, u)$ of the edge (v, u) is strictly positive, and it holds $\sum_{u \in \Gamma(v)} \mathsf{w}(v, u) = 1$. In other words, there is a probability distribution ξ_v over $\Gamma(v)$ such that $\xi_v(u) = \mathsf{w}(v, u)$. Let W be the $|V| \times |V|$ matrix such that $W[v, u] = \mathsf{w}(v, u)$ if $(v, u) \in E$, and 0 otherwise.[8]

Let $G = (V, E)$ be a directed, weighted, strongly connected, aperiodic graph, potentially with self-loops. The *Metropolis-Hastings (MH) algorithm* gives a way to sample an element of V according to a user-specified probability distribution ϕ. Let $v \in V$ be any state, chosen arbitrarily. We first draw a neighbor $u \in \Gamma(v)$ of v according to the distribution ξ_v. Then we "move" from v to u with probability

$$\min \left\{ 1, \frac{\phi(u)\xi_u(v)}{\phi(v)\xi_v(u)} \right\}, \tag{5}$$

otherwise, we stay in v. After a sufficiently large number of steps t, the state v_t is (either approximately or exactly) distributed according to ϕ and can be taken as a sample.

In summary, to be able to use MH, one must define the graph $G = (V, E)$, the neighbor sampling probability ξ_v for every $v \in V$, a procedure to sample a neighbor of v according to ξ_v, and the desired sampling distribution ϕ over V.

4 A more descriptive null model

As discussed in Sect. 3.2, a good null model should preserve important characteristics of the observed dataset $\mathring{\mathcal{D}}$, and we mentioned the two fundamental properties that were the focus of

[8] The strong-connectivity and aperiodicity of G, together with having $W[u, v] \geq 0$ iff $(u, v) \in E$, ensure that the Markov chain on V whose matrix of transition probabilities is W has a unique stationary distribution [38, Thm. 7.7].

previous work [6, 17]. We now introduce a null model that preserves an additional property, and then show efficient methods to sample datasets from it.

4.1 Datasets, matrices, and bipartite graphs

Before defining the additional characteristic quantity of $\mathring{\mathcal{D}}$ that we want to preserve, we must describe "alternative" representations of a dataset $\mathcal{D}$. The most natural one is a *binary matrix* $M_{\mathcal{D}}$ with $|\mathcal{D}|$ rows and $|\mathcal{I}|$ columns, where the (i, j) entry is 1 iff transaction $i \in \mathcal{D}$ contains item $j \in \mathcal{I}$, and where the order of the transactions (i.e., of the rows) is arbitrary [17, Sect. 4.1]. Since the order is arbitrary, there are *multiple matrices* that correspond to the same dataset, differing by the ordering of the rows. This fact is of key importance for the correctness of methods that sample datasets (and not matrices) from a null model, i.e., that are *row-order agnostic* [1].

Any matrix $M_{\mathcal{D}}$ corresponding to $\mathcal{D}$ can be seen as the *biadjacency matrix* of an *undirected bipartite graph* $G_{\mathcal{D}} = (\mathcal{D} \cup \mathcal{I}, E)$ corresponding to $\mathcal{D}$, where there is an edge[9] $(t, i) \in E$ iff transaction t contains the item i. Figure 1 (upper) depicts the bipartite graph corresponding to the dataset in the lower part of the figure. The left nodes (bottom nodes) model the three shopping baskets, while the right nodes (top nodes) represent the product bought. Different matrices M' and M'' corresponding to $\mathcal{D}$ are the biadjacency matrices of bipartite graphs that are *structurally equivalent*, up to the labeling of the transactions in $\mathcal{D}$. In other words, all graphs corresponding to a dataset share the *same structural properties*, no matter their biadjacency matrices. To define our new null model, we use the graph $G_{\mathring{\mathcal{D}}}$.

4.2 Preserving the Bipartite joint degree matrix

One of our goals is to define a null model $\Pi = (\mathcal{Z}, \pi)$ such that the datasets in $\mathcal{Z}$ preserve not only the two fundamental properties, but also an additional descriptive property of $\mathring{\mathcal{D}}$: the *Bipartite Joint Degree Matrix (BJDM)* $\mathsf{J}_{G_{\mathring{\mathcal{D}}}}$ of its bipartite graph representation $G_{\mathring{\mathcal{D}}}$.

Definition 1 (*BJDM*) Let $G = (L \cup R, E)$ be a bipartite graph, k_L and k_R be the largest degree of a node in L and R, respectively. The *Bipartite Joint Degree Matrix (BJDM)* J_G*of* G, is a $k_L \times k_R$ matrix whose (i, j)-th entry $\mathsf{J}_G[i, j]$ is the number of edges connecting a node $u \in L$ with degree $\mathsf{deg}(u) = i$ to a node $v \in R$ with degree $\mathsf{deg}(v) = j$, i.e.,

$$\mathsf{J}_G[i, j] \doteq |\{(u, v) \in E : \mathsf{deg}(u) = i \wedge \mathsf{deg}(v) = j\}| \ .$$

The BJDM of the graph in Fig. 1 (upper) is the following:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 2 & 2 \\ 2 & 3 \\ 3 & 3 \end{pmatrix}$$

We define $\mathcal{Z}$ as the set of all datasets $\mathcal{D}$ whose transactions are built on $\mathcal{I}$ and whose corresponding bipartite graph $G_{\mathcal{D}}$ has the same BJDM $\mathsf{J}_{G_{\mathcal{D}}}$. We justify this choice by first

[9] We always denote an edge of a bipartite graph corresponding to a dataset as (a, b) with $a \in \mathcal{D}$ and $b \in \mathcal{I}$, i.e., as an element of $\mathcal{D} \times \mathcal{I}$, to make it clear which endpoint is a transaction and which is an item.

showing that preserving the BJDM also preserves the two fundamental properties, and then that it preserves additional ones.

Fact 1 *For every* $1 \leq j \leq k_R$*, it holds*

$$|\{v \in R : \mathsf{deg}(v) = j\}| = \frac{1}{j} \sum_{i=1}^{k_L} \mathsf{J}_G[i, j], \tag{6}$$

i.e., the BJDM J_G *determines, for every* $1 \leq j \leq k_R$*, the number of vertices* $v \in R$ *of degree* $\mathsf{deg}(v) = j$.

Similarly, for every $1 \leq i \leq k_L$*, it holds*

$$|\{u \in L : \mathsf{deg}(u) = i\}| = \frac{1}{i} \sum_{j=1}^{k_R} \mathsf{J}_G[i, j], \tag{7}$$

i.e., the BJDM J_G *determines, for every* $1 \leq i \leq k_L$*, the number of vertices* $u \in L$ *with degree* $\mathsf{deg}(u) = i$.

Corollary 2 *For any dataset* $\mathcal{D}$*, the BJDM* $\mathsf{J}_{G_\mathcal{D}}$ *determines, for every* $1 \leq j \leq |\mathcal{I}|$*, the number of transactions in* $\mathcal{D}$ *with length* j*. Also, it determines, for every* $1 \leq i \leq |\mathcal{D}|$*, the number of items with support* i *in* $\mathcal{D}$.

Corollary 2 states that preserving the BJDM also preserves the two fundamental properties. We now show an additional property that is preserved, among others.

Let $\mathsf{z}(G_{\mathring{\mathcal{D}}})$ be the number of *simple paths of length three* in $G_{\mathring{\mathcal{D}}}$, which, since $G_{\mathring{\mathcal{D}}}$ is bipartite, is also known as the number of *caterpillars* of $G_{\mathring{\mathcal{D}}}$ [4]. Corollary 4 shows that preserving the BJDM of $G_{\mathring{\mathcal{D}}}$ preserves the number of caterpillars. The numbers of simple paths of length one and two are already preserved by preserving the two fundamental properties, thus preserving also the number of simple paths of length three is a natural step. Our desired result is a corollary of Lemma 3, which shows that $\mathsf{z}(G)$ can be expressed through the BJDM.

Lemma 3 *It holds*

$$\mathsf{z}(G) = \sum_{i=2}^{k_L} \sum_{j=2}^{k_R} \mathsf{J}_G[i, j](i-1)(j-1) \ .$$

Proof Each edge $(u, v) \in E$ is the middle edge of $(\mathsf{deg}(u) - 1)(\mathsf{deg}(v) - 1)$ caterpillars, so

$$\mathsf{z}(G) = \sum_{(u,v)\in E} (\mathsf{deg}(u) - 1)(\mathsf{deg}(v) - 1) \ . \tag{8}$$

From here, we can conclude that

$$\sum_{(u,v)\in E} (\mathsf{deg}(u) - 1)(\mathsf{deg}(v) - 1) = \sum_{i=2}^{k_L} \sum_{j=2}^{k_R} \mathsf{J}_G[i, j](i-1)(j-1)$$

because each edge $(u, v) \in E$ that connects a node $u \in L$ with degree $\mathsf{deg}(u) = i$ to a node $v \in R$ with degree $\mathsf{deg}(v) = j$ contributes $(i - 1)(j - 1)$ caterpillars to the summation in Eq. (8), and there are $\mathsf{J}_G[i, j]$ such edges. □

Corollary 4 *For any $\mathcal{D}$, the BJDM $\mathsf{J}_{G_{\mathcal{D}}}$ determines $\mathsf{z}(G_{\mathcal{D}})$.*

On the other hand, preserving the two fundamental properties and the number of caterpillars is not sufficient to preserve the BJDM: as we now show, it is easy to construct datasets that have the same transaction lengths, same item supports, and same number of caterpillars as an observed dataset $\mathring{\mathcal{D}}$, but whose BJDM is different than $\mathsf{J}_{G_{\mathring{\mathcal{D}}}}$. We show an example in Fig. 2. Both bipartite graphs in Fig. 2 have three connected components each, with a total of 27 left-hand side nodes (light-blue, striped nodes) and 8 right-hand side nodes (yellow, dotted nodes). It is easy to see that the two graphs have the same degree distributions, and the same number of caterpillars (48). In the upper graph, the leftmost component contains 36 caterpillars, while each of the other two components contains 6 caterpillars, for a total of 48 caterpillars. Similarly, in the lower graph, the leftmost component contains 36 caterpillars, and the other two 6 caterpillars each. The two graphs have, nevertheless, different BJDMs: in the upper graph there are edges connecting nodes with degree 4 to nodes with degree 5 (top left), but the lower graph has no such edge.

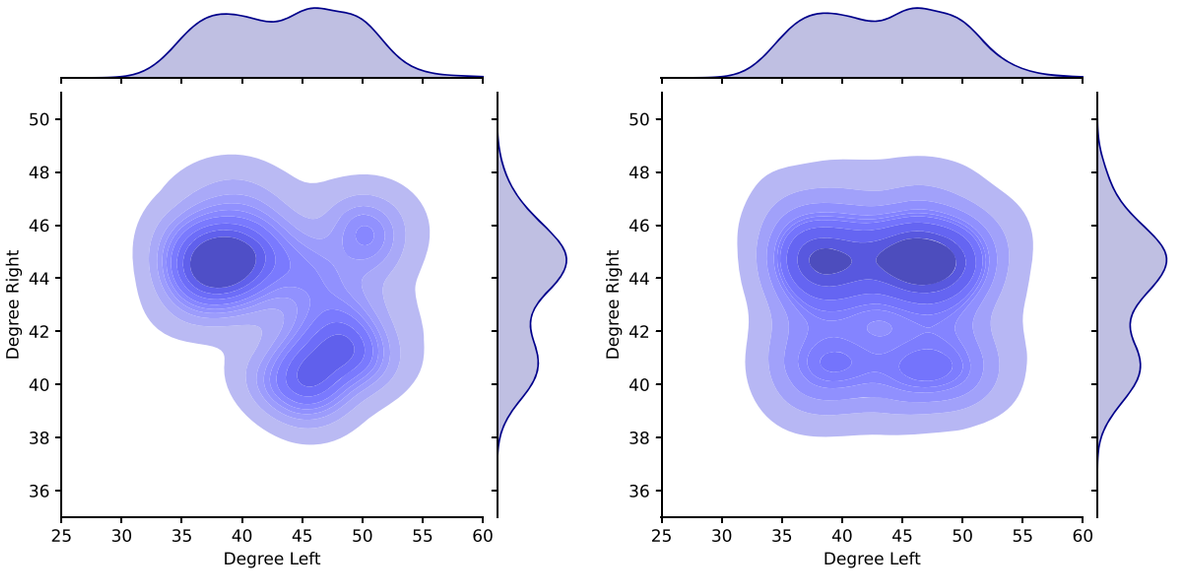
We considered preserving more "natural" characteristics than the BJDM, such as the support of each itemset of length two. However, doing so would lead to null sets $\mathcal{Z}$ that contain very few datasets in most cases, and are therefore not very informative about the data generation process, as they are likely overly constrained. Informally, the reason is that the biadjacency matrix $M_{\mathcal{D}}$ of the graph $G_{\mathcal{D}}$ corresponding to any dataset $\mathcal{D}$ in such a $\mathcal{Z}$ must satisfy $M_{\mathcal{D}}^{M_{\mathcal{D}}^\intercal} = M_{\mathcal{D}}^{M_{\mathring{\mathcal{D}}}^\intercal}$. Binary matrices A and B satisfying $AA^\intercal = BB^\intercal$ are known as *Gram mates* [29, 30]. [30, Corol. 1.1.1] shows an upper bound to the relative size of the set of Gram mates w.r.t. the set of all binary matrices, which decreases as the number of transactions in $\mathring{\mathcal{D}}$ and/or the number of items in $\mathcal{I}$ grow. While Kirkland [30] and Kim and Kirkland [29] construct infinite families of Gram mates, they observe that these families "possess a tremendous amount of structure" [30, Sect. 4], and it seems unlikely that such a structure would ever occur on matrices corresponding to real datasets, to the point that it is still an open question to determine whether a matrix A even admits *any* Gram mate, which would at least allow us to determine whether or not $|\mathcal{Z}| = 1$. On the other hand, if one can find at least one pair of Gram mates, [29, Sect. 5] give methods to build others (but possibly not *all*); thus, if the open question is settled in a constructive way, one may be able to sample from (a subset of) $\mathcal{Z}$, if so interested.

Finally, we give an intuition about the properties that ALICE preserves in addition to the fundamental ones. Preserving the BJDM of a bipartite graph means preserving the number of edges connecting two nodes with given degrees. This property implies, for instance, that the *assortativity* of the graph [39], i.e., the Pearson correlation coefficient of the vectors of degrees of nodes connected by an edge, is also maintained. Figure 3 shows an example of this property. Assume to have a dataset with an empirical joint degree distribution as in Fig. 3a. ALICE preserves this joint degree distribution exactly. Conversely, by preserving only the two fundamental properties, we only preserve the marginal distributions as in Fig. 3b. In this latter case, the joint distribution is simply the product of the marginals, i.e., the marginals are assumed independent.

5 Sampling from the null model

We now present ALICE- A and ALICE- B, two algorithms for sampling datasets from the null model $\Pi = (\mathcal{Z}, \pi)$.

These algorithms take the MCMC approach with MH (see Sect. 3.3). Their set of states is the set $\mathcal{M}$ of matrices defined as follows. Fix $M_{\mathring{\mathcal{D}}}$ to be any of the biadjacency matrices



(a) Joint distribution under Alice, which preserves the BJDM and maintains the degree assortativity of the dataset

(b) Joint degree distribution when preserving the two fundamental properties, where the left and right degree distributions are independent

Fig. 3 Example of two different joint degree distributions of bipartite graphs with the same marginal degree distributions

of a bipartite graph corresponding to the observed dataset $\mathring{\mathcal{D}}$. $\mathcal{M}$ contains all and only the matrices M of size $|\mathring{\mathcal{D}}| \times |\mathcal{I}|$ such that, when considering M as the biadjacency matrix of a bipartite graph G_M, it holds $\mathsf{J}_{G_M} = \mathsf{J}_{G_{\mathring{\mathcal{D}}}}$.

$\mathcal{M}$ may contain multiple matrices associated to the same dataset (see Sect. 4.1), and different datasets may have a different number of matrices in $\mathcal{M}$ associated to them. ALICE- A and ALICE- B take this fact into account to ensure that the sampling of datasets from $\mathcal{Z}$ is done according to π. For $M \in \mathcal{M}$, we use $\mathsf{dat}(M)$ to denote the unique dataset corresponding to M, and for a dataset $\mathcal{D} \in \mathcal{Z}$, we use $\mathsf{mat}(\mathcal{D})$ to denote the *set* of matrices in $\mathcal{M}$ corresponding to $\mathcal{D}$. [1, Lemma 3] give an expression for the size $\mathsf{c}(\mathcal{D}) \doteq |\mathsf{mat}(\mathcal{D})|$ of $\mathsf{mat}(\mathcal{D})$. The correctness of the two algorithms relies on it so we report it here.

Lemma 5 (1, Lemma 3) *For any dataset $\mathcal{D} \in \mathcal{Z}$, let $\{\ell_1, \ldots, \ell_{z_\mathcal{D}}\}$ be the set of the $z_\mathcal{D}$ distinct lengths of the transactions in $\mathcal{D}$. For each $1 \leq i \leq z_\mathcal{D}$, let T_i be the* bag *of transactions of length ℓ_i in $\mathcal{D}$. Let $\bar{T}_i = \{\tau_{i,1}, \ldots, \tau_{i,r_i}\}$ be the* set *of transactions of length ℓ_i in $\mathcal{D}$, i.e., without duplicates. For each $1 \leq j \leq r_i$, let $Q_{i,j} \doteq \{t' \in T_i : t' = \tau_{i,j}\}$ be the* bag *of transactions in T_i equal to $\tau_{i,j}$ (including $\tau_{i,j}$). Then, the number of matrices M in $\mathcal{M}$ such that $\mathsf{dat}(M) = \mathcal{D}$ is*

$$\mathsf{c}(\mathcal{D}) = \prod_{i=1}^{z_\mathcal{D}} \underbrace{\binom{|T_i|}{|Q_{i,1}|, \ldots, |Q_{i,r_i}|}}_{\textit{multinomial coefficient}} = \prod_{i=1}^{z_\mathcal{D}} \frac{|T_i|!}{\prod_{j=1}^{r_i} |Q_{i,j}|!} \,. \tag{9}$$

ALICE- A and ALICE- B take as inputs π and the observed dataset $\mathring{\mathcal{D}}$. It uses MH (see Sect. 3.3) to sample a matrix $M \in \mathcal{M}$ according to a distribution ϕ (defined below), and returns $\mathcal{D} = \mathsf{dat}(M) \in \mathcal{Z}$ distributed according to π. Both algorithms we present share the same set $\mathcal{M}$ of states, but they have different neighborhood structures (i.e., the graphs used by MH for the two algorithms have different sets of edges), different neighbor distributions ξ_M, $M \in \mathcal{M}$, and different neighbor sampling procedures.

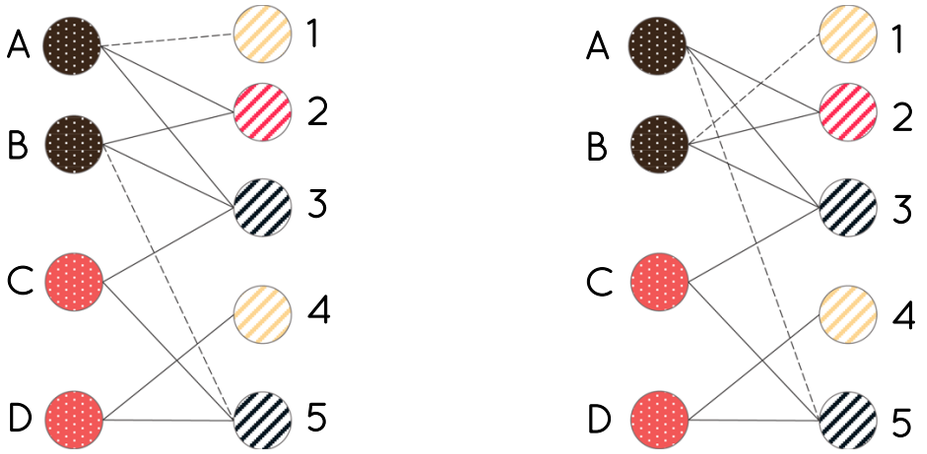


Fig. 4 The RSO denoted with dashed edges transforms the left graph into the right graph. Different patterns denote nodes on different sides of the graph, while different colors denote different degrees

5.1 ALICE-A: RSO-based algorithm

In our first algorithm, ALICE- A, the neighborhood structure over $\mathcal{M}$ is defined using *Restricted Swap Operations (RSOs)* [13, Sect. 2].

Definition 2 (*Restricted swap operation (RSO)*) Let M be the $|L| \times |R|$ biadjacency matrix of a bipartite graph $G = (L \cup R, E)$. Let $1 \leq a \neq b \leq |L|$ and $1 \leq c \neq d \leq |R|$ be the indices of two rows and columns of M, respectively, such that

$$M[a, c] = M[b, d] = 1 \wedge M[a, d] = M[b, c] = 0$$

and such that *at least one* of the following conditions holds

$$C_{ab} = \text{“}\sum_{j=1}^{|R|} M[a, j] = \sum_{j=1}^{|R|} M[b, j]\text{”}$$
$$C_{cd} = \text{“}\sum_{i=1}^{|L|} M[i, c] = \sum_{i=1}^{|L|} M[i, d]\text{”} \ .$$

The *Restricted Swap Operation (RSO)* $(a, c), (b, d) \rightarrow (a, d), (b, c)$ *on* M is the operation that obtains the matrix M' which is the same as M but $M'[a, c] = M[a, d]$, $M'[a, d] = M[a, c]$, $M'[b, c] = M[b, d]$, and $M'[b, d] = M[b, c]$.

Figure 4 (left) depicts a bipartite graph, where dotted nodes indicate left nodes, and striped nodes indicate right nodes. For ease of presentation, we use different colors to denote nodes with the same degree. A RSO in this graph is $(A, 1), (B, 5) \rightarrow (A, 5), (B, 1)$, because A and B satisfy condition C_{ab} and the edges $(A, 5)$ and $(B, 1)$ are not part of the graph. Figure 4 (right) shows the graph resulting from the application of the RSO. Dashed edges are edges involved in the RSO.

Any RSO on $M \in \mathcal{M}$ results in a matrix M' that belongs to $\mathcal{M}$ as well. In the graph $G = (\mathcal{M}, E)$ needed for MH, there is an edge from M to M' if there is a RSO from M to M'. Additionally, there are *self-loops* from any $M \in \mathcal{M}$ to itself. These self-loops do not correspond to RSOs, but they simplify the neighbor sampling procedure (described next).

There are zero or one RSOs between any pair of matrices in $\mathcal{M}$, but $\mathcal{M}$ is strongly connected by RSOs [13, Thm. 8].[10]

RSOs are just one of the many possible operations that make $\mathcal{Z}$ strongly connected. We discuss one such different operation in Sect. 5.2. Finding other operations to replace RSOs or to use in addition to RSOs is an interesting research direction.

We now discuss the second ingredient needed to use MH: the distribution ξ_M over the set of neighbors $\Gamma(M)$ of any $M \in \mathcal{M}$. At first, using a distribution ξ_M of the form

$$\xi_M(M') \doteq \begin{cases} \frac{2}{|\mathcal{I}|^2|\mathcal{D}|^2} & M' \in \Gamma(M) \setminus \{M\} \\ 1 - \frac{2(|\Gamma(M)|-1)}{|\mathcal{I}|^2|\mathcal{D}|^2} & M' = M \end{cases}$$

may seem an appealing option, because it could be realized by first drawing a 4-tuple (a, b, c, d) uniformly at random from $\mathcal{D} \times \mathcal{D} \times \mathcal{I} \times \mathcal{I}$, and then verifying whether $(a, c), (b, d) \to (a, d), (b, c)$ is a RSO: if it is, one would set M' to be the matrix resulting from applying the RSO to M, otherwise $M' = M$. The major issue with this approach is that, depending on M, the number of tuples that must be drawn before finding one that is a RSO may be very large, thus slowing down the process of moving on the graph. We briefly touch upon the convergence problem of this approach in Sect. 7. Conversely, more complex probability distributions that ensure drawing a neighbor different than M are quite easy to define, but come with the serious drawback that they need expensive computation and bookkeeping of quantities such as $|\Gamma(M)|$ and $|\Gamma(M')|$ for $M' \in \Gamma(M)$ (due to Eq. (5)), or the number of pairs of different rows or columns of the same lengths in M and $M' \in \Gamma(M)$. The process of sampling a neighbor would then be much more expensive, thus again slowing down the walk on the graph. We propose a distribution over $\Gamma(M)$ and a procedure to sample from it that strikes a balance between statistical and computational "efficiency": the probability of sampling M is smaller than in the naïve case described above, and sampling a neighbor is still quite efficient.

Let $M \in \mathcal{M}$ be the current state. For any $1 \le m \le |\mathcal{I}|$ (resp. $1 \le n \le |\mathcal{D}|$), let A_m be the set of row indices in M whose rows have sum m (resp. let B_n be set of column indices in M whose columns have sum n). To sample a neighbor M' of M, we start by flipping a fair coin. If the outcome is *heads*, we first draw a row sum $1 \le m \le |\mathcal{I}|$ with probability

$$\beta(m) = \binom{|A_m|}{2} \Big/ \sum_{j=1}^{|\mathcal{I}|} \binom{|A_j|}{2}, \tag{10}$$

and then we draw a pair (a, b) of *different* row indices in A_m uniformly at random between such pairs. If the row of index a and the row of index b in M are identical, then we set $M' = M$. Otherwise, consider the set $H_{a,b}$ of column index pairs (p, q) such that

$$M[a, p] = M[b, q] \wedge M[a, q] = M[b, p] \wedge M[a, p] \neq M[a, q] \ .$$

We draw a pair (c, d) from $H_{a,b}$ uniformly at random. Then, either $(a, c), (b, d) \to (a, d), (b, c)$ or $(a, d), (b, c) \to (a, c), (b, d)$ is a RSO by construction, and we set M' to be the matrix obtained by performing this RSO on M. If the outcome of the coin flip is

[10] The proof of [13, Thm. 8] must be adapted, in a straightforward way, to account for the fact that $\mathcal{M}$ contains biadjacency matrices of bipartite graphs.

tails, we first draw a column sum $1 \leq n \leq |\mathcal{D}|$ with probability

$$\gamma(n) = \binom{|B_n|}{2} \Bigg/ \sum_{j=1}^{|\mathcal{D}|} \binom{|B_j|}{2}, \tag{11}$$

and then we draw a pair (c, d) of different column indices in B_n uniformly at random between such pairs. If the column of index c and the column of index d in M are identical, then we set $M' = M$. Otherwise, consider the set $K_{c,d}$ of row index pairs (p, q) such that

$$M[p, c] = M[q, d] \wedge M[p, d] = M[q, c] \wedge M[p, c] \neq M[p, d] \ .$$

We draw a pair (a, b) from $K_{c,d}$ uniformly at random. Then, either $(a, c), (b, d) \rightarrow (a, d), (b, c)$ or is also a RSO by construction, and we set M' to be $(a, d), (b, c) \rightarrow (a, c), (b, d)$ is a RSO by construction, and we set M' to be the matrix obtained by performing this RSO on M.

This procedure induces a probability distribution ξ_M over $\Gamma(M)$. Let us analyze $\xi_M(M')$ for $M' \neq M$. W.l.o.g., let $(a, c), (b, d) \rightarrow (a, d), (b, c)$ be the sampled RSO, and let M' be the neighbor of M obtained by performing such RSO on M. Recall that the sampled RSO is the only RSO from M to M'. Consider the following events:

$$E_{\text{row}} \doteq \text{“rows } a \text{ and } b \text{ of } M \text{ have the same row sum } m''\text{;}$$
$$E_{\text{col}} \doteq \text{“columns } c \text{ and } d \text{ of } M \text{ have the same column sum } n''.$$

There are three possible cases for the probability $\xi_M(M')$ of sampling M':

- if only E_{row} holds, then

$$\xi_M(M') = \frac{1}{2} \frac{1}{\sum_{i=1}^{|\mathcal{I}|} \binom{|R_i|}{2}} \frac{1}{|H_{a,b}|}; \tag{12}$$

- if only E_{col} holds, then

$$\xi_M(M') = \frac{1}{2} \frac{1}{\sum_{j=1}^{|\mathcal{D}|} \binom{|C_j|}{2}} \frac{1}{|K_{a,b}|}; \tag{13}$$

- if both E_{row} and E_{col} hold, then M' (i.e., the RSO) may be sampled regardless of the outcome of the coin flip. Thus, $\xi_M(M')$ is the sum of r.h.s.'s of Eq. (12) and Eq. (13).

We do not need to analyze $\xi_M(M)$ because if M is drawn as the “neighbor”, then MH will definitively select M as the next state; thus, we do not need to explicitly compute its probability.

It holds that $\xi_M(M') = \xi_{M'}(M)$, which greatly simplifies the use of MH: from Eq. (5), we see that, thanks to the construction of the graph and the definition of the neighbor sampling distribution, we really only need the distribution ϕ over $\mathcal{M}$. We define it as

$$\phi(M) = \frac{\pi(\mathsf{dat}(M))}{\mathsf{c}(\mathsf{dat}(M))}, \tag{14}$$

where $\mathsf{c}(\mathsf{dat}(M))$ is from Eq. (9). The following lemma shows that ALICE- A samples a dataset $\mathcal{D}$ from $\mathcal{Z}$ according to π, i.e., it samples from the null model.

Lemma 6 *Let $\mathcal{D} \in \mathcal{Z}$.* ALICE- A *outputs $\mathcal{D}$ with probability $\pi(\mathcal{D})$.*

Proof Let $M \in \mathcal{M}$. From the correctness of MH we have that ALICE- A samples M according to ϕ from Eq.(14). The thesis then follows from noticing that $\mathcal{D}$ is returned in output whenever ALICE- A samples one of the $\mathsf{c}(\mathcal{D})$ matrices in $\mathcal{M}$ corresponding to $\mathcal{D}$. □

Algorithm 1 illustrates the main steps performed by ALICE- A to sample a dataset in $\mathcal{Z}$. The algorithm receives in input a matrix $M \in \mathcal{M}$ and a number of swaps s sufficiently large for convergence. Previous works estimated that a number of steps in order of the number of 1s in M is sufficient. We will discuss this aspect in Sect. 7.

Algorithm 1 ALICE

Require: Matrix $M \in \mathcal{M}$, Number of Swaps s
Ensure: Dataset $\mathcal{D}$ sampled from $\mathcal{Z}$ with probability $\pi(\mathcal{D})$
1: $\mathsf{c}(\mathsf{dat}(M)) \leftarrow$ Equation (9)
2: $i \leftarrow 0$
3: **while** $i < s$ **do**
4: $\quad i \leftarrow i + 1$
5: $\quad \mathsf{out} \leftarrow$ flip a fair coin
6: $\quad$ **if** out is *heads* **then**
7: $\quad\quad a, b \leftarrow$ different row indices drawn u.a.r. such that C_{ab} holds
8: $\quad\quad c, d \leftarrow$ pair drawn u.a.r. from H_{ab}
9: $\quad$ **else**
10: $\quad\quad c, d \leftarrow$ different column indices drawn u.a.r. such that C_{cd} holds
11: $\quad\quad a, b \leftarrow$ pair drawn u.a.r. from K_{cd}
12: $\quad$ **end if**
13: $\quad M' \leftarrow$ perform $(a, c), (b, d) \rightarrow (a, d), (b, c)$ on M
14: $\quad \mathsf{c}(\mathsf{dat}(M')) \leftarrow$ Equation (9)
15: $\quad p \leftarrow$ random real number in $[0, 1]$
16: $\quad a \leftarrow \min\left(1, \mathsf{c}(\mathcal{D})/\mathsf{c}(\mathcal{D}')\right)$
17: $\quad$ **if** $p \leq a$ **then** $M \leftarrow M'$
18: $\quad$ **end if**
19: **end while**
20: **return** $\mathsf{dat}(M)$

5.2 ALICE-B: Adapting curveball

We now introduce a second algorithm, ALICE- B, that can essentially perform multiple RSOs at each step of the Markov chain, thus leading to a faster mixing of the chain, i.e., to fewer steps needed to sample a dataset from Π. Our approach adapts the CURVEBALL algorithm [62], which samples a matrix from the space of binary matrices with fixed row and column sums, to use RSOs. ALICE- B is also an MCMC algorithm that uses MH. The vertex set of the graph $G = (\mathcal{M}, E)$ is still the set $\mathcal{M}$ previously defined, but ALICE- B uses a different set of edges than ALICE- A: there is an edge $(M, M') \in E$ from a matrix $M \in \mathcal{M}$ to $M' \in \mathcal{M}$ iff $M' = M$ or there is a *Restricted Binomial Swap Operation (RBSO)* on M that results in M'. RBSOs are defined as follows.

Definition 3 (*Restricted binomial swap operation* (*RBSO*)) Given a matrix $M \in \mathcal{M}$, let a and b be the indices of two *distinct and different* rows of M with the same row sum. Let $Z_a(M, b)$ be the set of column indices q such that $M[a, q] = 1$ and $M[b, q] = 0$, and define $Z_b(M, a)$ similarly (it holds $Z_a(M, b) \cap Z_b(M, a) = \emptyset$ and $|Z_a(M, b)| = |Z_b(M, a)|$). Let

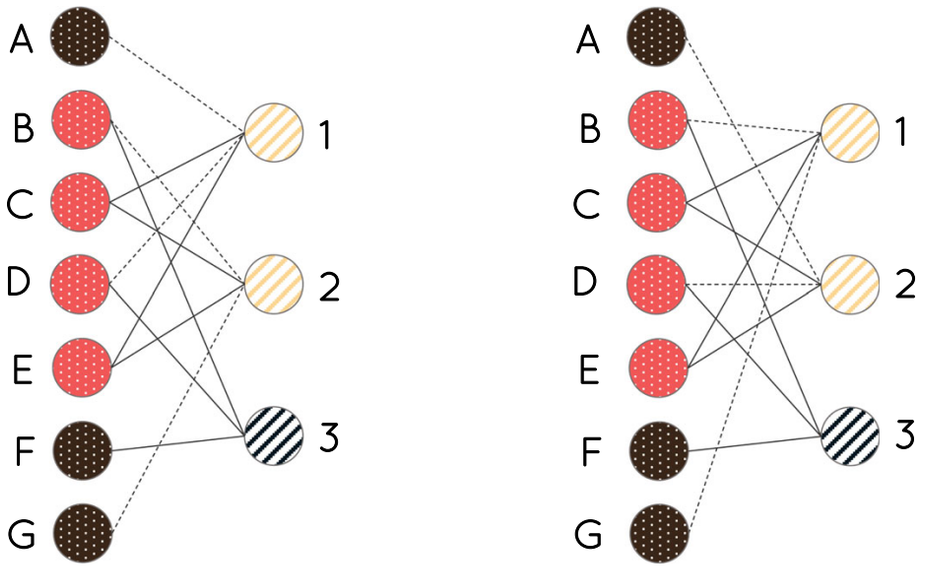


Fig. 5 The RBSO denoted with dashed edges transforms the left graph into the right graph. Different patterns denote nodes on different sides of the graph, while different colors denote different degrees

U be any subset of $Z_a(M, b) \cup Z_b(M, a)$ of size $|Z_a(M, b)|$. The *row Restricted Binomial Swap Operation (rRBSO)* (a, b, U) *on* M is the operation that obtains a matrix M' such that $M'[i, j] = M[i, j]$ except for $i \in \{a, b\}$, and such that the rows of index a and b of M' are

$$M'[a, q] \doteq \begin{cases} M[a, q] & q \notin Z_a(M, b) \cup Z_b(M, a) \\ 1 & q \in U \\ 0 & q \in (Z_a(M, b) \cup Z_b(M, a)) \setminus U \end{cases}$$

and

$$M'[b, q] \doteq \begin{cases} M[b, q] & q \notin Z_a(M, b) \cup Z_b(M, a) \\ 0 & q \in U \\ 1 & q \in (Z_a(M, b) \cup Z_b(M, a)) \setminus U \end{cases}$$

A corresponding definition for a *column RBSO (cRBSO)* can be given for a and b being the indices of two distinct and different columns with the same column sum.

We use "RBSO" to refer to either a rRBSO or a cRBSO, and the set of RBSOs is composed by all rRBSOs and cRBSOs.

Figure 5 (left) depicts a bipartite graph using the same style used in Fig. 4. Let $a = 1$ and $b = 2$, which are two right nodes with the same degree but different sets of neighbors. Then, $Z_a(M, b) = \{A, D\}$ and $Z_b(M, a) = \{B, G\}$. For $U = \{B, G\}$, the RBSO (a, b, U) generates the graph in Fig. 5 (right). Dashed edges are edges involved in the RBSO.

Any RBSO on a matrix M preserves J_M, and any RBSO can be seen as a sequence of RSOs. For any RSO $(a, c), (b, d) \rightarrow (a, d), (b, c)$ on M there is an equivalent RBSO $(a, b, (Z_a(M, b) \setminus \{c\}) \cup \{d\})$ from M, and thus the graph $G = (\mathcal{M}, E)$ is also strongly connected, as it has all the edges which are created by RSOs, plus potentially others.

Fact 7 *Let* (a, b, U) *be a cRBSO (resp. rRBSO) from* M *to* $M' \in \Gamma(M)$ *with* $M' \neq M$. *Then* $(a, b, Z_a(M, b))$ *is a cRBSO (resp. rRBSO) from* M' *to* M.

Lemma 8 *There are either one or two RBSOs from* $M \in \mathcal{M}$ *to* $M' \in \Gamma(M)$ *with* $M' \neq M$. *When there are* two *RBSOs, one is a cRBSO and the other is a rRBSO.*

Proof Let us start from the second part of the thesis. If $(a, b, \{c\})$ is a cRBSO (resp. rRBSO) from M to M', then

$$(c, (Z_a(M, b) \cup Z_b(M, a)) \setminus \{c\}, \{a\})$$

is a rRBSO (resp. cRBSO) from M to M'.

The fact that there can only be one or two RBSOs is a consequence of Fact 7. □

In order for two RBSOs from M to M' to exist, it is necessary that $|Z_a(M, b)| = |Z_b(M, a)| = 1$, the columns at indices a and b have the same sum, and the rows at indices c and $(Z_a(M, b) \cup Z_b(M, a))\setminus\{c\}$ have the same sum.

Corollary 9 *For any two M and M', there is the same number of RBSOs from M to M' as from M' to M.*

Let us now give the procedure to sample a neighbor $M' \in \Gamma(M)$ of M. The procedure is similar to the one for ALICE- A. First, we flip a fair coin. If the outcome is *heads*, we draw a row sum $1 \leq m \leq |\mathcal{I}|$ with probability as per Eq. (10), and then we draw a pair (a, b) of different row indices in R_m uniformly at random between such pairs. If the row of index a and the row of index b in M are identical, then we set $M' = M$. Otherwise, we compute the set $Z_a(M, b) \cup Z_b(M, a)$ defined in Definition 3 and the cardinality $|Z_a(M, b)|$ with a linear scan of the rows a and b. By using reservoir sampling [63], we obtain U through a linear scan of $Z_a(M, b) \cup Z_b(M, a)$. If the outcome of the coin flip is *tails*, we first draw a column sum $1 \leq n \leq |\mathcal{D}|$ with probability as per Eq. (11), then we draw a pair (a, b) of different column indices in C_n uniformly at random between such pairs. We then proceed in a fashion similar as for the row case. The purpose of flipping the coin at the start is to ensure that we can sample both rRBSOs (when the outcome is heads), and cRBSOs (otherwise).

The probability $\xi_M(M')$ of sampling a RBSO (a, b, U) on M that results in M', is not uniform. Rather than giving the expression for it, we use the fact that, in order to use MH, we really only need the distribution ϕ over $\mathcal{M}$, and the *ratio* ${}^{\xi_{M'}(M)}/_{\xi_M(M')}$ (see Eq. (5)), and we now show that $\xi_M(M') = \xi_{M'}(M)$, i.e., the ratio is always 1.

Lemma 10 *Let $M \in \mathcal{M}$ and $M' \in \Gamma(M)$. Then $\xi_M(M') = \xi_{M'}(M)$.*

Proof We assume that $M' \neq M$, otherwise the thesis is obviously true. For ease of presentation, we focus on the case where there is only a cRBSO (a, b, U) from M to M'. The analysis for the case when there is only a rRBSO follows the same steps, and the one for the case when there is both a cRBSO and a rRBSO follows by combining the two cases.

From Fact 7, the cRBSO $(a, b, Z_a(M, b))$ goes from M' to M. The probability that the coin flip is tails is the same no matter whether the current state is M or if it is M, as is the probability, given that the outcome was tails, of sampling the columns indices a and b. By definition, it holds that $|U| = |Z_a(M, b)|$, and it is easy to see that $Z_a(M, b) \cup Z_b(M, a) = Z_a(M', b) \cup Z_b(M', a)$, thus the probability of sampling U when the current state is M and we have sampled a and b, and the probability of sampling $Z_a(M, b)$ when the current state is M' and we have sampled a and b are the same. Thus, the probability of sampling (a, b, U) when the current state is M is the same as the probability of sampling $(a, b, Z_a(M, b))$ when the current state is M', and the proof is complete. □

Thus, to use MH, we really only need the distribution ϕ over $\mathcal{M}$. As in Sect. 5.1, in order to sample a dataset $D \in \mathcal{Z}$ according to π, we want to sample a matrix $M \in \mathcal{M}$ with the probability given in Eq. (14). We thus have all the ingredients to use MH, and our description of ALICE- B is complete. Note that ALICE- B follows the same structure presented

in Algorithm 1 but samples a rRBSO (a, b, U) at line 8:

$U \subset Z_a(M, b) \cup Z_b(M, a)$ s.t. $\lvert U\rvert = \lvert Z_a(M, b)\rvert$ obtained via reservoir sampling

and a cRBSO (c, d, U) at line 11:

$U \subset Z_c(M, d) \cup Z_d(M, c)$ s.t. $\lvert U\rvert = \lvert Z_c(M, d)\rvert$ obtained via reservoir sampling

6 Sequence datasets

Previous work studied null models for testing the statistical significance of results obtained from other kinds of datasets, such as sequence datasets [27, 35, 47, 60]. We now define a new null model for sequence datasets to also preserve the BJDM, and we introduce a new algorithm ALICE- S to sample from this null model.

6.1 Preliminaries on sequence datasets and multi-graphs

Let us start with a brief description of sequence datasets and related concepts. A *sequence* is a finite *ordered list* (or a *vector*) of not-necessarily distinct itemsets, i.e., $S = \langle A_1, \ldots, A_\ell \rangle$ for some $\ell \geq 1$, with $A_i \subseteq \mathcal{I}$, $1 \leq i \leq \ell$. Itemsets A_i *participate* in S, and we denote this fact with $A_i \in S$, $1 \leq i \leq \ell$. The *length* $|S|$ of a sequence is the number of itemsets participating in it. A *sequence dataset* $\mathcal{D}$ is a finite bag of sequences, which, as elements of $\mathcal{D}$, are known as *seq-transactions*. The *support* $\sigma_{\mathcal{D}}(A)$ of an itemset A in $\mathcal{D}$ is the number of seq-transactions of $\mathcal{D}$ in which A participates. The *multi-support* $\rho_{\mathcal{D}}(A)$ of A in $\mathcal{D}$ is the number of times that A participates *in total* in the seq-transactions of $\mathcal{D}$. For example, in the dataset $\mathcal{D} = \{\langle A, B\rangle, \langle A, C, A\rangle, \langle B, C\rangle\}$, it holds that $\sigma_{\mathcal{D}}(A) = 2$ and $\rho_{\mathcal{D}}(A) = 3$.

A sequence dataset $\mathcal{D}$ can be represented as a bipartite *multi*-graph $G_{\mathcal{D}} = (L \cup R, E)$, where L are the seq-transactions of $\mathcal{D}$, and R is the *set* of all and only the itemsets with support at least 1 in $\mathcal{D}$, i.e., participating in at least one seq-transaction of $\mathcal{D}$. Each vertex $v \in L$ has degree[11] equal to the length of the corresponding seq-transaction S_v of $\mathcal{D}$, i.e., $\mathsf{deg}(v) = |S_v|$. Each vertex $v \in L$ has $\mathsf{deg}(v)$ *ports*, which can be thought as the "locations" where the edges "connect" to v. The ports are arbitrarily labeled from 1 to $\mathsf{deg}(v)$. This labeling is needed to define the edge *multi*-set E as follows: there is an edge between $v \in L$ and $w \in R$ using port k of v iff the itemset B_w corresponding to the vertex w appears in position k of S_v, i.e., iff $S_v = \langle A_1, \ldots, A_{k-1}, B_w, A_{k+1}, \ldots, A_{|S_v|}\rangle$. We denote this edge as (v, k, w), thus E can also be thought as a set of such tuples. To the best of our knowledge, the one we just gave is the first description of sequence datasets as bipartite multi-graphs, which is somewhat surprising because representing transactional datasets as bipartite graphs has been a standard practice for a long time.

The definition of BJDM from Definition 1 is also valid for multi-graphs. Figure 6 shows

[11] In multi-graphs, the degree of a vertex v is still the number of edges incident to it, so each edge is counted, even if multiple edges connect v to the same vertex.

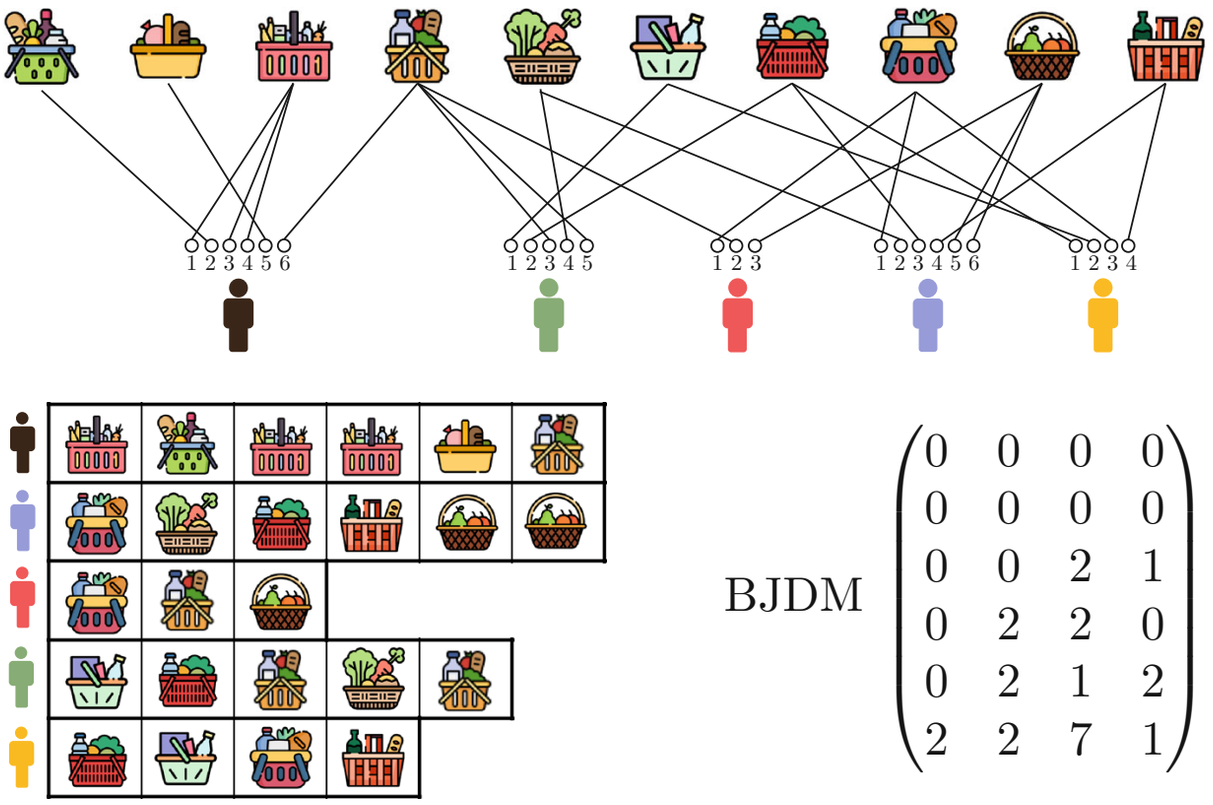


Fig. 6 Example of sequence dataset (lower left), corresponding multi-graph (top), and BJDM of the multi-graph (lower right)

an example of a sequence dataset (lower left), the corresponding multi-graph (top), and its BJDM (lower right).

6.2 BJDM-preserving null model for sequence datasets

Tonon and Vandin [60] introduce a null model $\Pi = (\mathcal{Z}, \pi)$ for sequence datasets that can be seen as an adaptation of Gionis et al. (2007)'s null model for transactional datasets. It preserves the following two properties of an observed dataset $\mathring{\mathcal{D}}$:

- the distribution of the seq-transaction lengths, i.e., for any seq-transaction length $\ell \in [1, \max_{S \in \mathring{\mathcal{D}}} |S|]$, any $\mathcal{D} \in \mathcal{Z}$ contains the same number of transactions of length ℓ as $\mathring{\mathcal{D}}$; and
- the multi-support of the itemsets participating in the seq-transactions of $\mathring{\mathcal{D}}$, i.e., for any $A \subseteq \mathcal{I}$ and $\mathcal{D} \in \mathcal{Z}$, $\rho_{\mathring{\mathcal{D}}}(A) = \rho_{\mathcal{D}}(A)$.

It should be evident how these two properties can be mapped to the two fundamental properties defined in Sect. 3.2 for transactional datasets, with the difference that itemsets participating in seq-transactions play the role that was of items in transactional datasets. Tonon and Vandin [60] gave a MCMC algorithm to sample from this null model, while Jenkins et al. (2022) gave an exact sampling algorithm.

The null model we define for sequence datasets preserves the BJDM of the multi-graph corresponding to the observed dataset. The following property can be derived in a way similar to that from Corollary 2 and confirms that preserving the BJDM also preserves the two above properties.

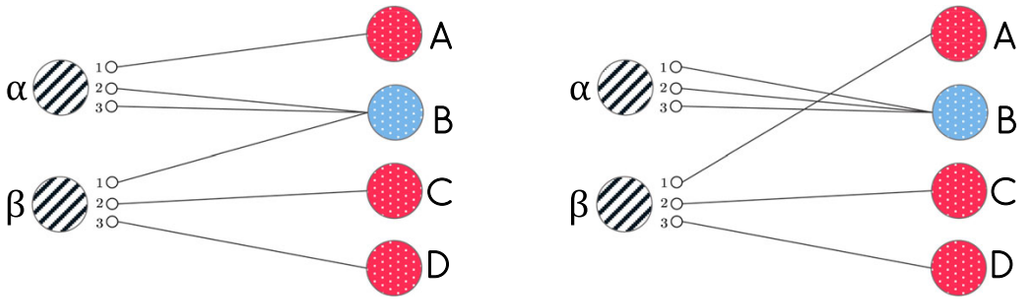


Fig. 7 Two bipartite multi-graphs with the same BJDM but different numbers of paths of length three. Different patterns denote nodes on different sides of the multi-graph, while different colors denote different degrees

Corollary 11 *For any sequence dataset $\mathcal{D}$, the BJDM $\mathsf{J}_{G_\mathcal{D}}$ determines, for every $1 \leq j \leq \max_{S \in \mathcal{D}} |S|$, the number of seq-transactions in $\mathcal{D}$ with length j. Also, it determines, for every $1 \leq i \leq |\mathcal{D}|$, the number of itemsets with multi-support i in $\mathcal{D}$.*

On the other hand, it is not true that preserving the BJDM also preserves the number of caterpillars on multi-graphs, i.e., there is no equivalent of Lemma 3 and Corollary 4. The reason is that the BJDM does not encode information that allows the distinction between simple and multiple edges, i.e., the fact that a vertex with degree x may have any number of neighbors between 1 and x. It is also easy to come up with examples showing that it is not true that preserving the BJDM on multi-graphs preserves the number of *not-necessarily simple* paths of length three composed of three distinct edges. For instance, the multi-graph in Fig. 7 (left) includes the following 10 paths of length three: $(\beta, 3, D) - (\beta, 1, B) - (\alpha, 3, B)$, $(\beta, 3, D) - (\beta, 1, B) - (\alpha, 2, B)$, $(\beta, 2, C) - (\beta, 1, B) - (\alpha, 3, B)$, $(\beta, 2, C) - (\beta, 1, B) - (\alpha, 2, B)$, $(\beta, 1, B) - (\alpha, 3, B) - (\alpha, 2, B)$, $(\beta, 1, B) - (\alpha, 2, B) - (\alpha, 3, B)$, $(\beta, 1, B) - (\alpha, 3, B) - (\alpha, 1, A)$, $(\beta, 1, B) - (\alpha, 2, B) - (\alpha, 1, A)$, $(\alpha, 3, B) - (\alpha, 2, B) - (\alpha, 1, A)$, and $(\alpha, 2, B) - (\alpha, 3, B) - (\alpha, 1, A)$. The multi-graph on the right, which can be obtained by applying the mRSO $(\alpha, 1, A), (\beta, 1, B) \rightarrow (\alpha, 1, B), (\beta, 1, A)$ has the same BJDM but only six paths of length three: $(\alpha, 1, B) - (\alpha, 2, B) - (\alpha, 3, B)$, $(\alpha, 1, B) - (\alpha, 3, B) - (\alpha, 2, B)$, $(\alpha, 2, B) - (\alpha, 1, B) - (\alpha, 3, B)$, $(\alpha, 2, B) - (\alpha, 3, B) - (\alpha, 1, B)$, $(\alpha, 3, B) - (\alpha, 2, B) - (\alpha, 1, B)$, and $(\alpha, 3, B) - (\alpha, 1, B) - (\alpha, 2, B)$.

Nevertheless, since the multi-graph corresponding to a sequence dataset may actually be a simple graph, preserving the BJDM preserves more structure of the observed dataset than just the two fundamental properties, as we discussed for the counterexample from Fig. 2.

6.3 ALICE-S: ALICE for sequence datasets

We now discuss ALICE- S, our algorithm for sampling from the BJDM-preserving null model for sequence dataset, which was defined in the previous section. Like the other members of the ALICE family, ALICE- S also takes the MCMC approach with MH. Its set of states though, is no longer the set $\mathcal{M}$ of biadjacency matrices, but a set $\mathcal{G}$ of bipartite multi-graphs defined as follows. Given the observed sequence dataset $\mathring{\mathcal{D}}$, let $G_{\mathring{\mathcal{D}}} = (L \cup R, E)$ be the multi-graph corresponding to it. $\mathcal{G}$ contains all and only the bipartite multi-graphs with node sets L and R, and with the same BJDM as $G_{\mathring{\mathcal{D}}}$. We remark that $\mathcal{G}$ therefore includes also bipartite multi-graphs that are isomorphic to each other but differ for the ports to which the edges are connected, as such graphs represent different sequence datasets where the order of the itemsets in (some of) the sequences is shuffled.

The reason for not using the set $\mathcal{M}$ of biadjacency matrices as the state space of ALICE- S is that a biadjacency matrix does not capture the entirety of the structure of a multi-graph

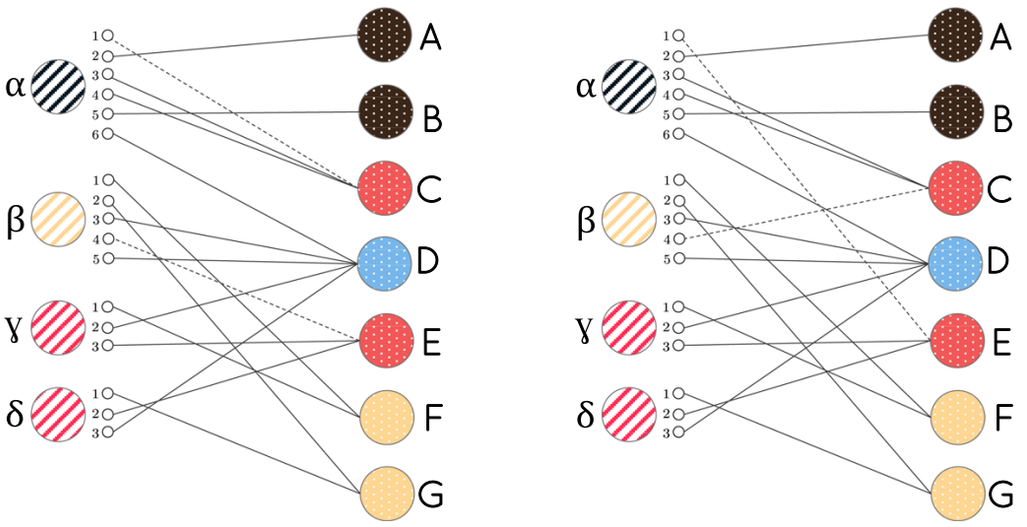


Fig. 8 Example of an mRSO. Dotted edges are edges involved in the mRSO, different patterns denote nodes on different sides of the graph, and different colors denote different degrees

corresponding to a sequence dataset, as it does not encode the information about the ports. It is important to understand that ALICE- A could have been easily presented in Sect. 5.1 with a state space composed of graphs, rather than biadjacency matrices. We chose not to do that because the presentation of ALICE- B greatly benefits from using matrices, although even in this case we could have used graphs, given that in the simple graph case, there is a bijection between bipartite graphs and biadjacency matrices. The flow and the notation in the following presentation of ALICE- S are similar to the one for ALICE- A, to highlight the many similarities between the two algorithms, but there are also many crucial differences.

We now define the concept of *multi-graph Restricted Swap Operation (mRSO)* as an operation that is applied to a multi-graph G to obtain another multi-graph G'.

Definition 4 (*Multi-graph restricted swap operation* (*mRSO*)) Let $G = (L \cup R, E)$ be a multi-graph, a and b be two non-necessarily distinct vertices in L, and c and d be two distinct vertices in R, such that there exist a port x of a and a port y of b such that

$$\{(a, x, c), (b, y, d)\} \subseteq E \wedge (\mathsf{deg}(a) = \mathsf{deg}(b) \vee \mathsf{deg}(c) = \mathsf{deg}(d)) \ .$$

The mRSO $(a, x, c), (b, y, d) \rightarrow (a, x, d), (b, y, c)$ is an operation that transforms G into the multi-graph $G' = (L \cup R, E')$ such that $E' = (E \backslash \{(a, x, c), (b, y, d)\}) \cup \{(a, x, d), (b, y, c)\}$.

It is easy to see that the multi-graph G' obtained by applying an mRSO to G is such that $\mathsf{J}_{G'} = \mathsf{J}_G$. There are zero or one mRSO between any two multi-graphs in $\mathcal{G}$. As an example, the mRSO $(\alpha, 1, C), (\beta, 4, E) \rightarrow (\alpha, 1, E), (\beta, 4, C)$ transforms the graph in Fig. 8 (left) to the graph on the right of such figure. Here, patterns denote the side of nodes on the graph, and colors denote different degrees. Dotted edges are the ones involved in the mRSO.

The neighborhood structure of the state space $\mathcal{G}$ is such that there is an edge from a multi-graph G to a multi-graph G' iff there is an mRSO transforming G into G'. In addition to these edges, there is a self-loop from each state to itself. This structure results in a strongly connected space, as can be seen by straightforwardly adapting [13, Thm. 8] in a way similar to what was done also for the bipartite simple graph case discussed in Sect. 5.1.

We now move to defining the neighbor sampling distribution ξ_G that is used to propose the next state $G' \in \Gamma(G)$ when the chain is at state $G \in \mathcal{G}$. As in Sect. 5.1, we first describe how to sample a neighbor of G and then analyze the resulting distribution.

For any $1 \leq m \leq |R|$ (resp. $1 \leq n \leq |\mathring{\mathcal{D}}|$), let A_m (resp. B_n) be the subset of L (resp. of R) containing all and only the vertices with degree m in G (but really, in any $G' \in \mathcal{G}$). The first operation to sample a neighbor of G is flipping a fair coin. If the outcome is *heads*, then we sample a degree m proportional to the number of pairs of *not-necessarily distinct* vertices in L with degree m, i.e., we draw $1 \leq m \leq |R|$ with probability

$$\beta(m) = \binom{|A_m| + 1}{2} \Bigg/ \sum_{j=1}^{|R|} \binom{|A_j| + 1}{2}$$

and then we draw two vertices a and b by sampling uniformly at random, with replacement, from A_m. By sampling with replacement, we ensure that a and b may be the same vertex. Consider now the set

$$H_{a,b} \doteq \{((a, x, f), (b, y, g)) : (a, x, f) \in E \wedge (b, y, g) \in E \wedge f \neq g\}$$

of pairs of edges one incident to a and one incident to b and with different endpoints in R, and sample a pair $((a, x, c), (b, y, d))$ uniformly at random from this set.

If the outcome of the fair coin flip is *tails*, we first sample a degree $1 \leq n \leq |L|$ proportional to the number of pairs of *distinct* vertices in R with degree n, i.e., we draw $1 \leq n \leq |L|$ with probability

$$\gamma(n) = \binom{|B_n|}{2} \Bigg/ \sum_{j=1}^{|\mathring{\mathcal{D}}|} \binom{|B_j|}{2},$$

and then we sample two *distinct* vertices c and d from B_n uniformly at random without replacement. Let now (a, x, c) (resp. (b, y, d)) be an edge sampled uniformly at random from those incident to c (resp. to d).

The mRSO $(a, x, c), (b, y, d) \rightarrow (a, x, d), (b, y, c)$, when applied to G, gives the neighbor G' which is the proposed next state for the Markov chain.

We now analyze the distribution ξ_G over $\Gamma(G)$ induced by this procedure. Let $(a, x, c), (b, y, d) \rightarrow (a, x, d), (b, y, c)$ be the sampled mRSO, and let $G' \in \Gamma(G)$ be the multi-graph obtained by applying this mRSO to G. It must be $G' \neq G$. Recall that this mRSO is the only one leading from G to G'. Consider the following events:

$$E_\ell \doteq \text{“}\mathsf{deg}(a) = \mathsf{deg}(b) = m\text{”};$$
$$E_\mathrm{r} \doteq \text{“}\mathsf{deg}(c) = \mathsf{deg}(d) = n\text{”} .$$

There are three possible cases for $\xi_G(G')$:

- If only E_ℓ holds, then

$$\xi_G(G') = \frac{1}{2} \frac{1}{\sum_{j=1}^{|R|} \binom{|A_j|+1}{2}} \frac{1}{H_{a,b}} . \tag{15}$$

- If only E_r holds, then

$$\xi_G(G') = \frac{1}{2} \frac{1}{\sum_{j=1}^{|\mathring{\mathcal{D}}|} \binom{|B_j|}{2}} \frac{1}{n^2} . \tag{16}$$

- If both E_ℓ and E_r hold, then $\xi_G(G')$ is the sum of the r.h.s.'s of Eqs. (15) and (16).

It is easy to see that $\xi_G(G') = \xi_{G'}(G)$, which, like for ALICE- A, greatly simplifies the use of MH. As in that case, we define ϕ over $\mathcal{G}$ as

$$\phi(G) \doteq \frac{\pi(\mathsf{dat}(G))}{\mathsf{c}(\mathsf{dat}(G))},$$

where $\mathsf{c}(\mathsf{dat}(G))$ is still as in Eq. (9) because the same result also holds for sequence datasets under the null model we are considering [1, Lemma 4]. We can then conclude on the correctness as follows, with the proof that is the same as that of Lemma 6.

Lemma 12 *Let $\mathcal{D} \in \mathcal{Z}$.* ALICE- S *outputs $\mathcal{D}$ with probability $\pi(\mathcal{D})$.*

Algorithm 2 reports the operations performed by ALICE- S to sample a sequence dataset in $\mathcal{Z}$. The algorithm receives in input the bipartite multi-graph $G \in \mathcal{G}$ corresponding to the observed sequence dataset $\mathring{\mathcal{D}}$ and a number of swaps s sufficiently large for convergence.

Algorithm 2 ALICE- S

Require: Multi-Graph $G \in \mathcal{G}$, Number of Swaps s
Ensure: Sequence Dataset $\mathcal{D}$ sampled from $\mathcal{Z}$ with probability $\pi(\mathcal{D})$
$\mathsf{c}(\mathsf{dat}(G)) \leftarrow$ Equation (9)
$i \leftarrow 0$
while $i < s$ **do**
  $i \leftarrow i + 1$
  out $\leftarrow$ flip a fair coin
  if out is *heads* **then**
    $a, b \leftarrow$ vertices in L drawn u.a.r. such that $\mathsf{deg}(a) = \mathsf{deg}(b)$
    $(a, x, c), (b, y, d) \leftarrow$ pair drawn u.a.r. from H_{ab}
  else
    $c, d \leftarrow$ different vertices in R drawn u.a.r. such that $\mathsf{deg}(c) = \mathsf{deg}(d)$
    $(a, x, c), (b, y, d) \leftarrow$ edges drawn u.a.r. from those incident to c,d
  end if
  $G' \leftarrow$ perform $(a, x, c), (b, y, d) \rightarrow (a, x, d), (b, y, c)$ on G
  $\mathsf{c}(\mathsf{dat}(G')) \leftarrow$ Equation (9)
  $p \leftarrow$ random real number in $[0, 1]$
  $a \leftarrow \min\left(1, \mathsf{c}(\mathcal{D})/\mathsf{c}(\mathcal{D}')\right)$
  if $p \leq a$ **then** $G \leftarrow G'$
  end if
end while
return $\mathsf{dat}(G)$

We leave for future work the development of a curveball-like approach for sampling sequence datasets from the null model. Jenkins et al. [27] propose other two null models for sequence datasets. Extending these null models to also preserve the BJDM is an interesting direction for future work.

7 Experimental evaluation

We now report on the results of our experimental evaluation of ALICE- A, ALICE- B, and ALICE- S. Our evaluation pursues three goals: empirically study the mixing time of the sampling algorithms, evaluate their scalability as the number of transactions increases, and show that the null model we introduce differs from that which only preserves the two fundamental properties, by showing that it leads to marking different hypotheses as significant.

Table 2 Datasets statistics: num. of transactions, num. of items, sum of transaction lengths, avg. transaction length, density, and number of caterpillars

Dataset	Trans Num	Item Num	Sum trans Lengths	AVG trans Length	Density	Num. Cater.
Iewiki	137	558	651	4.752	0.0085	10K
Kosarak	3000	5767	23,664	7.888	0.0014	88 M
Chess	3196	75	118,252	37.000	0.4933	9.93B
Foodmart	4141	1559	18,319	4.424	0.0028	954K
db-occ	10,000	8984	19,729	1.973	0.0002	7.5M
BMS1	59,602	497	149,639	2.511	0.0051	1.13B
BMS2	77,512	3340	358,278	4.622	0.0014	1.96B
Retail	88,162	16,470	908,576	10.306	0.0006	60B
SIGN	730	269	76,646	104.994	0.3903	696 M
LEVIATHAN	5834	9027	400,336	68.621	0.0076	22B
FIFA	20,450	2992	1,502,634	73.478	0.0246	159B
BIKE	21,078	69	327,844	15.554	0.2254	5.88B
BIBLE	36,369	13,907	1,610,501	44.282	0.0032	259B
BMS1	59,601	499	358,877	6.021	0.0121	1.13B

Datasets. We use eight real-world transactional datasets and six real-world sequence datasets,[12] listed in Table 2. Density is the ratio between the average transaction length and the number of items. **iewiki** is a user-edit dataset, where each transaction is a set of Wikibooks pages edited by the same user; **kosarak**, **BMS1**, **BMS2**, and **FIFA** are click-stream datasets; **chess** is a board-description datasets adapted from the UCI Chess (King-Rook vs King-Pawn) dataset; **foodmart** and **retail** are retail transaction datasets; **db-occ** includes user occupations taken from dbpedia; **SIGN** is a dataset of sign language utterance; **LEVIATHAN** and **BIBLE** are sentence datasets created from the novel Leviathan by Thomas Hobbes (1651) and the Bible, respectively; and in **BIKE** each sequence indicate the bike sharing stations where a bike was parked in Los Angeles over time.

Experimental Environment. We run our experiments on a 40-Core (2.40 GHz) Intel® Xeon® Silver 4210R machine, with 384GB of RAM, and running FreeBSD 14.0. Results are compared against GMMT [17], which is a swap randomization algorithm that samples from the null model that only maintains the two fundamental properties. The sampler GMMT-S is a variant of the SelfLoop version of GMMT that preserves the left and right degree sequences of the bipartite multi-graph representation of the observed sequence dataset. All the samplers are implemented in Java 1.8, and the code is available at https://github.com/acdmammoths/alice.

Convergence. To study the convergence of our samplers, we follow a procedure similar to the one proposed by Gionis et al. [17]. The mixing time, i.e., the number of steps needed for the state of the chain to be distributed according to π, is estimated by looking at the convergence of the *Average Relative Support Difference (ARSD)*, defined as

$$ARSD(\mathcal{D}^s) = \frac{1}{|\mathsf{FI}_\theta(\mathring{\mathcal{D}})|} \sum_{A \in \mathsf{FI}_\theta(\mathring{\mathcal{D}})} \frac{|\sigma_{\mathring{\mathcal{D}}}(A) - \sigma_{\mathcal{D}^s}(A)|}{|\sigma_{\mathring{\mathcal{D}}}(A)|},$$

[12] From www.philippe-fournier-viger.com/spmf/index.php and http://konect.cc/networks.

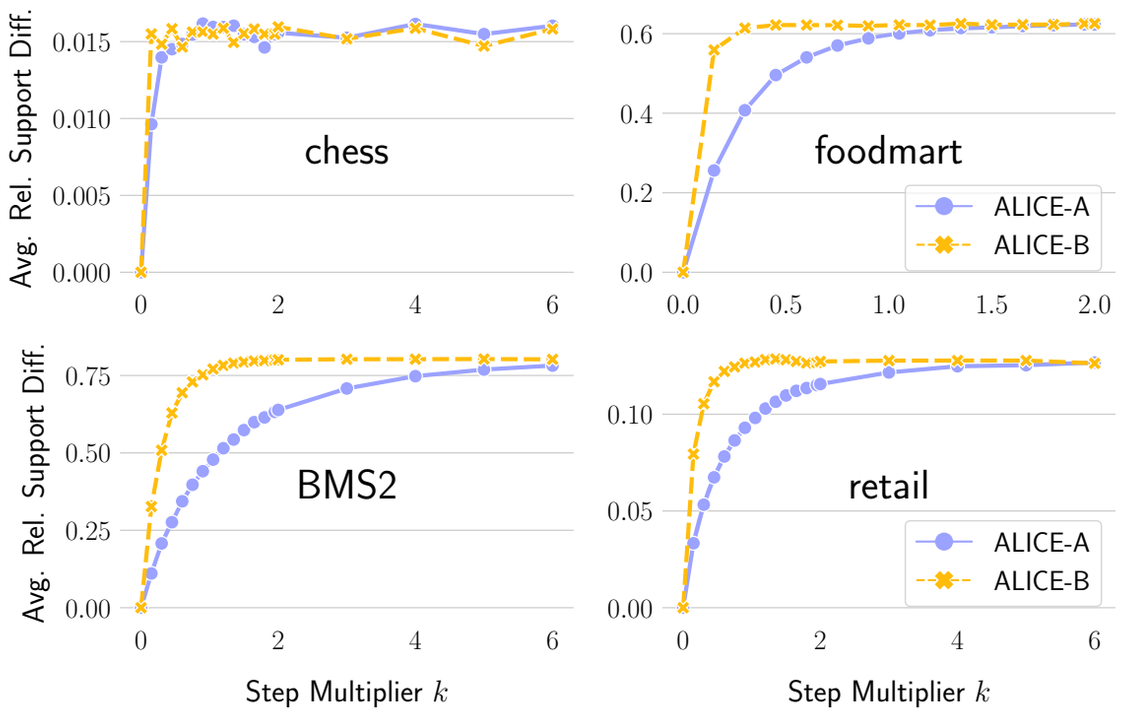


Fig. 9 Convergence of the samplers increasing the step number multiplier k, for chess (upper left), foodmart (upper right), BMS2 (lower left), and retail (lower right)

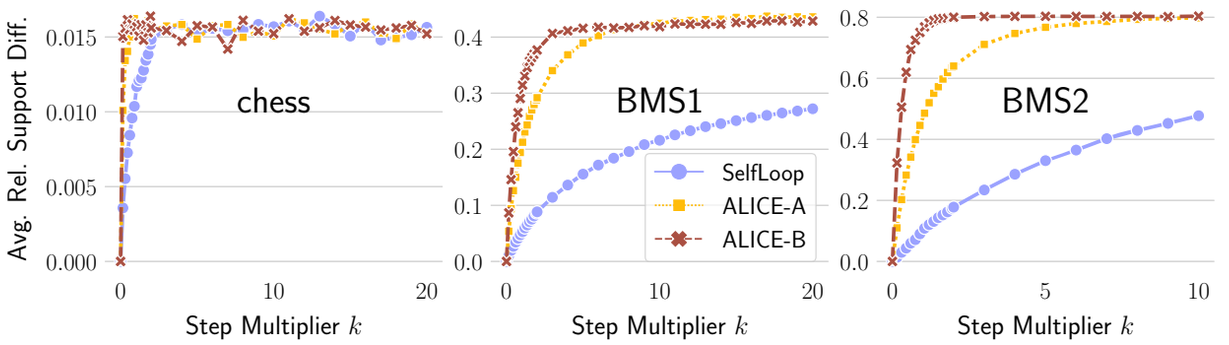


Fig. 10 Convergence of ALICE- A and ALICE- B vs SelfLoop increasing the step number multiplier k, for chess (left), BMS1 (middle), and BMS2 (right)

where $\mathcal{D}^s$ is the dataset obtained by the sampler after s steps. Figure 9 reports this quantity for chess (upper left), foodmart (upper right), BMS2 (lower left), and retail (lower right), for $s = \lfloor k \cdot w \rfloor$ with $k \in \{0, 0.15, 0.3, \ldots, 2, 3, \ldots, 6\}$ and $w = \sum_{t \in \mathring{\mathcal{D}}} |t|$. Results for other datasets were qualitatively similar. ALICE- B needs $^1/_3$ or even fewer steps than ALICE- A, thanks to the fact that it essentially performs multiple RSOs at each step (as each RBSO corresponds to one or more RSOs).

Despite the fewer number of *steps* needed, the *(wall clock) time* to convergence of ALICE- B (not reported in figures), however, is higher than that of ALICE- A. This difference is due to the fact that performing an RBSO, which is a more complex operation than an RSO, requires additional bookkeeping for each element in the set U (see Definition 3). In the worst cases (BMS1, and chess), ALICE- B takes almost 10x the time of ALICE- A to reach convergence. An interesting direction for future work is to study how to avoid this additional bookkeeping in ALICE- B to obtain the same advantage over ALICE- A observed for the number of steps to convergence also for the wall clock time.

Figure 10 compares the ARSD values obtained by ALICE with those measured in the states of the chain traversed by the naïve approach introduced in Sect. 5.1 (called SelfLoop in the

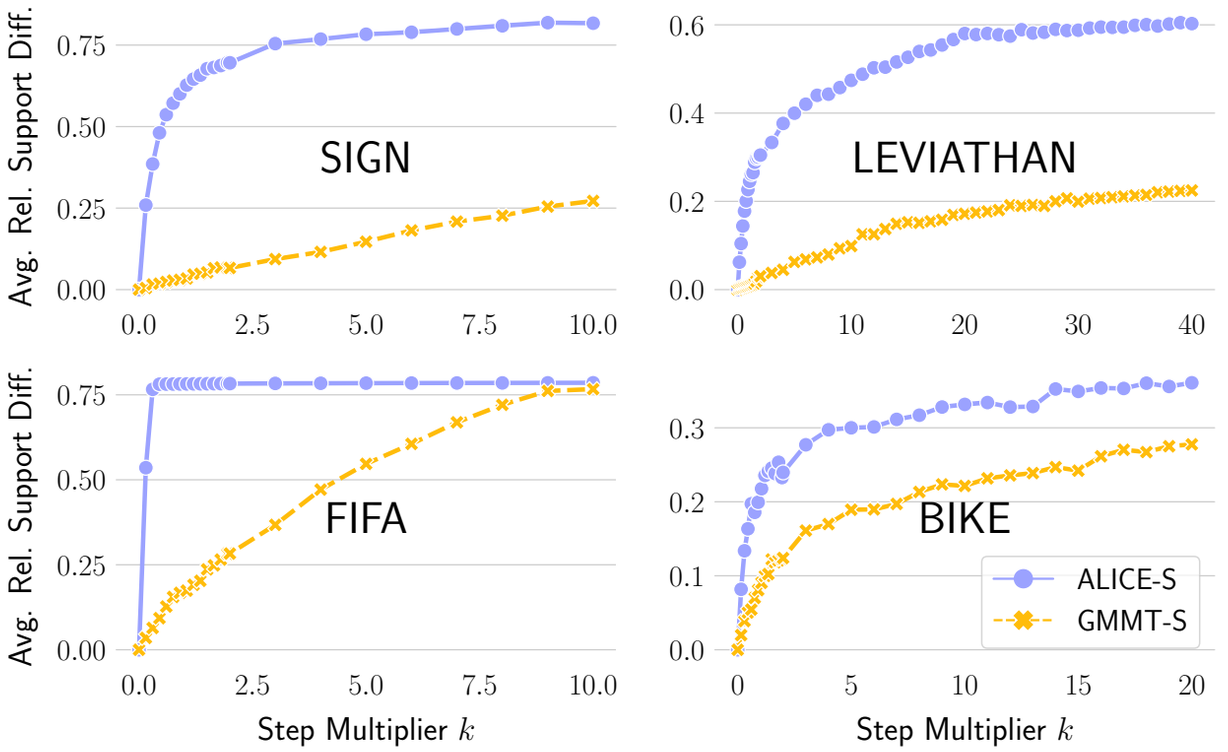


Fig. 11 Convergence of ALICE- S and GMMT-S increasing the step number multiplier k, for SIGN (upper left), LEVIATHAN (upper right), FIFA (lower left), and BIKE (lower right)

figure). Recall that, at each step, this approach draws two pairs (a, c) and (b, d) of row-column indices uniformly at random, and moves to the next state if $(a, c), (b, d) \rightarrow (a, d), (b, c)$ is a RSO. Especially for larger datasets, we observe that SelfLoop moves slowly in the state space, which prevents the ARSD from stabilizing even after $10w$ steps. As a result, a large number of steps is required to increase the likelihood of convergence, thus rendering SelfLoop impractical for use. In fact, the running time increases with the number of steps. In BMS1, for example, the ARSD for ALICE- B stabilizes around $k = 4$, with ALICE- B taking roughly 17 s to perform the $4w$ steps. In contrast, SelfLoop takes 397 s to perform the $10w$ steps.

We notice a similar behavior in Fig. 11, which illustrates the convergence of ALICE- S and GMMT-S for the sequence datasets SIGN (upper left), LEVIATHAN (upper right), FIFA (lower left), and BIKE (lower right). In this case, $w = |E|$, i.e. the number of edges in the multi-graph corresponding to the dataset. In SIGN and FIFA the ARSD stabilizes before $k = 3$ for ALICE- S, whereas for GMMT-S it stabilizes only in FIFA. In BIKE and LEVIATHAN both samplers move slowly, and thus, convergence is reached after almost $20w$ and $30w$ steps, respectively.

Scalability. To study the scalability of ALICE, we create synthetic datasets with increasing number of transactions and average transaction length 25, by using the IBM Quest generator [2]: five datasets with 100 items and 5k, 10k, 15k, 20k, and 100k transactions, and one dataset with 10k items and 1M transactions. For each sampler for transactional datasets, we perform 10k steps and compute the distribution of step times, reported in Fig. 12 (log values). For completeness, we include the step times of GMMT, although they are not really comparable to those of our algorithms, because GMMT samples from a different null set $\mathcal{Z}$ which includes datasets with different BJDMs. The median step time scales linearly with the size of the dataset. ALICE- A is the fastest sampler, requiring less than 8ms to perform a step in the largest dataset, and less than 1ms in most of the cases. In contrast, the step times of ALICE- B are characterized by more variability, as they depend on *(i)* whether the performed

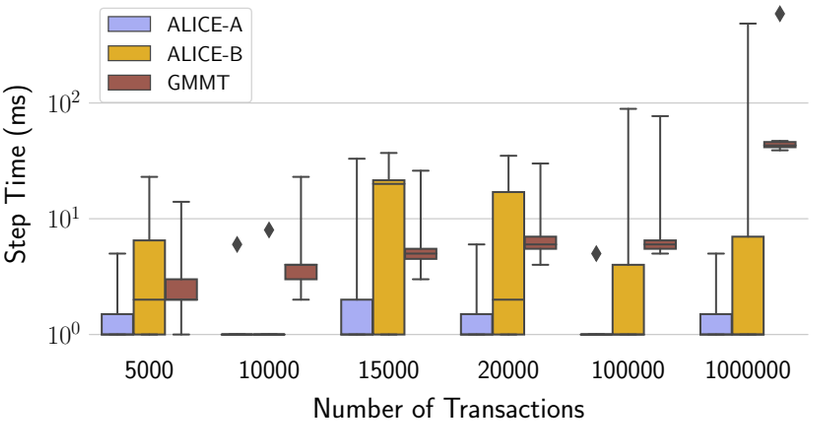


Fig. 12 Step times of the samplers in the synthetic datasets

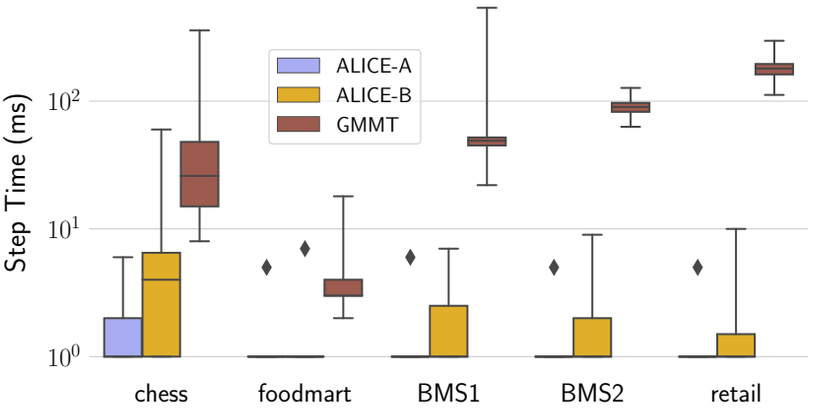


Fig. 13 Step times of the samplers in the real datasets (log times)

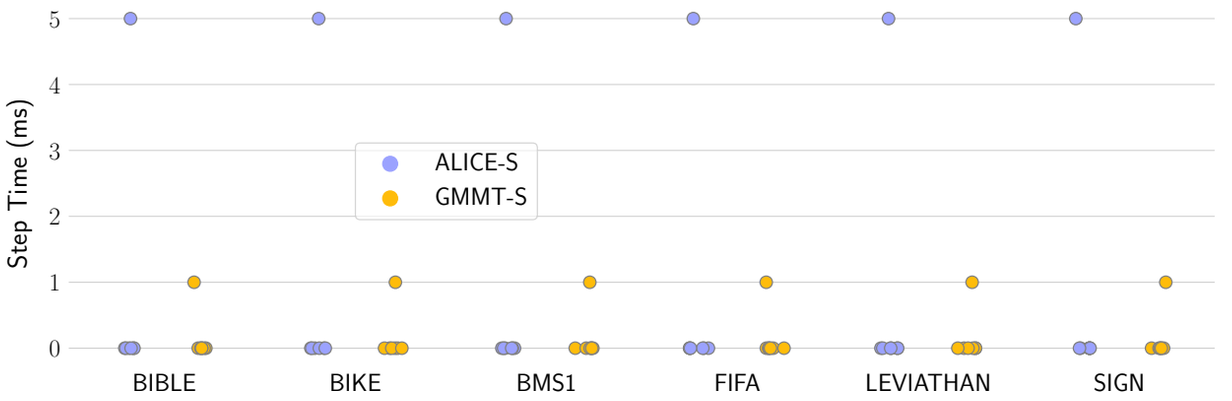


Fig. 14 Step times of the samplers in the real sequence datasets

RBSO is an rRBSO or a cRBSO, and *(ii)* the size of the set U: the time required to compute $\mathsf{c}(\mathcal{D})$ is larger for cRBSO, and it grows with the size of U.

Figure 13 reports the distribution of the time required to perform a step for each sampler in each transactional dataset. The step time of ALICE- B tends to be larger in chess, despite it not being the largest dataset. This fact is due to the high density of this dataset, and its large transaction length (37). Hence, the size of U is usually high. In foodmart, on the other hand, the average transaction length is 4.42 and the average item support is 5.6, so the size of U is often 1. An algorithmic improvement in the bookkeeping due to the size of U would result in better performance of ALICE- B, as mentioned above.

Figure 14 shows the distribution of step times for ALICE- S and GMMT-S in the sequence

Table 3 No. of FIs in the original dataset $\mathring{\mathcal{D}}$, avg. no. of FIs in the sample $\mathcal{D}_i$, estimated p value $\tilde{p}_{\mathring{\mathcal{D}},H_0}$ for H_0 from Eq. (1)

Dataset	$\lvert\mathsf{FI}_\theta(\mathring{\mathcal{D}})\rvert$	Sampler	$\frac{\sum_1^T \lvert\mathsf{FI}_\theta(\mathcal{D}_i)\rvert}{T}$	$\tilde{p}_{\mathring{\mathcal{D}},H_0}$
Iewiki	65,665	ALICE- A	173	2.3E-4
$\theta = 1.4\text{E-}2$		ALICE- B	171	2.3E-4
		GMMT	2257	1.8E-2
Kosarak	6277	ALICE- A	4865	2.3E-4
$\theta = 3.0\text{E-}3$		ALICE- B	4130	2.3E-4
		GMMT	31,774	1.0E-0
Chess[14]	8227	ALICE- A	6183	4.6E-4
$\theta = 0.8$		ALICE- B	6182	4.6E-4
		GMMT	6179	4.6E-4
Foodmart	4247	ALICE- A	2229	2.3E-4
$\theta = 3.0\text{E-}4$		ALICE- B	2228	2.3E-4
		GMMT	2226	2.3E-4
db-occ	834	ALICE- A	702	2.3E-4
$\theta = 5.0\text{E-}4$		ALICE- B	703	2.3E-4
		GMMT	598	2.3E-4
BMS1	3991	ALICE- A	1998	4.6E-4
$\theta = 0.001$		ALICE- B	1609	4.6E-4
		GMMT	1800	4.6E-4

datasets. The performance of ALICE- S is comparable with that of ALICE- A, as they follow a similar approach to sample the swap operations to perform. The median step time is always < 1, and the algorithm takes at most 5ms to perform a step. The step times of GMMT-S are far lower than its counterpart for transactional datasets, because this algorithm does not require bookkeeping to compute the transition acceptance probability. we recall that also in this case the running time of GMMT-S is not really comparable with that of our algorithm because they sample from different null models.

Significance of the Number of Frequent Itemsets. To show that the null model we introduce is different than the one that only preserves the two fundamental properties, we test the null hypothesis H_0 from Eq. (1), and estimate the p-value as in Eq. (3) with $T = 4352$ samples from the null model, for each sampler.[13] We remark that this kind of hypothesis is just a simple but clear example of the tasks that can (and should) be formed to assess the statistical validity of results obtained from transactional datasets. Other tasks include, for example, mining the statistically significant frequent itemsets. We limit ourselves to this task because it is straightforward to present and it is sufficient to show the significant (pun intended) difference between preserving the BJDM, as our null model does, and not preserving it.[14]

Table 3 reports the number of FIs in the observed dataset, the average number of FIs in the sampled datasets, and the empirical p-value, for datasets where GMMT terminated within two days. The fact that (very) different p-values can be obtained with ALICE and with GMMT, which sample from a different null model, highlights the striking impact of preserving the BJDM. As an example, for any critical value in $(0.00023, 0.01815)$, in iewiki H_0 would be

[13] The number of steps is empirically fixed according to the results obtained in the convergence experiment.

[14] For chess and BMS1, $T = 2176$, due to the prohibitive running time of GMMT.

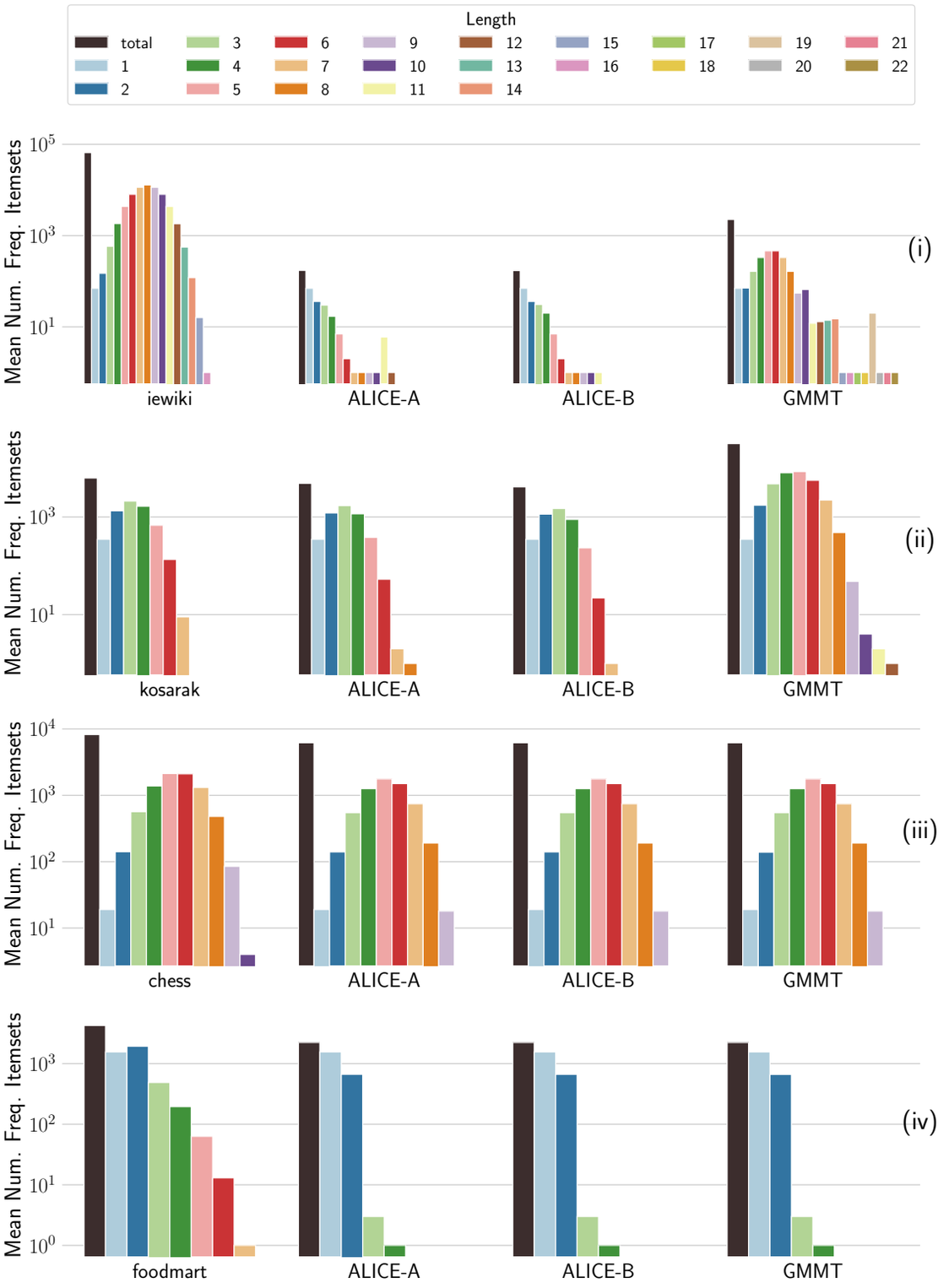


Fig. 15 Mean number of frequent itemsets per length for ALICE- A, ALICE- B, and GMMT, in iewiki (i), kosarak (ii), chess (iii), and foodmart (iv)

rejected under the null model we introduce, but not under the null model that only preserves the two fundamental properties.

Figures 15 and 16 show the distribution of the number of FIs of different lengths in the original datasets, and the average of the same quantity over the datasets sampled by the different samplers. For BMS2 and retail we do not report results for GMMT, due to its prohibitive running time. Since they sample from the same null model, ALICE- A and ALICE- B obtain the same distribution (up to sampling noise), which is quite different than the one

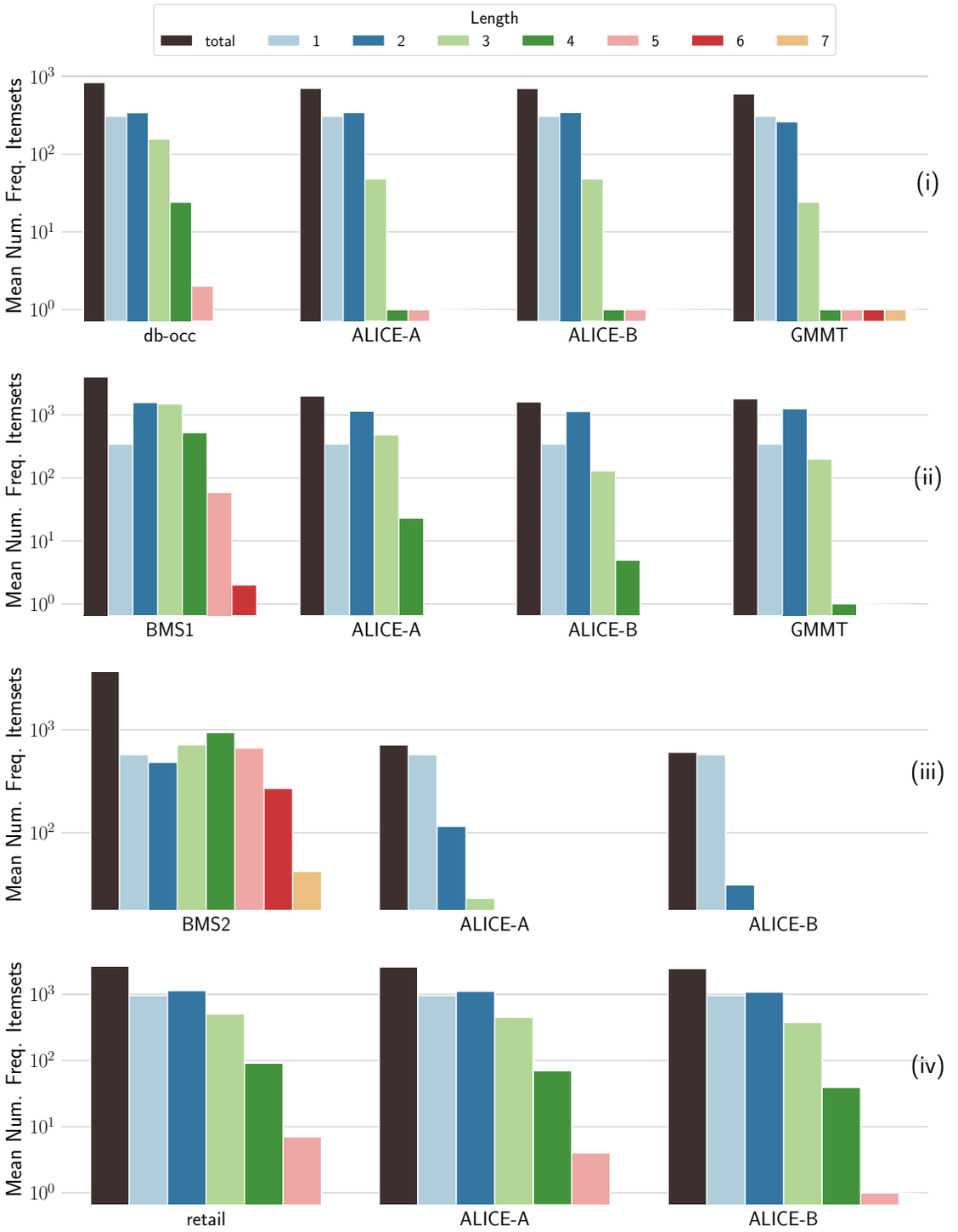


Fig. 16 Mean number of frequent itemsets per length for ALICE- A, ALICE- B, and GMMT (when available), in db-occ (i), BMS1 (ii), BMS2 (iii), and retail (iv)

obtained by GMMT. Note that whether the sampled datasets have more or less FIs than the observed dataset depends both on the null model and on the dataset. For instance, in iewiki (Fig. 15, i) datasets sampled from all null models have fewer FIs than the observed one. Conversely, in kosarak (Fig. 15, ii) the BJDM-preserving null model produces samples with a similar number of FIs, while the datasets sampled from the null model that preserves the two fundamental properties have a larger number of FIs. In addition, in iewiki, the samples from this latter model usually contain FIs of length larger than any FIs in the observed dataset: the max length of a FI in iewiki is 16, whereas it grows to 22 in the datasets sampled by GMMT. In kosarak, the datasets sampled by GMMT contain both a larger number of FIs per length

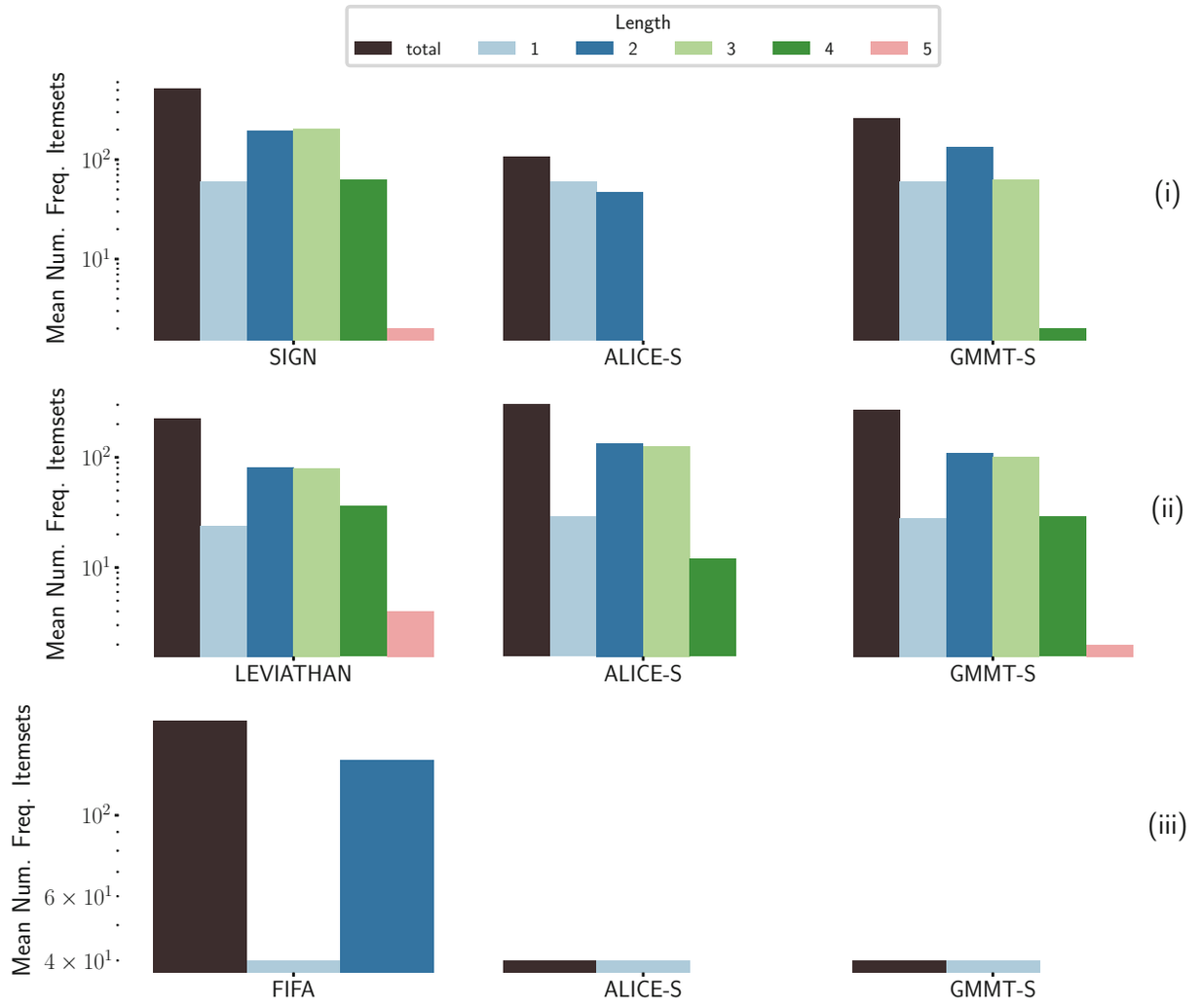


Fig. 17 Mean number of frequent itemsets per length for ALICE- S and GMMT-S, in SIGN (i), LEVIATHAN (ii), and FIFA (iii)

and FIs of larger length (12 vs. 7). The increase in the number of FIs of length three, leads to a substantial difference in the number of FIs of length in the range [4, 7]: we observe up to 246x more FIs in the sampled datasets. In contrast, since all the transactions in chess have the same length, we observe (Fig. 15, iii) similar average numbers of FIs across the samplers. In this dataset, any swap operation performed by GMMT is actually a RBSO, and hence also the datasets sampled by GMMT preserve the BJDM. Similarly, the fact that the nodes in the graph representation of foodmart (Fig. 15, iv) display high assortativity indicates that most of the swap operations of GMMT are RBSO. In fact, when the product between the two marginals is close to the BJDM in terms of Frobenius norm, preserving the marginals *almost* preserves the BJDM As a consequence, also in this case, the distribution of the numbers of FIs for GMMT is similar to that for ALICE.

We can see that the distribution of the number of FIs in the observed dataset is always different from those obtained from the sampled datasets. In particular, the longer itemsets are, in general, less frequent in the sampled datasets than in the original dataset. As an example, BMS2 (Fig. 16, iii) contains many FIs of length larger than three (roughly 52% of the FIs), while most of the FIs in the datasets sampled by ALICE have length one.

Figures 17 and 18 present the distribution of frequent sequential itemsets of different lengths in the original sequence datasets, and the average of the same quantity over the datasets sampled by ALICE- S and GMMT-S. The frequency thresholds used are taken from

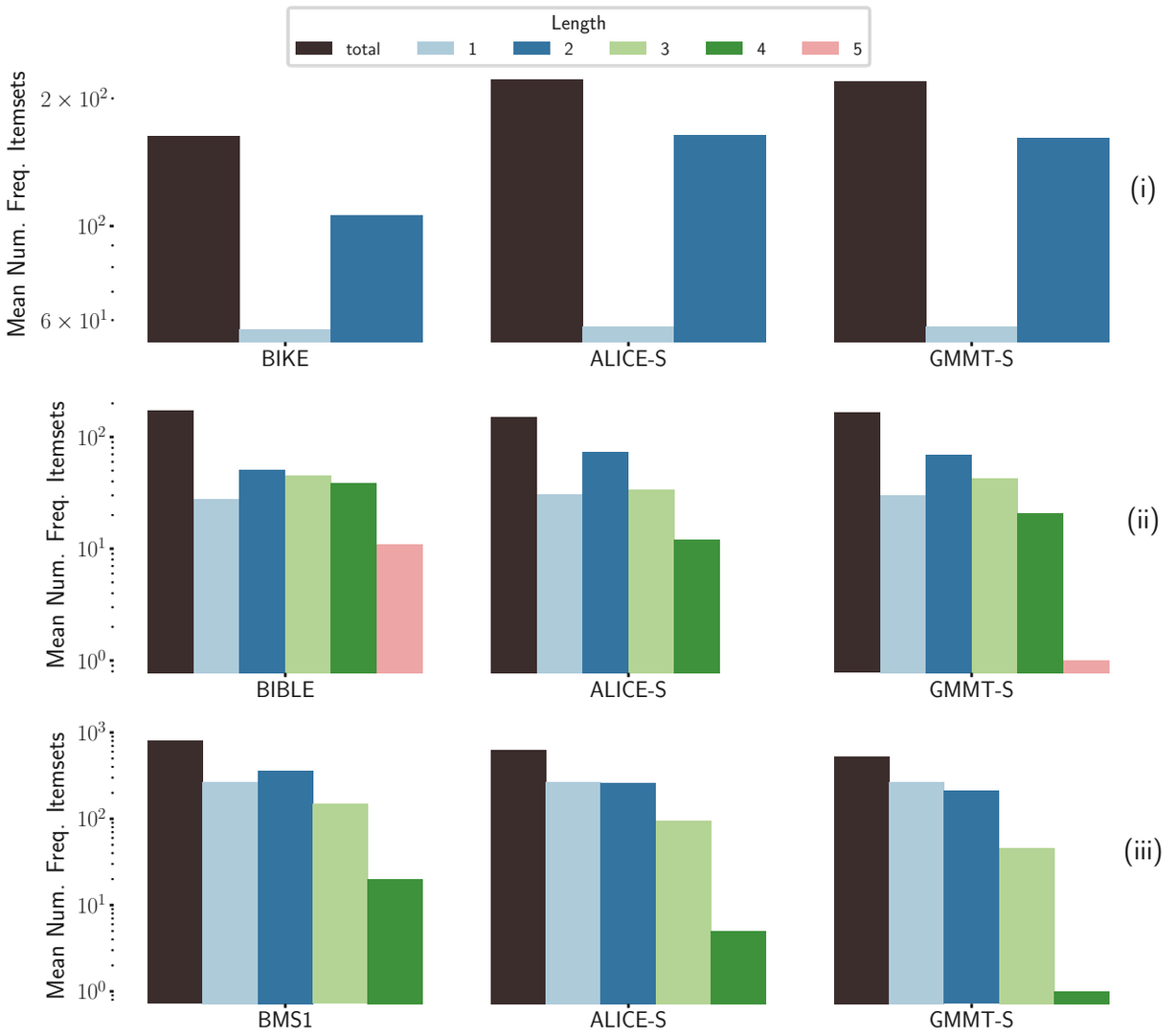


Fig. 18 Mean number of frequent itemsets per length for ALICE- S and GMMT-S, in BIKE (i), BIBLE (ii), and BMS1 (iii)

[60]: 0.4 for SIGN, 0.15 for LEVIATHAN, 0.275 for FIFA, 0.025 for BIKE, 0.1 for BIBLE, and 0.002 for BMS1. The number of samples extracted is always 4352 and the number of steps performed by ALICE- S is $10w$, while it is $50w$ for GMMT-S. Also in this case, w is the number of edges in the multi-graph corresponding to the dataset. Similarly to the transactional dataset case, we tend to observe frequent itemsets of larger size in the datasets sampled by GMMT-S, except in the case of few frequent itemsets in the original dataset (e.g., FIFA and BIKE). In such cases, only trivial itemsets are frequent, and their frequencies tend to be preserved by preserving the two fundamental properties.

Thanks to these results, we conclude that the BJDM captures important additional information about the data generation process. Therefore, using a null model that preserves it may lead to very different conclusions about the data generation process compared to one that does not. These results highlight, once more, how the choice of the null model by the user must be extremely deliberate.

8 Conclusion

We introduced a novel null model for statistically assessing the results obtained from an observed transactional or sequence dataset, preserving its Bipartite Joint Degree Matrix (BJDM). On transactional datasets, maintaining this property enforces, in addition to the dataset size, transaction lengths, and item supports, also the preservation of the number of *caterpillars* of the bipartite graph corresponding to the observed dataset, which is a natural and important property that captures additional structure. We describe ALICE, a suite of Markov chain Monte Carlo algorithms for sampling datasets from the null models. The results of our experimental evaluation show that ALICE scales well and that, when testing results w.r.t. our null models, different results are marked as significant than when using existing null models.

A good direction for future work includes a rigorous theoretical analysis and/or experimental evaluation of the trade-offs between the time taken to perform a single step and the mixing time of the Markov chain when using different neighbor sampling distribution. Toward making statistically sound knowledge discovery a reality, we also suggest the development of even more descriptive null models (e.g., by preserving the number of *butterflies* [51]), and of efficient procedures to sample from them, which is usually the challenging aspect. Another interesting direction is proposing null models for real-valued transactional datasets, such as those used for high-utility itemsets mining.

Acknowledgements This work is sponsored in part by NSF award IIS-2006765.

References

1. Abuissa M, Lee A, Riondato M (2023) ROhAN: Row-order agnostic null models for statistically-sound knowledge discovery. Data Min Knowl Discov. https://doi.org/10.1007/s10618-023-00938-4
2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, VLDB '94, pp 487–499
3. Akoglu L, Faloutsos C (2009) Rtg: A recursive realistic graph generator using random typing. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, pp 13–28
4. Aksoy SG, Kolda TG, Pinar A (2017) Measuring and modeling bipartite graphs with community structure. J Complex Netw 5(4):581–603
5. Amanatidis G, Green B, Mihail M (2015) Graphic realizations of joint-degree matrices. arXiv preprint arXiv:1509.07076
6. Bie TD (2010) Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Min Knowl Disc 23(3):407–446. https://doi.org/10.1007/s10618-010-0209-3
7. Bonferroni CE (1936) Teoria statistica delle classi e calcolo delle probabilità. Pubblicazioni del Regio Istituto Superiore di Scienze Economiche e Commerciali di Firenze 8:3–62
8. Bonifati A, Holubová I, Prat-Pérez A et al (2020) Graph generators: State of the art and open challenges. ACM Comput Surv (CSUR) 53(2):1–30
9. Boroojeni AA, Dewar J, Wu T et al (2017) Generating bipartite networks with a prescribed joint degree distribution. J Complex Netw 5(6):839–857
10. Brin S, Motwani R, Silverstein C (1997) Beyond market baskets: Generalizing association rules to correlations. In: Proceedings of the 1997 ACM SIGMOD international conference on management of data, SIGMOD '97, pp 265–276
11. Carstens CJ (2015) Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. Phys Rev E 91(4):042812
12. Cimini G, Squartini T, Saracco F et al (2019) The statistical physics of real-world networks. Nat Rev Phys 1(1):58–71
13. Czabarka É, Dutle A, Erdős PL et al (2015) On realizations of a joint degree matrix. Discret Appl Math 181:283–288

14. Duivesteijn W, Knobbe A (2011) Exploiting false discoveries–statistical validation of patterns and quality measures in subgroup discovery. In: 2011 IEEE 11th international conference on data mining, IEEE, pp 151–160
15. Ferkingstad E, Holden L, Sandve GK (2015) Monte Carlo null models for genomic data. Stat Sci 30(1):59–71
16. Fischer R, Leitao JC, Peixoto TP et al (2015) Sampling motif-constrained ensembles of networks. Phys Rev Lett 115(18):188701
17. Gionis A, Mannila H, Mielikäinen T et al (2007) Assessing data mining results via swap randomization. ACM Trans Knowl Discov Data (TKDD) 1(3):14
18. Goeman JJ, Solari A (2014) Multiple hypothesis testing in genomics. Stat Med 33(11):1946–1978
19. Greenhill C (2022) Generating graphs randomly. arXiv preprint arXiv:2201.04888
20. Günnemann S, Dao P, Jamali M, et al (2012) Assessing the significance of data mining results on graphs with feature vectors. In: 2012 IEEE 12th international conference on data mining, pp 270–279, https://doi.org/10.1109/ICDM.2012.70
21. Hämäläinen W (2010) StatApriori: an efficient algorithm for searching statistically significant association rules. Knowl Inf Syst 23(3):373–399. https://doi.org/10.1007/s10115-009-0229-8
22. Hämäläinen W (2016) New upper bounds for tight and fast approximation of Fisher's exact test in dependency rule mining. Comput Stat Data Anal 93:469–482
23. Hämäläinen W, Webb GI (2019) A tutorial on statistically sound pattern discovery. Data Min Knowl Disc 33(2):325–377
24. Hanhijärvi S (2011) Multiple hypothesis testing in pattern discovery. In: International conference on discovery science, Springer, pp 122–134
25. Hanhijärvi S, Garriga GC, Puolamäki K (2009) Randomization techniques for graphs. In: Proceedings of the 2009 SIAM international conference on data mining, SDM '09, pp 780–791, https://doi.org/10.1137/1.9781611972795.67
26. He J, Li F, Gao Y et al (2021) Resampling-based stepwise multiple testing procedures with applications to clinical trial data. Pharm Stat 20(2):297–313
27. Jenkins S, Walzer-Goldfeld S, Riondato M (2022) SPEck: Mining statistically-significant sequential patterns efficiently with exact sampling. Data Min Knowl Discov 36(4)
28. Karrer B, Newman ME (2011) Stochastic blockmodels and community structure in networks. Phys Rev E 83(1):016107
29. Kim S, Kirkland S (2022) Gram mates, sign changes in singular values, and isomorphism. Linear Algebra Appl 644:108–148
30. Kirkland S (2018) Two-mode networks exhibiting data loss. J Complex Netw 6(2):297–316
31. Komiyama J, Ishihata M, Arimura H, et al (2017) Statistical emerging pattern mining with multiple testing correction. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 897–906
32. Lehmann EL, Romano JP (2022) Testing statistical hypotheses, 4th edn. Springer, Berlin
33. Lijffijt J, Papapetrou P, Puolamäki K (2014) A statistical significance testing approach to mining the most informative set of patterns. Data Min Knowl Disc 28(1):238–263. https://doi.org/10.1007/s10618-012-0298-2
34. Llinares-López F, Sugiyama M, Papaxanthos L, et al (2015) Fast and memory-efficient significant pattern mining via permutation testing. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 725–734
35. Low-Kam C, Raïssi C, Kaytoue M, et al (2013) Mining statistically significant sequential patterns. In: 2013 IEEE 13th international conference on data mining, IEEE, pp 488–497
36. Megiddo N, Srikant R (1998) Discovering predictive association rules. In: Proceedings of the 4th international conference on knowledge discovery and data mining, KDD '98, pp 274–278
37. Minato Si, Uno T, Tsuda K, et al (2014) A fast method of statistical assessment for combinatorial hypotheses based on frequent itemset enumeration. In: Joint European conference on machine learning and knowledge discovery in databases, Springer, pp 422–436
38. Mitzenmacher M, Upfal E (2005) Probability and computing: randomized algorithms and probabilistic analysis. Cambridge University Press, Cambridge
39. Newman MEJ (2002) Assortative mixing in networks. Phys Rev Lett 89(20):208701. https://doi.org/10.1103/PhysRevLett.89.208701
40. Ojala M (2010) Assessing data mining results on matrices with randomization. In: 2010 IEEE international conference on data mining, pp 959–964, https://doi.org/10.1109/ICDM.2010.20
41. Ojala M, Garriga GC, Gionis A, et al (2010) Evaluating query result significance in databases via randomizations. In: Proceedings of the 2010 SIAM international conference on data mining (SDM), pp 906–917, https://doi.org/10.1137/1.9781611972801.79

42. Orsini C, Dankulov MM, Colomer-de Simón P et al (2015) Quantifying randomness in real networks. Nat Commun 6(1):1–10
43. Papaxanthos L, Llinares-López F, Bodenham D, et al (2016) Finding significant combinations of features in the presence of categorical covariates. In: Advances in neural information processing systems, pp 2279–2287
44. Pellegrina L, Vandin F (2020) Efficient mining of the most significant patterns with permutation testing. Data Min Knowl Disc 34:1201–1234
45. Pellegrina L, Riondato M, Vandin F (2019a) Hypothesis testing and statistically-sound pattern mining. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. ACM, KDD '19, pp 3215–3216. https://doi.org/10.1145/3292500.3332286
46. Pellegrina L, Riondato M, Vandin F (2019b) SPuMANTE: Significant pattern mining with unconditional testing. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. ACM, KDD '19, pp 1528–1538. https://doi.org/10.1145/3292500.3330978
47. Pinxteren S, Calders T (2021) Efficient permutation testing for significant sequential patterns. In: Proceedings of the 2021 SIAM international conference on data mining (SDM), SIAM, pp 19–27
48. Preti G, De Francisci Morales G, Riondato M (2022) ALICE and the caterpillar: A more descriptive null models for assessing data mining results. In: Proceedings of the 22nd IEEE international conference on data mining, pp 418–427
49. Relator RT, Terada A, Sese J (2018) Identifying statistically significant combinatorial markers for survival analysis. BMC Med Genom 11(2):31
50. Ritchie M, Berthouze L, Kiss IZ (2017) Generation and analysis of networks with a prescribed degree sequence and subgraph family: higher-order structure matters. J Complex Netw 5(1):1–31
51. Sanei-Mehri SV, Sariyuce AE, Tirthapura S (2018) Butterfly counting in bipartite networks. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2150–2159
52. Saracco F, Di Clemente R, Gabrielli A et al (2015) Randomizing bipartite networks: the case of the world trade web. Sci Rep 5(1):1–18
53. Sese J, Terada A, Saito Y, et al (2014) Statistically significant subgraphs for genome-wide association study. In: Statistically sound data mining, pp 29–36
54. Silva ME, Paredes P, Ribeiro P (2017) Network motifs detection using random networks with prescribed subgraph frequencies. In: International workshop on complex networks, Springer, pp 17–29
55. Sugiyama M, Llinares-López F, Kasenburg N, et al (2015) Significant subgraph mining with multiple testing correction. In: Proceedings of the 2015 SIAM international conference on data mining, SIAM, pp 37–45
56. Terada A, Okada-Hatakeyama M, Tsuda K et al (2013) Statistical significance of combinatorial regulations. Proc Natl Acad Sci 110(32):12996–13001
57. Terada A, Tsuda K, Sese J (2013b) Fast Westfall-Young permutation procedure for combinatorial regulation discovery. In: 2013 IEEE international conference on bioinformatics and biomedicine (BIBM), IEEE, pp 153–158
58. Terada A, Kim H, Sese J (2015) High-speed Westfall-Young permutation procedure for genome-wide association studies. In: Proceedings of the 6th ACM conference on bioinformatics, computational biology and health informatics, ACM, pp 17–26
59. Tillman B, Markopoulou A, Gjoka M et al (2019) 2k+ graph construction framework: targeting joint degree matrix and beyond. IEEE/ACM Trans Netw 27(2):591–606
60. Tonon A, Vandin F (2019) Permutation strategies for mining significant sequential patterns. In: 2019 IEEE international conference on data mining (ICDM), IEEE, pp 1330–1335
61. Van Koevering K, Benson A, Kleinberg J (2021) Random graphs with prescribed k-core sequences: a new null model for network analysis. Proc Web Conf 2021:367–378
62. Verhelst ND (2008) An efficient MCMC algorithm to sample binary matrices with fixed marginals. Psychometrika 73(4):705–728
63. Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57
64. Vreeken J, Tatti N (2014) Interesting patterns. In: Frequent pattern mining. Springer, pp 105–134
65. Webb GI (2007) Discovering significant patterns. Mach Learn 68(1):1–33
66. Westfall PH, Young SS (1993) Resampling-based multiple testing: examples and methods for p-value adjustment. Wiley-Interscience
67. Wu J, He Z, Gu F et al (2016) Computing exact permutation p-values for association rules. Inf Sci 346:146–162

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Giulia Preti is a researcher in the Social Algorithmics Team (SALT) at CENTAI in Turin, Italy, working on null models for assessing the statistical significance of the results of pattern mining algorithms in the context of transactional databases, graphs, and hypergraphs. She got her PhD in Information and Communication Technology at the University of Trento, Italy, working on techniques for mining relevant structures in dynamic and heterogeneous datasets. She has been PC Chair of several conferences including SDM, WSDM, WWW, AAAI, and ECMLPKDD, and a reviewer for computer science journals such as TKDE and TKDD.

Gianmarco De Francisci Morales is a Principal Researcher at CENTAI, a private research institute that focuses on Artificial Intelligence and Complex Systems, where he leads the Social Algorithmics Team (SALT). Previously, he worked as a Senior Researcher at ISI Foundation in Turin, as a Scientist at Qatar Computing Research Institute in Doha, as a Visiting Scientist at Aalto University in Helsinki, as a Research Scientist at Yahoo Labs in Barcelona, and as a Research Associate at ISTI-CNR in Pisa. He received his PhD in Computer Science and Engineering from the IMT Institute for Advanced Studies of Lucca in 2012. His research focuses on computational social science and data mining, with an emphasis on polarization and echo chambers on social media. He is a member of ELLIS, European Laboratory for Learning and Intelligent Systems. He has been a member of the open-source community of the Apache Software Foundation, where he has worked on the Hadoop ecosystem, and has been a committer for the Apache Pig project. He was one of the lead developers of Apache SAMOA, an open-source platform for mining big data streams. He commonly serves on the PC of several major conferences in the area of data mining, including WSDM, WWW, KDD, and ICWSM. He co-organized the workshop series on Social News on the Web (SNOW), co-located with the WWW conference. He has published more than 90 scientific articles and won best paper awards at WSDM, CHI, WebSci, and SocInfo.

Matteo Riondato is associate professor of computer science at Amherst College. His research focuses on algorithms for knowledge discovery, data mining, and machine learning with strong statistical and computational guarantees. He received a NSF CAREER award and other NSF grants. His work was recognized with best-of-conference awards at SDM'14, KDD'16, ICDM'18, and WSDM'21.

Lightning Fast and Space Efficient k-clique Counting

Xiaowei Ye
Beijing Institute of Technology
Beijing, China
yexiaowei@bit.edu.cn

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Qiangqiang Dai
Beijing Institute of Technology
Beijing, China
qiangd66@gmail.com

Hongzhi Chen
ByteDance
Beijing, China
chenhongzhi@bytedance.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

K-clique counting is a fundamental problem in network analysis which has attracted much attention in recent years. Computing the count of k-cliques in a graph for a large k (e.g., $k = 8$) is often intractable as the number of k-cliques increases exponentially w.r.t. (with respect to) k. Existing exact k-clique counting algorithms are often hard to handle large dense graphs, while sampling-based solutions either require a huge number of samples or consume very high storage space to achieve a satisfactory accuracy. To overcome these limitations, we propose a new framework to estimate the number of k-cliques which integrates both the exact k-clique counting technique and two novel color-based sampling techniques. The key insight of our framework is that we only apply the exact algorithm to compute the k-clique counts in the *sparse regions* of a graph, and use the proposed sampling-based techniques to estimate the number of k-cliques in the *dense regions* of the graph. Specifically, we develop two novel dynamic programming based k-color set sampling techniques to efficiently estimate the k-clique counts, where a k-color set contains k nodes with k different colors. Since a k-color set is often a good approximation of a k-clique in the dense regions of a graph, our sampling-based solutions are extremely efficient and accurate. Moreover, the proposed sampling techniques are space efficient which use near-linear space w.r.t. graph size. We conduct extensive experiments to evaluate our algorithms using 8 real-life graphs. The results show that our best algorithm is at least one order of magnitude faster than the state-of-the-art sampling-based solutions (with the same relative error 0.1%) and can be up to three orders of magnitude faster than the state-of-the-art exact algorithm on large graphs.

CCS CONCEPTS

• **Networks** → **Data path algorithms**; • **Theory of computation** → **Randomness, geometry and discrete structures**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512167

KEYWORDS

k-clique counting, cohesive subgraphs, graph sampling

ACM Reference Format:
Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning Fast and Space Efficient k-clique Counting . In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3512167

1 INTRODUCTION

Real-life networks, such as social networks, web graphs, and biological networks, often contain frequently-occurring small subgraph structures. Such frequent small subgraphs are referred to as network motifs [31]. Counting the motifs is a fundamental tool in many network analysis applications, including social network analysis, community detection, and bioinformatics [10, 18, 31, 35, 38]. Perhaps the most elementary motif in a graph is the k-clique which has been widely used in a variety of network analysis applications [7, 10, 31, 39, 42].

Given a graph G, a k-clique is a complete subgraph of G with k nodes. Counting the k cliques in a graph has found many important applications in dense subgraph mining and social network analysis. For example, Sariyüce et al. [37] proposed a nucleus decomposition method to find the hierarchy of dense subgraphs, which uses the k-clique counting operator as a basic building block. Tsourakakis [42] studied a k-clique densest subgraph problem which also uses the k-clique counting operator as a building block. Additionally, the k-clique counting operator has also been applied to detect higher-order organizations in social networks [6, 45].

Motivated by the above applications, many practical k-clique counting algorithms have been proposed [2, 13, 15, 19, 23, 24, 28, 29, 36]. Existing k-clique counting algorithms can be classified into (1) exact k-clique counting methods, and (2) sampling-based approximation solutions. Chiba and Nishizeki [13] developed the first exact k-clique counting algorithm based on k-clique enumeration which is very efficient on real-life sparse graphs for a small k. Such an algorithm was recently improved by Finocchi et al. [19] based on a degree ordering technique. Subsequently, Danisch et al. [15] further improved this algorithm by using a degeneracy ordering technique [30]. More recently, Li et al. [28] developed a further improved algorithm based on a hybrid of degeneracy and color ordering technique. All these exact k-clique counting algorithms are based on k-clique enumeration, which are typically intractable on large graphs for a large k (e.g., $k \geq 8$) due to combinatorial

explosion. To overcome this issue, Jain and Seshadhri developed an elegant algorithm, called PIVOTER, based on a classic pivoting technique which was widely used for pruning the search branches in maximal clique enumeration [41]. The key idea of PIVOTER is that it can implicitly construct a succinct clique tree (SCT) by using the pivoting technique in the search procedure. Such a SCT structure maintains a unique representation of all k-cliques, but its size is much smaller than the number of k-cliques. PIVOTER was shown to be much faster than previous k-clique enumeration based algorithms [13, 15, 19, 28, 29].

Although PIVOTER is often very efficient for handling real-life sparse graphs, it may still have a very deep recursion tree when processing the dense regions of the graph, which is the main bottleneck of the PIVOTER algorithm. Moreover, PIVOTER is based on the idea of enumeration of maximal cliques to count the k-cliques. It is often not very fast on the dense regions of the graph, because the dense regions of the graph may contain many maximal cliques (with complicated overlap relationships), resulting in a large search tree of Pivoter (e.g., see the results on the LiveJournal dataset in [24]).

Approximation solutions based on sampling are typically able to handle large dense graphs when k is not very large [9, 23, 36]. However, to achieve a desired accuracy, previous sampling-based solutions either require a huge number of samples [26, 36, 44] or consume very high storage space [2, 8, 9, 23] for a relatively large k (e.g., $k \geq 8$). The state-of-the-art sampling-based approximation algorithm is the TuranShadow algorithm which was proposed by Jain and Seshadhri [23]. As shown in [23], TuranShadow is much faster and more accurate than the other sampling-based algorithms. The main limitation of TuranShadow is that it needs to take $O(n\alpha^{(k-1)} + m)$ time and $O(n\alpha^{(k-2)})$ space to construct a data structure called Tuŕan Shadow for sampling, where α denotes the arboricity of the graph [13]. Therefore, on large graphs, TuranShadow is very costly for a large k.

To overcome the limitations of the state-of-the-art algorithms, we propose a new framework to estimate the number of k-cliques in a graph which integrates both the exact PIVOTER algorithm and two newly-developed sampling-based techniques. Our framework is based on a simple but effective observation: PIVOTER is extremely efficient to compute the number of k-cliques in the *sparse regions* of the graph, while sampling-based solutions are often very efficient and accurate to estimate the k-clique counts in the *dense regions* of a graph. Base on this crucial observation, we can first partition the graph into sparse and dense regions. Then, for the sparse regions, we invoke PIVOTER to exactly compute the k-clique counts. For the dense regions, we propose two novel sampling technique based on a concept of graph coloring [3] to estimate the k-clique counts. Specifically, we first present a new concept called k-color set which denotes a set of k nodes with k different colors. Then, we propose a dynamic programming (DP) based k-color set sampling algorithm to estimate the k-clique counts. Since a k-color set is typically a good approximation for a k-clique in the dense regions of a graph, our algorithm is extremely efficient and accurate. In addition, we also propose a DP-based k-color path sampling technique to further improve the efficiency and accuracy. Here a k-color path is a connected k-color set which is more effective to approximate a k-clique. Moreover, unlike TuranShadow, both of our sampling-based solutions take near-linear space w.r.t. the graph size. In summary, we make the following contributions.

New algorithmic framework. We propose a new algorithmic framework for estimating k-clique counting which can circumvent the defects of the existing exact and approximation algorithms. We show that our framework is extremely efficient and accurate. It can achieve a 10^{-5} relative error by sampling a reasonable number of samples.

Two novel sampling algorithms. We develop two DP-based k-color set sampling techniques to estimate the number of k-cliques in the dense regions of the graph. Our novelty is in the algorithmic use of classic graph coloring technique for sampling. The striking features of our techniques are that they are not only very efficient and accurate, but also use near-linear space w.r.t. the graph size.

Extensive experiments. We evaluate our algorithms on 8 large real-life graphs. The results show that (1) our best algorithm is at least one order of magnitude faster than the state-of-the-art approximate algorithm (TuranShadow) to achieve a 0.1% relative error, using much smaller space; and (2) it can be up to three orders of magnitude faster than the state-of-the-art exact algorithm (PIVOTER) on large graphs. For example, on the hardest dataset LiveJournal with $k = 8$, TuranShadow takes more than 120 seconds and PIVOTER cannot terminate within 5 hours, while our best algorithm consumes around 20 seconds to achieve a 0.1% relative error. Moreover, our algorithms also exhibit an excellent parallel performance which can achieve $12\times \sim 14\times$ speedup ratios when using 16 threads in our experiments. For reproducibility purpose, the source code of this paper is released at https://github.com/LightWant/dpcolor.

2 PRELIMINARIES

Let $G = (V, E)$ be an undirected graph, where V and E denotes the set of nodes and edges respectively. Let n and m be the number of nodes and edges of G respectively. Denote by $N_v(G)$ the set of neighbors of v in G. The degree of v, denoted by $d_v(G)$, is the size of the neighbor set of v, i.e., $d_v(G) = |N_v(G)|$. Given a subset S of V, we denote by $G(S) = (V_S, E_S)$ the subgraph of G induced by S, where $E_S = \{(u, v) \in E | u, v \in S\}$. A k-clique is a complete subgraph of G in which every pair of nodes is connected by an edge.

Given a graph G and an integer k, the k-clique counting problem is to compute the number of k-cliques in G. Practical algorithms for solving the k-clique counting problem are often based on some ordering-based heuristic techniques [15, 23, 24, 28].

Let $\pi : V \rightarrow \{v_1, ..., v_n\}$ be a total order of the nodes in G. For two nodes u and v of G, we say that $\pi(v) < \pi(u)$ if u comes after v in the ordering of π. Then, based on such an ordering, we can obtain a DAG (directed acyclic graph) $\vec{G}$ by orienting the edges of the undirected graph G. Specifically, for each undirected edge (u, v) in G, we obtain a directed edge (u, v) in $\vec{G}$ if $\pi(u) < \pi(v)$, otherwise we get a directed edge (v, u). The k-clique counting problem in G is equivalent to computing the number of k-cliques in $\vec{G}$. Existing k-clique counting algorithms that work on the DAG $\vec{G}$ (instead of the original graph G) can guarantee that each k-clique is only explored once, thus significantly improving the efficiency.

Algorithm 1: The Proposed Framework

Input: A graph $G = (V, E)$, an integer k, and the sample size t
Output: The number of k-$cliques$ in G

1 $\vec{G} \leftarrow$ the DAG generated by the degeneracy ordering of G;
2 $ans \leftarrow 0$; $S \leftarrow \emptyset$;
3 **foreach** $v \in V$ **do**
4 **if** $\bar{d}(G(N_v(\vec{G}))) < k$ **then** $ans \leftarrow ans$ + PIVOTER($N_v(\vec{G})$, $k-1$);
5 **else** $S \leftarrow S \cup \{v\}$;
6 **return** ans + Sampling($\vec{G}$, S, k, t);

Note that many different ordering heuristics for k-clique counting have been developed in the literature [28]. Among them, a widely-used ordering heuristics is the degeneracy ordering [30], where the degeneracy is a metric to measure the sparsity of a graph [12]. Specifically, the degeneracy ordering of nodes in G is defined as an ordering $\{v_1, ..., v_n\}$ such that the degree of v_i is minimum in the subgraph of G induced by $\{v_i, ..., v_n\}$ for each v_i in G. We can make use of a classic peeling algorithm to generate the degeneracy ordering in $O(m + n)$ time [4]. Let δ be the degeneracy of G. Then, we can easily derive that $d_v(\vec{G}) \leq \delta$. Since δ is often very small in real-world graphs [12, 30], the degeneracy ordering based k-clique counting algorithms are often very efficient in practice [28]. In this work, we will also use the degeneracy ordering to design our algorithms.

3 THE PROPOSED FRAMEWORK

In this section, we propose a new algorithmic framework to estimate the number of k-cliques which combines both the exact PIVOTER algorithm and the sampling-based algorithms. The key idea of our framework is based on a simple but effective observation. The PIVOTER algorithm often works very efficient in the *sparse regions* of the graph, in which the number of k-cliques is typically not very large. However, in the *dense regions* of the graph, PIVOTER may be very costly to compute the k-clique counts, as the dense regions of the graph may contain a huge number of k-cliques. On the contrary, the sampling-based solutions are often very efficient and accurate to estimate the number of k-cliques in the dense regions of the graph, but they generally perform very bad in the sparse regions of the graph. This is because the k-cliques are relatively easier to be sampled in the dense regions, but they are often very hard to be drawn from the sparse regions of the graph. Therefore, to overcome the limitations of both the exact and sampling algorithms, we can apply the exact PIVOTER algorithm to calculate the k-clique counts in the sparse regions of the graph, and use the sampling-based techniques to estimate the number of k-cliques in the remaining dense regions of the graph. The details of our framework is shown in Algorithm 1.

Note that in Algorithm 1, we make use of the average degree of the nodes in the subgraph $C = (V_C, E_C)$ of G, denoted by $\bar{d}(V_C) = \sum_{v \in V_C} d_v(C)/|V_C|$, as an indicator to measure the sparsity of C. We refer to a subgraph C of G as a dense subgraph of G if $\bar{d}(V_C) \geq k$ (i.e., it lies in the dense regions of G), otherwise it is called a sparse subgraph. In Algorithm 1, it first computes a DAG $\vec{G}$ by the degeneracy ordering of G (line 1). Let $N_v(\vec{G})$ be the out-neighbors of a node v in $\vec{G}$, and $G(N_v(\vec{G}))$ be the subgraph induced by $N_v(\vec{G})$ in G. If the average degree of $G(N_v(\vec{G}))$ is smaller than k, the algorithm invokes PIVOTER to exactly compute the number of $(k-1)$-cliques contained in $N_v(\vec{G})$ (line 4). Otherwise, the subgraph $G(N_v(\vec{G}))$ is considered as a dense region of G, and the $(k-1)$-cliques contained in $N_v(\vec{G})$ are estimated by a sampling algorithm (lines 5-6).

The remaining question is how can we devise an efficient and effective sampling algorithm to estimate the number of k-cliques in the dense regions of G. Traditional edge sampling algorithms, such as [36, 43], are often inefficient, because those algorithms require a considerable number of samples to achieve a desired accuracy [23]. The color-coding based techniques often consume a significant number of space [2, 8, 9] and also they are less efficient than the TuranShadow algorithm [23]. The TuranShadow algorithm [23], which is the state-of-the-art sampling-based technique, also needs much space to store the Tuŕan Shadow. Moreover, the construction time of the Tuŕan Shadow is often very long for large graphs, because the worst-case time complexity of TuranShadow is exponential. In Sections 4 and 5, we will propose two novel sampling algorithms to tackle this problem.

Parallel implementation. Note that the proposed framework (Algorithm 1) can be easily parallelized, because the number of k-cliques in the subgraph induced by the out-neighbors for each node in $\vec{G}$ is independent. Specifically, in lines 3-5 of Algorithm 1, we can process the nodes in the sparse regions in parallel by independently invoking the PIVOTER algorithms. In the dense regions, the sampling-based techniques are also easily to be parallelized, because we can always draw t independent samples in parallel. In our experiments, we will show that our parallel implementations can achieve a near-linear speedup ratio on real-life graphs.

4 K-COLOR SET SAMPLING

In this section, we develop a novel sampling approach to estimate the k-clique counts in the dense regions of the graph, called k-color set sampling. Our technique is based on a concept of graph coloring [3, 21, 46]. Specifically, we first color the nodes in a graph such that each pair of adjacent nodes are colored with different colors. Let χ be the number of colors that are used to color all nodes in the graph G. The graph coloring procedure assigns an integer color value taking from $[1, \cdots, \chi]$ to each node in G, and no two adjacent nodes have the same color value. Note that since the minimum coloring problem (χ is minimum) is NP-hard [3], we use a linear-time greedy coloring algorithm [21, 46] to obtain a feasible coloring solution. Based on a feasible coloring solution, we define a concept called k-color set as follows.

Definition 4.1. A set of nodes V_k in the colored graph G is called a k-color set if it contains k nodes with k different colors.

Note that by Definition 4.1, the nodes of any k-clique must form a k-color set. In particular, we have the following lemma.

Lemma 4.2. *Given a graph G, all k-cliques must be contained in the set of all k-color sets.*

Let $\mathsf{cnt}_k(G, clique)$ and $\mathsf{cnt}_k(G, color)$ be the number of k-cliques and k-color sets of G respectively. Denoted by ρ_k the k-clique density of a graph G which is defined as the ratio between the number of k-cliques and the number of k-color sets of G, i.e., $\rho_k = \frac{\mathsf{cnt}_k(G,clique)}{\mathsf{cnt}_k(G,color)}$. Intuitively, in the dense regions of the graph G, a k-color set is *likely* to be a k-clique. Therefore, the k-clique density ρ_k

of the dense region of G is often not very small. As a consequence, an effective sampling technique to estimate the number of k-cliques can be obtained by estimating ρ_k.

There are two nontrivial problems needed to be tackled to develop such a sampling technique. First, we need to devise an efficient algorithm to compute the number of k-color sets. Second, to estimate ρ_k, we need to develop a uniform sampling mechanism to sample the k-color sets. Below, we will propose a dynamic programming algorithm to solve these issues.

4.1 DP-based k-color set sampling

Here we first propose a DP algorithm to compute the number of k-color sets. Then, we show how to use the DP algorithm to uniformly sample a k-color set.

Counting the number of k-color sets. Let χ be the number of colors of the graph G obtained by the greedy coloring algorithm [21, 46]. Denote by a_i the number of nodes in G with the color $i \in [1, \chi]$. Let G_i be the subgraph of G that only contains the nodes of G with color values no larger than i, i.e., $G_i = (V_i, E_i)$, where $V_i = \{v \in V | c(v) \leq i\}$, $E_i = \{(u, v) \in E | u, v \in V_i\}$, and $c(v)$ is the color value of v in G. Let $F(i, j)$ be the number of j-color sets in G_i. Then, we have the following recursive function for all $i, j \in [1, \chi]$.

$$F(i, j) = a_i \times F(i-1, j-1) + F(i-1, j). \tag{1}$$

The key idea of Eq. (1) is that the number of j-color sets in G_i can be derived by considering two cases: (1) the color i is included in the j-color sets; and (2) the color i is not included in the j-color sets. For the first case, the number of j-color sets in G_i is equal to a_i times the number of $(j-1)$-color sets in G_{i-1}, i.e., $a_i \times F(i-1, j-1)$. For the second case, the number of j-color sets is equal to the number of j-color sets in G_{i-1}, which is $F(i-1, j)$. Thus, the total number of j-color sets in G_i is the sum over these two cases. Clearly, the number of k-color sets in G is equal to the number of k-color sets in G_χ, i.e., $F(\chi, k)$. In addition, the initial states of $F(i, j)$ are as follows:

$$\begin{cases} F(i, 0) = 1, & \text{for all } i \in [0, \chi], \\ F(i, j) = 0, & \text{for all } i \in [0, \chi], j \in [i+1, \chi]. \end{cases} \tag{2}$$

Based on Eqs. (1) and (2), we can compute the number of k-color sets $F(\chi, k)$ in $O(k\chi)$ time by dynamic programming. The detailed implementation of the DP algorithm can be found in the DPCount procedure of Algorithm 2 (see lines 5-12).

From counting to uniformly sampling. Here we propose an efficient approach to uniformly sample a k-color set based on the k-color set counting technique. For convenience, we refer to a set of k different colors selected from $[1, \chi]$ as a k-color class. Clearly, in a graph G, a k-color class may contain a set of k-color sets.

To generate a uniform k-color set, a potential method is that we first sample a k-color class, and then we randomly select a node in G with color i for each i in the sampled k-color class. The challenge of this method is that how can we sample the k-color class to guarantee that the resulting k-color set is uniformly generated. Obviously, the straightforward method that uniformly picks k different colors from $[1, \chi]$ is incorrect in our case. This is because the numbers of k-color sets contained in various k-color classes are different. Thus, uniformly sampling a k-color class from $[1, \chi]$ will introduce biases for generating a uniform k-color set.

Algorithm 2: DPSampler (G, χ, k)

```
Input: A colored graph G = (V, E), an integer k, and the maximum color number χ
Output: A uniformly sampled k-color set
1  F ← DPCount(G, χ, k);
2  p_(i,j) ← a_i × F(i−1, j−1) / F(i, j) for all i ∈ [1, χ] and j ∈ [1, k];
3  R ← DPSampling(G, P, ∅, χ, k);
4  return R;
5  Procedure DPCount(G, χ, k)
6      Let a_i be the number of nodes with color i in G;
7      F(i, j) ← 0 for all i ∈ [0, χ] and j ∈ [i + 1, k];
8      foreach i = 0 to χ do F(i, 0) = 1;
9      foreach i = 1 to χ do
10         for j = 1 to k do
11             F(i, j) = a_i × F(i − 1, j − 1) + F(i − 1, j);
12     return F;
13 Procedure DPSampling(G, P, R, i, j)
14     if j = 0 then return R;
15     Sampling the color i with probability p_(i,j);
16     if the color i is sampled then
17         Randomly choose a node v in G with color i;
18         DPSampling (G, P, R ∪ {v}, i − 1, j − 1);
19     else DPSampling (G, P, R, i − 1, j) ;
```

To overcome this challenge, we propose a DP algorithm to sample a k-color class which can guarantee that the resulting k-color set is uniformly drawn. In particular, given a j-color class in G_i, it either (1) contains the color i, or (2) does not contain the color i. If the first case is true, the other $j-1$ colors of the j-color class are selected from $[1, i-1]$ in G_{i-1}. However, for the second case, the j-color class must be selected from $[1, i-1]$ in G_{i-1}. Therefore, we can sample a k-color class in G based on a similar DP equation as shown in Eq. (1). More specifically, to sample a j-color class, we define the probability of selecting the color i in G_i as

$$p_{(i,j)} = \frac{a_i \times F(i-1, j-1)}{F(i, j)}. \tag{3}$$

Clearly, the probability that does not choose the color i in G_i is $1 - p_{(i,j)} = F(i-1, j)/F(i, j)$. Based on Eq. (3), we can sample a j-color class using the following recursive sampling procedure. In each recursion, we pick a color i in G_i with the probability $p_{(i,j)}$. If the color i is sampled, we recursively sample the $(j-1)$-color class in G_{i-1}. Otherwise, we recursively sample the j-color class in G_{i-1}. After obtaining a k-color class, a k-color set is generated by randomly selecting a node with each color i in the k-color class. The detailed implementation of our algorithm for uniformly sampling a k-color set is shown in Algorithm 2.

Algorithm 2 first invokes the DP procedure to compute $F(i, j)$ for every $i \in [1, \chi]$ and $j \in [1, k]$ (line 1 and lines 5-12). Then, the algorithm computes the probability $p_{(i,j)}$ based on Eq. (3) (line 2). After that, the algorithm calls the recursively sampling procedure to uniformly generate a k-color set (line 3 and lines 13-19). The following results ensure the correctness of Algorithm 2.

Lemma 4.3. *The* DPSampling *procedure in Algorithm 2 outputs a k-color set of G if $\chi \geq k$.*

Theorem 4.4. *Algorithm 2 outputs a uniform k-color set.*

The following theorem shows the complexity of Algorithm 2.

Theorem 4.5. *Suppose that the graph G is colored and the nodes in each color group are obtained. Then, both the time and space complexity of Algorithm 2 are $O(\chi k)$.*

Algorithm 3: Estimating the number of k-cliques by k-color set sampling

Input: A graph $\vec{G}$, a node set S, an integer k, and the sample size t
Output: An estimation of the number of k-cliques

1 Coloring the graph $\vec{G}$ using a linear-time algorithm [21, 46];
2 Let χ be the number of colors obtained;
3 **foreach** $v \in S$ **do**
4 $\quad F_v \leftarrow \text{DPCount}(G(N_v(\vec{G})), \chi, k-1)$;
5 $cntKCol \leftarrow \sum_{v \in S} F_v(\chi, k-1)$;
6 Set the probability distribution D over the nodes in S where $p(v) = F_v(\chi, k-1)/cntKCol$ for each $v \in S$;
7 $successTimes \leftarrow 0$;
8 **for** $i = 1$ *to* t **do**
9 $\quad$ Independently sample a node v from D;
10 $\quad R \leftarrow \{v\} \cup \text{DPSampler}(G(N_v(\vec{G})), \chi, k-1)$;
11 $\quad$ **if** R *is a* k*-clique* **then**
12 $\quad\quad successTimes \leftarrow successTimes + 1$;
13 $\rho_k \leftarrow successTimes/t$;
14 **return** $\rho_k \times cntKCol$;

4.2 Estimating the number of k-cliques

By Theorem 4.4, we can first make use of Algorithm 2 to uniformly sample k-color sets from G, and then estimate the clique density ρ_k in the k-color sets of G. After that, the number of k-cliques in G can be estimated by $\rho_k \times F(\chi, k)$. Based on this idea, we propose a weighted sampling algorithm to estimate the number of cliques in the dense regions of G. The detailed implementation of our algorithm is shown in Algorithm 3.

Let S be a set of nodes whose neighborhood subgraphs are *dense regions* of G, i.e., $\bar{d}(G(N_v(\vec{G}))) \geq k$ for each $v \in S$. Algorithm 3 first colors the graph using a linear-time greedy algorithm [21, 46] (line 1). Then, the algorithm invokes the DPCount procedure to compute the number of k-color sets for each $v \in S$ (lines 3-4). Let $cntKCol$ be the total number of k-color sets (line 5). Then, we can obtain a probability distribution D over S where $p(v) = F_v(\chi, k-1)/cntKCol$ for each $v \in S$ (line 6). After that, Algorithm 3 draws t k-color sets by (1) sampling a node $v \in S$ with probability $p(v)$ (line 9), and (2) uniformly sampling a $(k-1)$-color set from $G(N_v(\vec{G}))$ (line 10). The algorithm computes the k-clique density ρ_k in the sampled k-color sets (lines 11-13), and then estimates the k-clique count as $\rho_k \times cntKCol$ (line 14). The following theorem shows that Algorithm 3 can obtain an unbiased estimator.

Theorem 4.6. *Algorithm 3 outputs an unbiased estimator for the number of k-cliques in the dense regions of G.*

By applying the classic Chernoff bound, we can easily derive that Algorithm 3 is able to produce a $1-\epsilon$ approximation of the k-clique count in the dense regions of the graph.

Theorem 4.7. *Algorithm 3 returns a $1-\epsilon$ approximation of the number of k-cliques in the dense regions of G with probability $1-2\sigma$ if $t \geq \frac{3}{\rho_k \epsilon^2} ln\frac{1}{\sigma}$, where ϵ and σ are small positive values and t is the sample size.*

Note that by Theorem 4.7, the sample size of our algorithm relies on the k-clique density ρ_k. Since ρ_k is often not very small in the dense regions of a graph, Algorithm 3 is expected to be very efficient in practice which is also confirmed in our experiments. Below, we analyze the time and space complexity of Algorithm 3.

Theorem 4.8. *Algorithm 3 consumes $O((|S|+t)\chi k + k^2 t + m + n)$ time and $O(m+n+\chi k)$ space.*

Remark. The proposed k-color set sampling algorithm is completely different from the traditional color coding technique [2, 8, 9] for k-clique counting. The color coding technique randomly assigns a *color* to each node (it is actually not a valid graph coloring), in which two adjacent nodes may have the same color. However, our k-color set based sampling algorithm is based on the graph coloring technique which requires two adjacent nodes having different colors. For the color coding technique, the probability of each k-clique being colored with k different colors is $\frac{k!}{k^k}$ [2]. With the increase of k, such a probability decreases dramatically. However, our technique can ensure that the k-clique of G is a k-color set no matter what k is. Moreover, unlike color coding, the probability of sampling k nodes with k different colors from G (the colored graph) is nonuniform in our algorithm.

5 CONNECTED k-COLOR SET SAMPLING

Recall that to achieve a $1-\epsilon$ approximation, the sample size of Algorithm 3 heavily relies on the k-clique density over the k-color sets, i.e., ρ_k (see Theorem 4.7). Although the dense regions of a graph G often have a relatively high ρ_k, it may still be very small in some cases as the k-color sets do not fully capture the clique property. To improve the effectiveness of the sampling algorithm, we propose a novel technique which can further boost the k-clique density by considering the connectivity of the k-color set.

For any k-color set in G, we can observe that it is definitely not a k-clique if the subgraph induced by the k-color set is not connected. Clearly, such disconnected k-color sets are unpromising samples for our sampling algorithm. Therefore, to improve the sampling performance, a natural question is that can we directly sample the connected k-color sets from G? In this section, we answer this question affirmatively by devising a novel k-color path sampling technique. The insight is that we only sample the k-color set in which there exists a simple path with length $k-1$ in the subgraph induced by the k-color set. For convenience, we refer to such a connected k-color set as a k-color path.

Similar to sampling k-color sets in G, we also need to uniformly sample the k-color paths. Unfortunately, the solutions proposed in Section 4 are no longer applicable for sampling k-color paths. Below, we develop a new DP-based sampling technique to uniformly generate the k-color paths.

5.1 DP-based k-color path sampling

Counting the number of k-color paths. We start by developing an algorithm to count the number of k-color paths in a graph G. We assume that the graph G is colored with the color values selected from $[1, \chi]$. Based on the color values, we can obtain a *color ordering* by sorting the nodes in a non-decreasing ordering of their color values. Note that we can use the nodes IDs to break ties to obtain a total ordering. It is worth mentioning that such a color ordering was used in the k-clique listing algorithms [28]. Clearly, we are able to construct a DAG $\vec{G}$ by the color ordering, where a directed edge $(u, v) \in \vec{G}$ is obtained by orienting the direction of $(u, v) \in G$ if v

comes after u in the color ordering. Based on the DAG $\vec{G}$, we can obtain the following results.

Theorem 5.1. *Let $\vec{G}$ be the DAG generated by the color ordering. Then, any $(k-1)$-path in $\vec{G}$ forms a k-color path.*

Theorem 5.2. *Let $\vec{G}$ be the DAG generated by the color ordering. Then, any k-clique $C = \{v_1, v_2, \cdots, v_k\}$ in G is a k-color path in $\vec{G}$.*

Note that a k-color path in $\vec{G}$ does not necessarily form a k-clique in G. However, the set of k-color paths is clearly a subset of the set of k-color sets. Thus, the k-clique density over the k-color paths, denoted by ρ_p, must be no smaller than the k-clique density over the k-color sets. To estimate the number of k-cliques in G, we need to compute ρ_p and the number of k-color paths as well. Below, we propose a DP algorithm to achieve this goal.

Let $\vec{G}_{v_i}$ be a subgraph of $\vec{G}$ induced by $\{v_i, ..., v_n\}$. Denote by $H(v_i, j)$ the number of j-paths containing the node v_i in $\vec{G}_{v_i}$. Clearly, each j-path containing v_i in $\vec{G}_{v_i}$ must start from v_i, since the node v_i in $\vec{G}_{v_i}$ only has out-neighbors. Thus, the total number of $(k-1)$-paths of $\vec{G}$, denoted by $\mathsf{cnt}_{k-1}(\vec{G}, path)$, can be computed by the following formula:

$$\mathsf{cnt}_{k-1}(\vec{G}, path) = \sum_{v_i \in \vec{G}} H(v_i, k-1). \tag{4}$$

Observe that the second node in each $(k-1)$-path containing v_i in $\vec{G}_{v_i}$ must be an out-neighbor of v_i. Thus, if we have the count of the $(j-2)$-paths containing v_x in $\vec{G}_{v_x}$ for each $v_x \in N_{v_i}(\vec{G}_{v_i})$, the count of $(j-1)$-paths containing v_i in G_{v_i} can be easily obtained. Specifically, we have the following recursive equation:

$$H(v_i, j) = \sum_{v_x \in N_{v_i}(\vec{G}_{v_i})} H(v_x, j-1). \tag{5}$$

Initially, we have

$$\begin{cases} H(v_i, 0) = 1, & \text{for all } i \in [1, n], \\ H(v_i, j) = 0, & \text{for all } i \in [1, n], j \in [1, k-1]. \end{cases} \tag{6}$$

Based on Eqs. (4), (5) and (6), we can easily devise a DP algorithm to compute $\mathsf{cnt}_{k-1}(\vec{G}, path)$ which is detailed in the DPPathCount procedure of Algorithm 4 (lines 5-12). It is easy to derive that the time complexity of DPPathCount is $O(kn\chi)$, where χ is the maximum color value of G. This is because the cardinality of the out-neighbors for any node in $\vec{G}$ is bounded by $O(\chi)$.

Sampling a uniform k-color path. Similar to the DP-based sampling technique developed in Section 4.1, here we also propose a DP-based sampling algorithm to uniformly sample the k-color paths. Suppose without loss of generality that there is a randomly sampled k-color path of $\vec{G}$ starting from a node v, denoted by P_v. Then, for the second node in P_v, it must be an out-neighbor of v in $\vec{G}$. According to the DP equation (Eq. (5)), the number of $(k-1)$-paths starting from v is equal to the sum of the number of $(k-2)$-paths starting from each node in $N_v(\vec{G})$. Therefore, the next node of a random k-color path starting from v, denoted by u, should be drawn from $N_v(\vec{G})$ with probability $\frac{H(u,k-2)}{H(v,k-1)}$ by Eq. (5). We can recursively perform this sampling procedure to obtain a k-color path. The detailed implementation of this sampling technique is shown in Algorithm 4.

Algorithm 4: DPPathSampler (G, k)

```
Input: A colored graph G = (V, E), and an integer k
Output: A uniformly sampled k-color path
1  G⃗ ← the DAG generated by the color ordering of G;
2  H ← DPPathCount(G⃗, k);
3  R ← DPPathSampling(G⃗, H, k);
4  return R;
5  Procedure DPPathCount(G⃗, k)
6      H(v_i, j) ← 0, for i ∈ [1, n] and j ∈ [1, k − 1];
7      foreach i = 0 to n do H(v_i, 0) = 1;
8      foreach j = 1 to k − 1 do
9          for i = 1 to n do
10             for v_x ∈ N_{v_i}(G⃗) do
11                 H(v_i, j) ← H(v_i, j) + H(v_x, j − 1);
12     return H;
13 Procedure DPPathSampling(G⃗, H, k)
14     R ← ∅; Q ← V;
15     for i = 0 to k − 1 do
16         cnt ← Σ_{u∈Q} H(u, k − i − 1) ;
17         Set the probability distribution D over the nodes in Q where
             p(u) = H(u, k − i − 1)/cnt for each u ∈ Q;
18         Sample a node u from D;
19         R ← R ∪ {u}; Q ← N_u(G⃗);
20     return R;
```

Algorithm 4 first constructs a DAG $\vec{G}$ by the color ordering (line 1). Then, the algorithm invokes DPPathCount to derive the DP table H (line 2). After that, Algorithm 4 calls the DPPathSampling procedure to uniformly sample a k-color path (line 3). Specifically, when sampling a node u from $N_v(\vec{G})$, DPPathSampling needs to set a probability distribution D over the set $N_v(\vec{G})$ based on Eq. (5) (lines 16-18). After choosing a node u, DPPathSampling turns to sample the next node from $N_u(\vec{G})$ (line 19). The DPPathSampling procedure terminate when k nodes are sampled.

It is important to note that Algorithm 4 can always obtain a k-color path if the DAG $\vec{G}$ contains at least one k-color path. This is because in lines 16-18, if a node u is sampled, then $H(u, k-i-1)$ must be larger than 0, indicating that the out-neighborhood $N_u(\vec{G})$ must be non-empty. As a consequence, if there is a k-color path in $\vec{G}$, the **for** loop in line 15 of Algorithm 4 will be executed k times which results in a k-color path. The following theorem shows that Algorithm 4 can obtain a uniform k-color path.

Theorem 5.3. *Algorithm 4 outputs a uniform k-color path.*

We analyze the time and space complexity of Algorithm 4 in the following theorem.

Theorem 5.4. *Given an input graph G with n nodes and m edges, Algorithm 4 takes $O(\chi nk + m)$ time and uses $O(kn + m)$ space, where χ is the maximum color value.*

5.2 Estimating the k-clique counts

Based on Algorithm 4, we can devise a weighted sampling algorithm to construct an unbiased estimator to compute the number of k-cliques. Specifically, we can slightly modify Algorithm 3 by (1) replacing the DPCount procedure in line 4 of Algorithm 3 with the DPPathCount procedure, and (2) replacing DPSampling in line 10 of Algorithm 3 with DPPathSampling. Due to the space limit, we omit the details of this modified algorithm. Similar to Theorems 4.6 and 4.7, the estimator based on the k-color path sampling is also

Table 1: Datasets

Networks	n	m	δ
webStanford	281,903	1,992,636	71
DBLP	425,957	1,049,866	113
webBerkStan	685,230	6,649,470	201
webGoogle	916,428	4,322,051	44
Skitter	1,696,415	11,095,298	111
Orkut	3,072,627	117,185,083	253
LiveJournal	4,036,538	34,681,189	360
Friendster	65,608,366	1,806,067,135	304

unbiased, and the sample size can also be bounded by using the Chernoff bound. Moreover, it is easy to check that the sample size is no larger than that of Algorithm 3, because $\rho_p \geq \rho_k$.

For the time complexity, such a modified algorithm takes $O(|S|\delta^2 k)$ to compute the DP tables (i.e., H) for all nodes in S (because the input graph $G(N_v(\vec{G}))$ for the DPPathCount procedure has at most δ nodes), and consumes $O(\delta k + k^2)$ to sample a k-color path. Thus, the total time complexity of the algorithm is $O(|S|\delta^2 k + (\delta k + k^2)t + m + n)$, where $O(m+n)$ is taken for computing the graph coloring. The space overhead of the modified algorithm is $O(m + n + \delta k)$, because the DP table takes $O(\delta k)$ space.

6 EXPERIMENTS

6.1 Experimental setup

We compare the proposed algorithms with three state-of-the-art k-clique counting algorithms which are kClist [15, 28], PIVOTER [24], TuranShadow [23]. The kClist algorithm is an exact k-clique counting algorithm which is based on k-clique enumeration [15]. Note that the original kClist algorithm is based on the degeneracy ordering. Li et al. [28] proposed an improved version based on a hybrid of the degeneracy and color ordering. In our experiment, kClist denotes such an improved version. PIVOTER and TuranShadow are the state-of-the-art exact and approximate k-clique counting algorithms respectively. Both PIVOTER and TuranShadow were proposed by Jain and Seshadhri [23, 24]. PEANUTS [25] is an improved version of TuranShadow by the technique of Prefixed-Shadow, thus we use PEANUTS as the baseline instead of TuranShadow. The C++ codes of all these three algorithms are publicly available, thus we use their implementations in our experiments. For our algorithms, we implement DPColor and DPColorPath. DPColor denotes Algorithm 1 integrated with the k-color set sampling algorithm (Algorithm 2 and Algorithm 3), while DPColorPath is Algorithm 1 equipped with the k-color path sampling technique (Algorithm 4). Both DPColor and DPColorPath are implemented in C++. All algorithms are evaluated on a PC with two 2.1 GHz Xeon CPUs (16 cores in total) and 128GB memory running CentOS 7.6.

Datasets. We use 8 large real-life datasets in our experiments. Table 1 summarizes the detailed statistic information of all datasets. The last column of Table 1 denotes the degeneracy of the graph. The webStanford, webBerkStan, and webGoogle datasets are web graphs. DBLP is a co-authorship network, and Skitter is an internet graph. Orkut, LiveJournal, and Friendster are social networks. All datasets are downloaded from (snap.stanford.edu).

6.2 Experimental results

Runtime of different algorithms. In this experiment, we compare the running time of different algorithms on all datasets. Note that for each approximation algorithm (PEANUTS, DPColor, and DPColorPath), we record its running time when the algorithm achieves a 0.1% relative error. Here the relative error is computed by $|f - \hat{f}|/f$, in which f is the exact k-clique count and $\hat{f}$ is the estimated count. For all algorithms, if they cannot terminate within 5 hours, we set their running time to "INF". Fig. 1 shows the running time of different algorithms with varying k.

We first compare our algorithms with kClist and PIVOTER. As can be seen, both DPColor and DPColorPath are significantly faster than kClist and PIVOTER on most datasets with varying k. The kClist algorithm is generally intractable for large k on all datasets. On most datasets, DPColorPath is around one order of magnitude faster than PIVOTER. The hardest instance is the LiveJournal graph, on which PIVOTER only obtains the number of 4-cliques within 5 hours, whereas DPColorPath takes around 20 seconds to achieve a 0.1% relative error (DPColorPath can achieve at least three orders of magnitude faster than PIVOTER on LiveJournal). Note that since both kClist and PIVOTER are intractable on LiveJournal when $k \geq 6$, we use the exact k-clique count obtained from [1], where $k \leq 8$, to compute the relative errors for the approximation algorithms. Moreover, as reported in [1], the running time of such a GPU-parallelized PIVOTER algorithm using 5120 CUDA Cores is 6,851 seconds for $k = 8$, while our *sequential* DPColorPath (DPColor) take around 20 seconds to obtain a very accurate k-clique count. These results indicate that our algorithms are extremely efficient for k-clique counting.

By comparing our algorithms with PEANUTS, we can see that both DPColor and DPColorPath are consistently faster than PEANUTS on all datasets with varying k. On most datasets, DPColorPath is orders of magnitude faster than PEANUTS. For example, on DBLP, both DPColor and DPColorPath take around 0.1 second, while PEANUTS consumes more than 1 seconds for most k values. In addition, on Orkut and Friendster, PEANUTS and DPColor cannot achieve a desired relative error within 5 hours for large k values, while DPColorPath is still very efficient on these two datasets. For our algorithms, DPColorPath is generally faster than DPColor. Moreover, the performance of DPColorPath is much more stable than DPColor on all datasets. These results confirm our theoretic analysis in Sections 4 and 5.

Relative errors with varying sample size. Fig. 2 shows the relative errors of three algorithms with varying k on webStanford and LiveJournal. Similar results can also be observed on the other datasets. As shown in Fig. 2, the relative error of DPColorPath is consistently lower than those of DPColor and PEANUTS with the same sample size. In general, the relative errors of all algorithms decrease with the sample size increases. Moreover, we can see that DPColorPath obtain a 10^{-5} relative error on all datasets when the sample size is 10^8, indicating that DPColorPath can achieve very high accuracy using a reasonable number of samples. These results further confirm the efficiency and effectiveness of our techniques.

K-clique density. In this experiment, we evaluate the k-clique densities over the k-color sets (ρ_k) and the k-color paths (ρ_p) in the dense regions of the graph, respectively. The results on all datasets are reported in Table 2. As expected, ρ_p is no larger than ρ_k on all datasets. Moreover, both ρ_k and ρ_p can achieve a very high value on most datasets. For example, on DBLP and webBerkStan, both

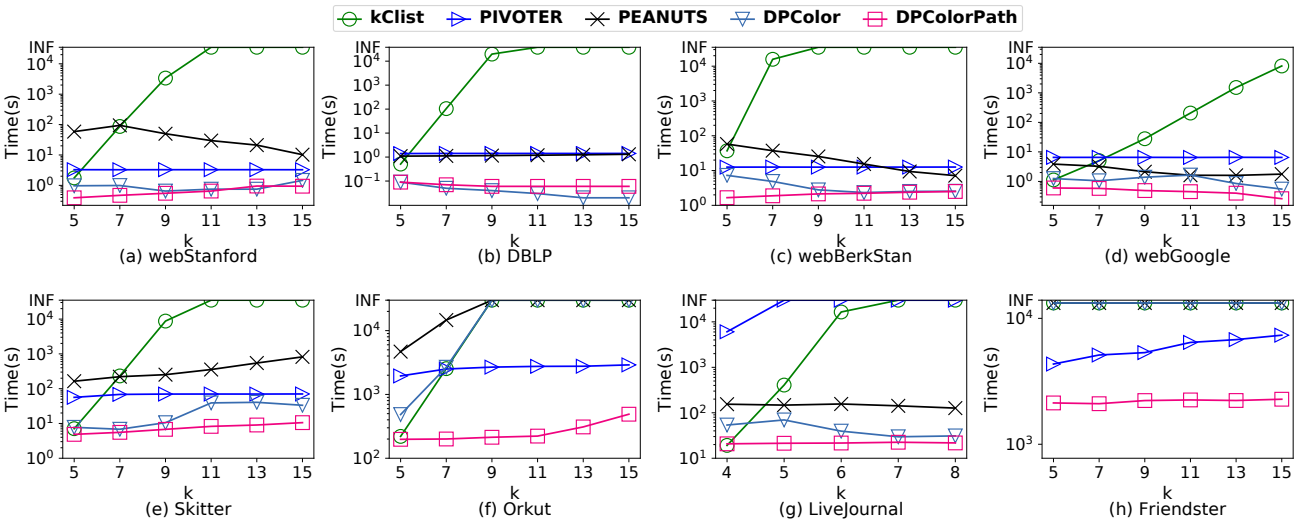


Figure 1: Running time of different algorithms (the relative errors for PEANUTS, DPColor, DPColorPath are set to 0.1%)

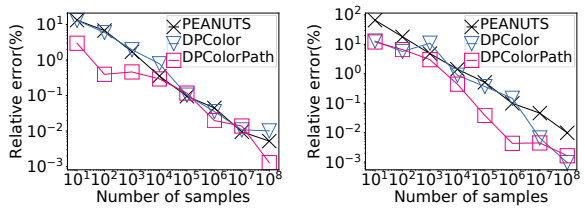


Figure 2: Relative errors with varying sample size ($k = 8$)

Table 2: The *k-clique* densities (ρ_k/ρ_p) in the dense regions (%)

Networks	$k = 3$	$k = 8$	$k = 12$	$k = 15$
webStanford	83.0/100.0	70.8/77.7	54.3/61.2	45.8/53.1
DBLP	97.2/100.0	100.0/100.0	100.0/100.0	100.0/100.0
webBerkStan	94.7/100.0	99.9/100.0	100.0/100.0	100.0/100.0
webGoogle	94.6/100.0	91.2/94.4	86.1/87.8	84.7/85.7
Skitter	42.3/100.0	5.3/26.4	0.7/6.4	0.1/2.3
Orkut	20.5/100.0	0.0/2.6	0.0/0.2	0.0/0.0002
LiveJournal	54.3/100.0	80.4/91.0	-/-	-/-
Friendster	10.1/100.0	0.0/18.1	0.0/55.6	0.0/52.2

ρ_k and ρ_p are near to 100%. In general, both ρ_k and ρ_p decreases with k increases. Nevertheless, on most datasets, ρ_p is always very large even when $k = 15$. These results further confirm that the proposed techniques can achieve high accuracy on real-life graphs.

Memory overheads. Fig. 3 shows the memory usages of various algorithms on webBerkStan and LiveJournal for $k = 8$. The results for the other k values and datasets are consistent. As expected, the space consumption of PEANUTS is significantly higher than the other algorithms, as it needs to store the Tuŕan Shadow structure. The space overheads of our algorithms and PIVOTER are comparable, while kClist consumes slightly more space than our algorithms. These results demonstrate that our algorithms are space efficient.

Parallel performance of our algorithms. In this experiment, we evaluate the parallel performance of our algorithms. To this end, we implement the parallel versions for both DPColor and DPColorPath

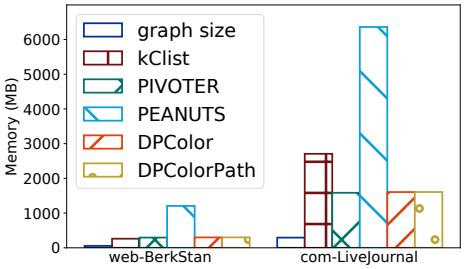


Figure 3: Memory usage of various algorithms ($k = 8$)

Table 3: Runtime of our parallel algorithms ($k = 8$, $t = 5\times10^6$)

Threads	DPColor (sec.)		DPColorPath (sec.)	
	LiveJournal	Friendster	LiveJournal	Friendster
1	24.8	2481.5	28.4	2132.2
4	7.1	650.3	7.5	559.6
8	4.3	341.6	3.9	293.3
12	2.7	244.4	2.7	210.3
16	2.1	196.5	2.1	171.9

using OpenMP. We fix the sample size as 5×10^6 to evaluate the runtime of DPColor and DPColorPath on the two largest datasets. The results are shown in Table 3. As can be seen, both DPColor and DPColorPath perform very well. Both DPColor and DPColorPath can achieve $12\times \sim 14\times$ speedups when using 16 threads. This result indicates a high degree of parallelism of our algorithms.

7 CONCLUSION

In this paper, we propose a time and space efficient framework for k-clique counting. Our framework first divides the graph into sparse and dense regions based on the average degree. Then, for the sparse regions, we use the state-of-the-art PIVOTER algorithm to compute the exact number of k-cliques. For the dense regions, we develop two novel DP-based k-color set and k-color path sampling techniques to estimate the k-clique count, respectively. Extensive experiments on 8 real-life graphs show that our algorithms are very efficient and accurate.

ACKNOWLEDGMENTS

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034, U1809206, and 61772346. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-Mei W. Hwu. 2021. K-Clique Counting on GPUs. *CoRR* abs/2104.13209 (2021).
[2] Noga Alon, Raphael Yuster, and Uri Zwick. 1994. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *STOC*.
[3] Balabhaskar Balasundaram and Sergiy Butenko. 2006. Graph Domination, Coloring and Cliques in Telecommunications. In *Handbook of Optimization in Telecommunications*. Springer, 865–890.
[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (2003).
[5] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2008. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*.
[6] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016).
[7] J. W. Berry, B. Hendrickson, R. A. Laviolette, and C. A. Phillips. 2011. Tolerating the Community Detection Resolution Limit with Edge Weighting. *Physical Review E Statistical Nonlinear & Soft Matter Physics* 83, 5 (2011), 056119.
[8] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discov. Data* 12, 4 (2018), 48:1–48:25.
[9] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: Fast Motif Counting via Succinct Color Coding and Adaptive Sampling. *Proc. VLDB Endow.* 12, 11 (2019), 1651–1663.
[10] S Ronald Burt. 2004. Structural holes and good ideas. *Amer. J. Sociology* 110, 2 (2004), 349–399.
[11] Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. 2021. Tight Distributed Listing of Cliques. In *SODA*.
[12] Lijun Chang and Lu Qin. 2019. Cohesive Subgraph Computation Over Large Sparse Graphs. In *ICDE*.
[13] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.
[14] Shumo Chu and James Cheng. 2011. Triangle listing in massive networks and its applications. In *KDD*.
[15] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in Sparse Real-World Graphs. In *WWW*.
[16] Talya Eden, Dana Ron, and C. Seshadhri. 2018. On approximating the number of k-cliques in sublinear time. In *STOC*.
[17] Talya Eden, Dana Ron, and C. Seshadhri. 2020. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *SODA*.
[18] Katherine Faust. 2010. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Soc. Networks* 32, 3 (2010), 221–233.
[19] Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. 2015. Clique Counting in MapReduce: Algorithms and Experiments. *ACM J. Exp. Algorithmics* 20 (2015), 1.7:1–1.7:20.
[20] Lukas Gianinazzi, Maciej Besta, Yannick Schaffner, and Torsten Hoefler. 2021. Parallel Algorithms for Finding Large Cliques in Sparse Graphs. In *SPAA*.
[21] William Hasenplaugh, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2014. Ordering heuristics for parallel graph coloring. In *SPAA*.
[22] Lin Hu, Lei Zou, and Yu Liu. 2021. Accelerating Triangle Counting on GPU. In *SIGMOD*.
[23] Shweta Jain and C. Seshadhri. 2017. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *WWW*.
[24] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *WSDM*.
[25] Shweta Jain and C. Seshadhri. 2020. Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. 1966–1976.
[26] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *WWW*.
[27] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* 407, 1-3 (2008), 458–473.
[28] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *Proc. VLDB Endow.* 13, 11 (2020), 2536–2548.
[29] Kazuhisa Makino and Takeaki Uno. 2004. New Algorithms for Enumerating All Maximal Cliques. In *9th Scandinavian Workshop on Algorithm Theory*.
[30] David W. Matula and Leland L. Beck. 1983. Smallest-Last Ordering and clustering and Graph Coloring Algorithms. *J. ACM* 30, 3 (1983), 417–427.
[31] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D Chklovskii, and U. Alon. 2010. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (2010), 763–764.
[32] Mark Ortmann and Ulrik Brandes. 2014. Triangle Listing Algorithms: Back from the Diversion. In *ALENEX*.
[33] Noujan Pashanasangi and C. Seshadhri. 2020. Efficiently Counting Vertex Orbits of All 5-vertex Subgraphs, by EVOKE. In *WSDM*.
[34] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *WWW*.
[35] Natasa Przulj, Derek G. Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinform.* 20, 18 (2004), 3508–3515.
[36] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An Efficient Graphlet Counting Method for Large Graph Analysis. *IEEE Trans. Knowl. Data Eng.* 26, 10 (2014), 2466–2478.
[37] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2015. Finding the Hierarchy of Dense Subgraphs using Nucleus Decompositions. In *WWW*.
[38] Comandur Seshadhri and Srikanta Tirthapura. 2019. Scalable Subgraph Counting: The Methods Behind The Madness. In *WWW*.
[39] Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proc. VLDB Endow.* 13, 10 (2020), 1628–1640.
[40] Ancy Sarah Tom, Narayanan Sundaram, Nesreen K. Ahmed, Shaden Smith, Stijn Eyerman, Midhunchandra Kodiyath, Ibrahim Hur, Fabrizio Petrini, and George Karypis. 2017. Exploring optimizations on shared-memory platforms for parallel triangle counting algorithms. In *HPEC*.
[41] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.* 363, 1 (2006), 28–42.
[42] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *WWW*.
[43] Charalampos E. Tsourakakis, U Kang, Gary L. Miller, and Christos Faloutsos. 2009. DOULION: counting triangles in massive graphs with a coin. In *KDD*.
[44] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A Fast Method of Approximating Counts of 5-Node Graphlets in Large Graphs. *IEEE Trans. Knowl. Data Eng.* 30, 1 (2018), 73–86.
[45] Hao Yin, Austin R. Benson, and Jure Leskovec. 2017. Higher-order clustering in networks. *Physical Review E* 97, 5 (2017), 052306.
[46] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. Effective and Efficient Dynamic Graph Coloring. *Proc. VLDB Endow.* 11, 3 (2017), 338–351.

Table 4: The ratio of the k-cliques in the sparse regions.

Networks	$k = 3$	$k = 8$	$k = 12$	$k = 15$
webStanford	6.15%	0.01%	0.00%	0.00%
DBLP	33.11%	0.00%	0.00%	0.00%
webBerkStan	3.80%	0.00%	0.00%	0.00%
webGoogle	11.63%	6.47%	0.69%	0.36%
Skitter	19.57%	0.03%	0.00%	0.00%
Orkut	6.47%	0.07%	0.00%	0.00%
LiveJournal	9.30%	0.00%	0.00%	0.00%
Friendster	26.15%	0.30%	0.01%	0.00%

8 SUPPLEMENTARY MATERIALS

8.1 Additional experiments

The number of k-cliques in the sparse regions. In this experiment, we evaluate the number of k-cliques in the sparse regions of a graph on all datasets. Note that a node's neighborhood-induced subgraph is called a *sparse region* of a graph if the average degree of such a subgraph is smaller than k. Clearly, if the sparse regions have less number of k-cliques, then the PIVOTER algorithm should be more efficient. Table 4 reports our results on all datasets. As can be seen, for a relatively large k, the number of k-cliques in the sparse regions of all datasets only accounts for a small portion of the total number of k-cliques. On most datasets, such a ratio usually does not exceed 0.1%. These results indicate that the proposed framework, which integrates both PIVOTER and sampling techniques, can be very efficient for handling real-life graphs.

Trade-off between accuracy and sample size. Table 5 shows the sample sizes needed by TuranShadow, DPColor and DPColorPath to meet a desired relative error given that $k = 8$. The results are consistent when setting k to other values. As expected, the sample size required by different algorithms increases with the relative error decreases. Both DPColor and DPColorPath requires less samples than TuranShadow to achieve a desired relative errors on most datasets. Moreover, we can see that in most settings, the sample size of DPColorPath is one order of magnitude less than those of TuranShadow and DPColor. For example, to achieve a 0.01% relative error on LiveJournal, DPColorPath only needs 2×10^5 samples, whereas TuranShadow and DPColor require 3×10^7 and 7×10^6 samples respectively. These results further confirm the superiority of the proposed k-color path sampling technique.

8.2 Missing proofs

The proof of lemma 4.3. On the one hand, it is easy to verify that there are at most k colors outputted by the DPSampling procedure, since $p_{(i,0)} = 0$ and DPSampling will terminate immediately when $j = 0$. On the other hand, by Eq. (3), we can derive that $p_{(i,i)} = 1$. This is because $F(i-1, i) = 0$ by definition, thus $1 - p_{(i,i)} = F(i-1, i)/F(i, i) = 0$. As a result, the probability of sampling a color i with $p_{(i,i)}$ is always 1, thus there are at least k colors that are sampled by DPSampling if $\chi \geq k$. Putting it all together, the lemma is established. □

The proof of Theorem 4.4. Let X be the event of a random k-color class of G sampled by DPSampling. For each color j from 1 to χ, let Y_j be an indicator random variable, which is equal to 1 if the color j is selected in the event X, otherwise it is equal to 0. Let $\Pr(X)$ be the occurrence probability of the event X. Then, we have the following equation:

$$\Pr(X) = \Pr((\sum_{i=1}^{\chi} Y_i) = k). \tag{7}$$

Recall that DPSampling draws k colors following the decreasing order of the color values (i.e., from χ to 1). For each color $j \in [1, \chi]$, the probability of selecting the color j in G_i is $p_{(i,j)}$. Assume that the sampled k-color class of G is $C = \{c_1, ..., c_k\}$, where each c_i is a color value of G and $c_1 > c_2 > \cdots > c_k$. Clearly, a k-color class C partitions the interval $[1, \chi]$ into at most $2k+1$ sub-intervals as $\{[c_1+1, \chi], [c_1, c_1], [c_2+1, c_1-1], \cdots, [c_k+1, c_{k-1}-1], [c_k, c_k], [1, c_k-1]\}$. Note that DPSampling only selects a color in the sub-intervals $[c_i, c_i]$ for every $i = 1, \cdots, k$, and no color is selected in the other sub-intervals. Therefore, the probability of $\Pr((\sum_{i=1}^{\chi} Y_i) = k)$ can be computed by

$$\begin{aligned}
&\frac{F(\chi-1,k)}{F(\chi,k)} \times \frac{F(\chi-2,k)}{F(\chi-1,k)} \times \cdots \times \frac{F(c_1,k)}{F(c_1+1,k)} \times \frac{a_{c_1} \times F(c_1-1,k-1)}{F(c_1,k)} \\
&\times \frac{F(c_1-2,k-1)}{F(c_1-1,k-1)} \times \cdots \times \frac{F(c_2,k-1)}{F(c_2+1,k-1)} \times \frac{a_{c_2} \times F(c_2-1,k-2)}{F(c_2,k-1)} \\
&\times \cdots \times \frac{F(c_k,1)}{F(c_k+1,1)} \times \frac{a_{c_k} \times F(c_k-1,0)}{F(c_k,1)} \\
&= \frac{a_{c_1} \times a_{c_2} \times \cdots \times a_{c_k}}{F(\chi,k)}.
\end{aligned} \tag{8}$$

After obtaining a k-color class C, the algorithm further samples k nodes with k different colors in C from G. Let $\Pr(k\text{-color set})$ be the probability of sampling a k-color set from G. Then, we have

$$\begin{aligned}
\Pr(k\text{-color set}) &= \Pr(k \text{ nodes with different colors}|X) \times \Pr(X) \\
&= \frac{1}{a_{c_1} \times a_{c_2} \times \cdots \times a_{c_k}} \times \frac{a_{c_1} \times a_{c_2} \times \cdots \times a_{c_k}}{F(\chi,k)} \\
&= \frac{1}{F(\chi,k)} = \frac{1}{\mathrm{cnt}_k(G, color)}
\end{aligned} \tag{9}$$

By Eq. (9), each k-color set is uniformly sampled, thus the theorem is established. □

The proof of Theorem 4.5. Clearly, the time complexity of the DP procedure for counting the number of k-color sets is $O(\chi k)$. In the DPSampling procedure, we can randomly choose a node with color i in constant time if the color groups are obtained (line 17). The total time costs of the DPSampling procedure are bounded by $O(\chi + k)$. As a result, the time complexity of Algorithm 2 is $O(\chi k)$. For the space complexity, Algorithm 2 only requires $O(\chi k)$ additional space to store the DP table F and the probabilities p. □

The proof of Theorem 4.6. Let $X_i = 1$ if the i_{th} sampled k-color set is a k-clique, otherwise $X_i = 0$. Observe that

$$\begin{aligned}
\Pr(X_i = 1) = \sum_{v \in S} [&\Pr(choose\ v\ from\ D) \\
&\times \Pr(choose\ a\ clique\ from\ G(N_v(\vec{G})))].
\end{aligned} \tag{10}$$

In the summation, the former probability is $\frac{F_v(\chi,k-1)}{\sum_{v \in S} cnt_k(G(N_v(\vec{G})), color)}$, and the latter is exactly $\frac{cnt_k(G(N_v(\vec{G})), clique)}{F_v(\chi,k-1)}$. Thus, $Pr(X_i = 1) =$

Table 5: Sample sizes vs. relative errors of TuranShadow, DPColor **and** DPColorPath (k = **8**)

Networks	ϵ = 1%			ϵ = 0.1%			ϵ = 0.01%		
	TuranShadow	DPColor	DPColorPath	TuranShadow	DPColor	DPColorPath	TuranShadow	DPColor	DPColorPath
webStanford	$5e10^3$	$1e10^4$	$2e10^3$	$1e10^5$	$1e10^5$	$7e10^4$	$1e10^8$	$5e10^6$	$1e10^6$
DBLP	$1e10^1$	$1e10^1$	$1e10^1$	$1e10^1$	$1e10^1$	$1e10^1$	$5e10^3$	$1e10^1$	$1e10^1$
webBerkStan	$1e10^3$	$1e10^1$	$1e10^1$	$1e10^3$	$1e10^1$	$1e10^1$	$5e10^5$	$5e10^5$	$2e10^5$
webGoogle	$1e10^4$	$1e10^3$	$1e10^2$	$1e10^5$	$3e10^4$	$2e10^4$	$4e10^7$	$4e10^5$	$8e10^5$
Skitter	$1e10^3$	$3e10^5$	$1e10^4$	$1e10^6$	$1e10^7$	$5e10^5$	$2e10^7$	$5e10^7$	$4e10^6$
Orkut	$1e10^6$	$>1e10^8$	$3e10^6$	$5e10^6$	$>1e10^8$	$3e10^7$	$1e10^7$	$>1e10^8$	$8e10^7$
LiveJournal	$1e10^4$	$1e10^4$	$3e10^3$	$1e10^5$	$5e10^6$	$1e10^4$	$3e10^7$	$7e10^6$	$2e10^5$
Friendster	–	$>1e10^8$	$5e10^6$	–	$>1e10^8$	$1e10^7$	–	$>1e10^8$	$1e10^7$

$\frac{\sum_{v\in S} cnt_k(G(N_v(\vec{G})),clique)}{\sum_{v\in S} cnt_k(G(N_v(\vec{G})),color)}$. This implies that the probability of sampling a k-clique is exactly the k-clique density in the dense regions. By the linearity of expectation, we have

$$E[cntKCol \times \frac{\sum_{i\le t} X_i}{t}] = \sum_{v\in S} cnt_k(G(N_v(\vec{G})), color) \times \frac{\sum_{i\le t} E[X_i]}{t} = \sum_{v\in S} cnt_k(G(N_v(\vec{G})), clique). \tag{11}$$

Therefore, Algorithm 3 returns an unbiased estimator of the k-clique count in the dense regions of G. □

The proof of Theorem 4.7. Denote by $\hat{\rho_k}$ the estimator of the k-clique density (line 13 of Algorithm 3). Since our estimator is unbiased, we have $E[\hat{\rho_k}] = \rho_k$. Then, the expected number of k-cliques in the t samples is $E[\hat{\rho_k}t] = \rho_k t$. Based on the Chernoff bound, we easily obtain the following results:

$$\Pr(\hat{\rho_k}t \le (1-\epsilon)\rho_k t) \le \exp(-\frac{\epsilon^2\rho_k t}{2}) \le \exp(-\frac{\epsilon^2\rho_k t}{3}), \tag{12}$$

$$\Pr(\hat{\rho_k}t \ge (1+\epsilon)\rho_k t) \le \exp(-\frac{\epsilon^2\rho_k t}{3}). \tag{13}$$

Further, we have:

$$\Pr(\frac{|\hat{\rho_k}-\rho_k|}{\rho_k} \ge \epsilon) \le 2\exp(-\frac{\epsilon^2\rho_k t}{3}). \tag{14}$$

Let $\exp(-\frac{\epsilon^2\rho_k t}{3}) \le \sigma$. Then, we can derive that $t \ge \frac{3}{\rho_k\epsilon^2}\ln\frac{1}{\sigma}$. This completes the proof. □

The proof of Theorem 4.8. For the time complexity, Algorithm 3 takes $O(m+n)$ time to obtain a feasible graph coloring. Then, it consumes $O(|S|\chi k)$ time to compute F_v for each $v \in S$. After that, to draw a k-color set, the algorithm takes $O(\chi k)$ time and $O(k^2)$ time to check whether it is a clique. Thus, the total time used in the k-color set sampling stage is $O(t(\chi k + k^2))$. As a consequence, the time complexity of Algorithm 3 is $O((|S|+t)\chi k + m + n + k^2 t)$. For the space complexity, the algorithm needs to store the graph G and the colors which takes $O(m+n)$ space in total. Additionally, the algorithm uses $O(\chi k)$ space to store the DP table when sampling a k-color set. Note that the algorithm does not store all the DP tables for all samples. Thus, the total space overhead of Algorithm 3 is $O(m+n+\chi k)$. □

The proof of Theorem 5.1. Let $P = \{(v_1,v_2),(v_2,v_3),\cdots,(v_{k-1},v_k)\}$ be a $(k-1)$-path in $\vec{G}$. By the color ordering, we have $c(v_i) \le c(v_{i+1})$ for every $i \in [1,k-1]$, where $c(v_i)$ denotes the color value of v_i. Since any two adjacent nodes have different colors, we have $c(v_i) \ne c(v_{i+1})$ for each $i \in [1,k-1]$. As a result, the path S is a k-color path. □

The proof of Theorem 5.2. Let $C = \{v_1,v_2,\cdots,v_k\}$ be a k-clique in G. Clearly, the nodes in C have different colors. Suppose without loss of generality that $c(v_1) < c(v_2),\cdots,c(v_k)$. Since $\vec{G}$ is generated by the color ordering, there must exist a path $\{(v_1,v_2),\cdots,(v_{k-1},v_k)\}$ in $\vec{G}$ which also forms a valid k-color path. □

The proof of Theorem 5.3. Consider a path $\{v_1,v_2,\cdots,v_k\}$. Let X be the event of this path being sampled by Algorithm 4. Denote by Y_i the event of a node v_i appearing in the path. Clearly, the probability of the first node v_1 being sampled is $\Pr(Y_1) = \frac{H(v_1,k-1)}{\sum_{u\in V} H(u,k-1)}$. Observe that in the i^{th}-iteration of the **for** loop (line 15), the distribution D for node v_i is constructed from $N_{v_{i-1}}(\vec{G})$. The node v_i being sampled in the **for** loop can be represented as an event $Y_i|Y_{i-1}$ (conditioned on Y_{i-1}), thus we have $\Pr(Y_i|Y_{i-1}) = \frac{H(v_i,k-i)}{\sum_{u\in N_{v_{i-1}}(\vec{G})} H(u,k-i)}$.

As a consequence, we have

$$\begin{aligned}\Pr(X) &= \Pr(Y_1)\times\Pr(Y_2|Y_1)\times\cdots\times\Pr(Y_k|Y_{k-1})\\ &= \frac{H(v_1,k-1)}{\sum_{u\in V} H(u,k-1)} \times \frac{H(v_2,k-2)}{\sum_{u\in N_{v_1}(\vec{G})} H(u,k-2)}\times\\ &\quad\cdots\times\frac{H(v_k,0)}{\sum_{u\in N_{v_{k-1}}(\vec{G})} H(u,0)}\\ &= \frac{1}{\sum_{u\in V} H(u,k-1)}.\end{aligned} \tag{15}$$

Since the number of k-color paths in G is equal to $\sum_{u\in V} H(u,k-1)$, each k-color path is sampled uniformly. □

The proof of Theorem 5.4. First, the algorithm consumes $O(m+n)$ time to obtain a DAG. Second, as above analyzed, the DPPathCount procedure takes $O(nk\chi)$ time. Third, the DPPathSampling procedure uses $O(n+\chi k)$ time. This is because setting the probability distribution for the first node takes $O(n)$ time, while for the other nodes it takes at most $O(\chi)$ time. Thus, the total time complexity of Algorithm 4 is $O(\chi nk + m)$. For the space complexity, the algorithm needs to store the DAG and the DP table H which uses $O(nk+m)$ space in total. □

8.3 Further related work

***K*-clique and triangle counting.** Except the practical algorithms introduced above, there also exist some theoretical studies on the

k-clique counting problem [11, 16, 17, 20]. Most of these theoretical work focus mainly on devising an algorithm to achieve a better worst-case time complexity. The practical performance of such algorithms is often much worse than the state-of-the-art practical algorithms [28]. Triangle is a specific k-clique for $k = 3$. The problem of counting triangles in a graph has a long history. There are many algorithms in the literature [5, 14, 27, 32, 43]. For example, both [27] and [32] are ordering-based exact triangle counting algorithms. Chu and Cheng [14] developed an I/O-efficient algorithm exact algorithm for triangle listing. Tsourakakis et al. [43] proposed an edge sampling algorithm to approximate the number of triangles in a graph. Becchetti et al. [5] presented an approximate triangle counting algorithm in the semi-streaming model. Tom et al. [40] and Hu et al. [22] developed efficient GPU-parallel algorithms for triangle counting in the shared-memory many-core platforms.

Motif counting. Many exact and sampling-based approximation algorithms have been proposed for motif counting [8, 9, 33, 34, 36]; and some of them can also be used to count k-cliques. Notable example include the color coding based algorithms [8, 9], and edge sampling based algorithms [36]. However, as shown in [23], all these algorithms cannot scale for large graphs and also their practical performance is worse than TuranShadow.

Motif Counting Beyond Five Nodes

MARCO BRESSAN and FLAVIO CHIERICHETTI, Sapienza University of Rome
RAVI KUMAR, Google Research
STEFANO LEUCCI, ETH Zürich
ALESSANDRO PANCONESI, Sapienza University of Rome

Counting graphlets is a well-studied problem in graph mining and social network analysis. Recently, several papers explored very simple and natural algorithms based on Monte Carlo sampling of Markov Chains (MC), and reported encouraging results. We show, perhaps surprisingly, that such algorithms are outperformed by color coding (CC) [2], a sophisticated algorithmic technique that we extend to the case of graphlet sampling and for which we prove strong statistical guarantees. Our computational experiments on graphs with millions of nodes show CC to be more accurate than MC; furthermore, we formally show that the mixing time of the MC approach is too high in general, even when the input graph has high conductance. All this comes at a price however. While MC is very efficient in terms of space, CC's memory requirements become demanding when the size of the input graph and that of the graphlets grow. And yet, our experiments show that CC can push the limits of the state-of-the-art, both in terms of the size of the input graph and of that of the graphlets.

CCS Concepts: • **Mathematics of computing** → *Graph enumeration*; *Graph algorithms*; • **Theory of computation** → *Random walks and Markov chains*; • **Information systems** → *Data mining; Web mining*;

Additional Key Words and Phrases: Motif counting, subgraph counting, color coding, graph mining

ACM Reference format:
Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discov. Data.* 12, 4, Article 48 (April 2018), 25 pages.
https://doi.org/10.1145/3186586

1 INTRODUCTION

Counting graphlets is a well-studied problem in graph mining and social-networks analysis [1, 3, 7, 8, 11, 14, 18, 20, 27–29, 32]. Given an input graph, the problem asks to count the frequencies of all induced connected subgraphs (called *graphlets*), up to isomorphism, of a certain size. This problem

A preliminary version of this article appeared in the *Proceedings of ACM WSDM'17* [6].
Part of the work was done while S. Leucci was at Sapienza University of Rome.
M. Bressan is supported in part by the ERC Starting Grant DMAP 680153 and by the SIR Grant RBSI14Q743. F. Chierichetti is supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award, and by the SIR Grant RBSI14Q743. A. Panconesi is supported in part by a Google Focused Research Award and by the Sapienza inter-disciplinary research grant C26M15ALKP.
Authors' addresses: M. Bressan (corresponding author), F. Chierichetti, and A. Panconesi, Department of Computer Science, Sapienza University of Rome, via Salaria 113, 00198 Roma, Italy; emails: {bressan, flavio, ale}@di.uniroma1.it; R. Kumar, Google Research, 1600 Amphitheater Parkway, Mountain View, CA 94043, USA; email: ravi.k53@gmail.com; S. Leucci, Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland; email: stefano.leucci@inf.ethz.ch.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 1556-4681/2018/04-ART48 $15.00
https://doi.org/10.1145/3186586

is highly motivated in the context of studying behavioral and biological networks. Understanding the distribution of graphlets allows us to make key inferences about the structural properties of the underlying graph and the interaction of the nodes in the graph (e.g., [22]). It sheds light on the type of local structures that are present in the graph, which can be used for a myriad of analysis [3, 8, 16, 27–29]. For example, an extreme case of graphlets are three-node graphlets: counting triangles is a fundamental problem that has been repeatedly studied for the insights it can yield about the health of a network and also for pushing the boundaries of computation that is possible with large networks [13, 21, 23]. How the graphlets form in the first place and how they temporally evolve are semantically more actionable than the interpretation yielded by the mere evolution of nodes and edges.

Since the exact counting of graphlets can be computationally demanding, one usually settles for less ambitious goals. One such goal, and the one we pursue in this article, is frequency estimation: for each subgraph we want to estimate, as accurately as possible, its relative frequency among all subgraphs of the same size. Less ambitiously still, since the number of subgraphs of a given size grows exponentially, we will be interested in the problem of estimating the relative frequency of only the most frequent ones, say, those that appear at least a certain fraction of the time.

There have been two popular approaches to obtaining such estimates. The first is to use Markov Chain Monte Carlo (henceforth, MC). Given an input graph, consider a MC whose nodes (states) are the graphlets, and where two graphlets are connected if they differ by a node. After adding an appropriate number of self-loops to make the chain regular, it follows from standard facts that a random walk of length equal to (or greater than) the mixing time will stop at a uniform node (i.e., a graphlet). This gives a very simple and space-efficient way to sample the population of graphlets. Just repeat the walk independently a large number of times. Recently, this approach has been tried by several authors with encouraging results [3, 11, 20, 27]. However, for this type of approach to be statistically reliable the crucial question is: how long is the mixing time of this natural walk? To the best of our knowledge, this question has not been addressed in a principled manner.

A second approach to count graphlets (especially, trees) is to use color coding (CC), an elegant randomized algorithmic technique introduced in [2]. CC provides strong, provable statistical guarantees for the problem of approximating exact graphlet counts, from which the frequencies can be easily derived. From a computational point of view, perhaps its main drawback is its space requirement, that can quickly become insurmountable as the graphlet size grows. Furthermore, for its nice statistical guarantees to hold in the case of graphlets, one needs to run CC an exponential (in the subgraph size) number of times, which can be prohibitive. This heavy price must be paid if one needs precise estimates of exact counts. But what if one is just interested, as we are in this article, in estimating the frequency of the most common graphlets, can a linear upper bound be attained?

1.1 Our Contributions

In this work, we study MC and CC as the most viable methods for counting reasonably sized graphlets in massive graphs. Our goal is to understand and compare these methods from a practical point of view and en route show provable guarantees. Let n be the size of the input graph and let k be the size of graphlet. We are interested in input sizes that are currently considered challenging, i.e., in the range $n \geq 10^6$ and $k \geq 5$. As of today, efficient algorithms for counting the frequencies of *all* k-graphlets are known only for $k \leq 5$, and if one wants to scale to $k > 5$, then only results about special classes of graphlets are available (see Section 2).

Our first contribution is to study the mixing time of MC. We show that even if the input graph is well-mixing (as most social networks are) and even if there is one graphlet that appears more that 99% of the time, it is possible that MC will take $\Omega(n^{k-1})$ steps before sampling the most frequent graphlet just a single time! Note that this is not far from the naive $O(n^k)$ bound needed to

perform exact counting by an exhaustive enumeration. In particular, this shows that the mixing time of MC can be huge even when that of the input graph is very small, a somewhat counterintuitive statement. For large graphs and even modest values of k this readily implies that, unless one makes specific assumptions about the input graph and exploits them in the analysis, several MC approaches that have been used in the past effectively give no statistical guarantees. On the positive side, we show that the mixing time of MC is $O(n^2\Delta^{2k})$, a bound that can be useful for graphs of moderate size and small maximum degree Δ.

Our next contribution is to study the effectiveness of CC for graphlet counting. We show that the classic CC technique can be easily extended to sample induced graphlets almost uniformly at random in the graph, which enables us to estimate their frequency. This CC extension has two phases: a building phase and a sampling phase. The building phase, which is basically a dynamic program, is a time and space consuming process and is, in a sense, inevitable. The sampling phase, instead, is rather efficient and in practice much faster than MC. We then show that even a single run of CC, whose output can be seen as a large sample of the population of graphlets, gives reasonably good statistical guarantees. We remark that these bounds are still too weak to provide strong confidence in realistic situations. But, at the very least, they offer some evidence that by compounding the estimates obtained with a very few runs of CC one can get good, perhaps even strong, statistical guarantees. We view our result as an encouraging step along this direction, a line of research that we believe is an interesting one. We also describe an alternative extension of CC that requires less space in the building phase at the expense of sampling efficiency; this is the version we employ in our experiments.

Our last contribution is an experimental analysis with real-world graphs to compare the performance of MC and CC when the goal is estimating the frequencies of the most common graphlets. Our experiments are performed on the largest graphs used in recent work, whose sizes vary from small to several millions of nodes, on a commodity machine. The values of k we use range from 5 to 8. In a nutshell, it turns out that CC provides much better estimates, both on a sample-size budget and on a running-time budget. The drawback of CC is mainly its space complexity, which limits the size of the largest instances on which it can run. It is often the case that theoretical bounds are too coarse, while in actuality algorithms are much better behaved. Indeed, this seems to be the case with CC, for which we consistently observe that the estimate given by just a single run of CC is comparable to that obtained by averaging many runs, an outcome that is in line with our bounds. All in all, CC allowed us to reach beyond the current limits of graphlet counting. Notably, we were able to estimate the distribution of graphlets of size 6, 7, and 8 in graphs for which $k = 5$ was the state-of-the-art. As a rule of thumb, in our opinion CC remains preferable where accuracy guarantees are critical, while MC becomes competitive in the remaining cases.

1.2 Outline of the Article

The rest of the article is organised as follows. Section 2 discusses related work. Section 3 pins down the notation and definitions used throughout the article. Our results on MC are given in Section 4. The CC extension is described and analyzed in Section 5. Experimental results are presented in Section 6. The final remarks of Section 7 conclude the article.

2 RELATED WORK

The naive algorithm for counting the exact number of occurrences of all graphlets of size k in an n-node graph by enumeration takes $O(k^2n^k)$ time. Faster exact algorithms are known [10, 30], but their complexity remains $n^{\Theta(k)}$ and are infeasible in practice already for moderate values of n and k. Indeed, the problem is #W[1]-hard and thus unlikely to admit an $f(k)n^{O(1)}$-time algorithm [12]. For $k = 4$, a major improvement in exact counting is the combinatorial method of [1], which was

shown to scale to $n = 148$M. For $k = 5$, [18] managed graphs on up to $n = 4.8$M nodes by exploiting a decomposition of graphlets together with a degree-based orientation of the host graph. Note that these methods are tailored to $k = 4$ and 5 and it is not known how to extend them to $k > 5$. In this work, instead, we focus on techniques that can scale to $k > 5$, at least in principle, even if by only approximating the graphlet count.

In practice, counting (large) graphlets is often performed using approximate algorithms or heuristics. One heuristic approach is *path sampling*, a technique introduced in [14], which consists in sampling a path uniformly at random from G and checking the graphlet it induces; in this way one can sample graphlets having paths as spanning trees. In practice this has been shown to be effective for 4-graphlets [14], and can be adapted to $k = 5$ by using a set of sampling strategies for trees on five nodes [28, 29]. However, for $k > 5$ this method becomes overly intricate and can significantly suffer from rejection (the path is rejected if its k nodes are not distinct). Again, here we aim at counting graphlets of size $k > 5$ through approaches that give provable guarantees.

The first random walk-based algorithm, GUISE, was introduced in [3] and allowed the authors to collect, in just a few minutes, samples of graphlets of size $k = 4$ and 5 in graphs with up to four million nodes—a significant advancement of the state-of-the-art at the time. Further generalizations and refinement of this technique followed [8, 11, 19, 27], confirming its prowess at least for sampling graphlets of size $k \leq 5$ in graphs of a few million nodes (and $k = 6$ on one small graph). Unfortunately, for the two of these algorithms that are currently faster [8, 11] it is unclear how to extend the techniques to the case $k > 5$; in fact, the results available are for $k \leq 5$. The two techniques developed earlier [19, 27], although reportedly slower, are less sophisticated and can be easily employed for any value of k. However, no non-trivial bounds on the running time or number of samples needed to achieve a given accuracy are known for these methods. Obtaining such a result is part of the goals of our work.

The attractive bounds offered by CC has made it possible to push the task of estimating subgraph counts in the realm of graphs with millions of nodes. A first distributed algorithm based on CC, ParSE [32], was used to count seven different subgraphs of size k ranging from 4 to 10 in graphs with up to 20 million nodes. A subsequent distributed scalable implementation of CC, SCALA [20], allowed the authors to count on graphs with 1—2M nodes the number of non-induced paths and trees. Another recent effort to scale CC is [7]: using a distributed algorithm, the authors estimate the occurrences of 10 different subgraphs of treewidth 2 and size up to $k = 10$ nodes, in graphs of up to 2M nodes. While these encouraging results make clear that CC is a promising approach, they leave wide open the important question of estimating the distribution of *induced* subgraphs, aka graphlets. In this article, we show how CC fits the purpose—with almost no overhead.

Finally, a preliminary version of the present work [6] provided a first theoretical and experimental comparison between random walks and CC, suggesting the two are both viable, with CC winning on some large instances. In this article, we complement those by extensively showing that CC is the most promising technique for scaling graphlet counting to $k > 5$ in graphs with millions of nodes. We also improve the lower bound of Lemma 14 in [6] from $\Omega(n^{k-2})$ to $\Omega(n^{k-1})$, see Theorem 4.4.

3 PRELIMINARIES

A graph $G = (V, E)$ is composed of a set V of nodes and a set $E \subseteq \binom{V}{2}$ of edges.[1] In this article, we will assume the graph is connected and undirected. The *degree* of a node $v \in V$ is the number of nodes w such that $\{v, w\}$ is an edge of G: $\deg_G(v) = |\{w \mid \{v, w\} \in E\}|$. We use $\Delta(G)$ to denote the

[1]For a finite set S and an integer $0 \leq k \leq |S|$, we use $\binom{S}{k}$ to denote the set of k-subsets of S, i.e., $\binom{S}{k} = \{T \mid T \subseteq S, |T| = k\}$.

maximum degree of G: $\Delta(G) = \max_{v \in V} \deg_G(v)$. When G is obvious from the context, we simply use $\deg(\cdot)$ and Δ.

Graphlets. Given a graph $G = (V, E)$, and a subset $W \subseteq V$ of its nodes, we let the subgraph of G induced by W, or $G|W$, be the graph composed of the set of nodes W and the set of edges $E \cap \binom{W}{2}$. If $|W| = k$ and if $G|W$ is connected, then $G|W$ is a *k-graphlet* of G. We denote by $\mathcal{V}_k(G)$ the set of k-graphlets of G. Finally, if H is a connected undirected graph on k nodes, we say "the number of occurrences of H in G" to refer to the number of elements in $\mathcal{V}_k(G)$ that are isomorphic to H.

4 GRAPHLETS VIA RANDOM WALKS

In this section, we analyze the performance of graphlet-sampling algorithms based on random walks. Such algorithms are appealing for their simplicity and their encouraging empirical performance; on the other hand, however, they often come with weak theoretical guarantees in terms of n and k. Here, we aim at obtaining bounds on those guarantees. We focus on the two algorithms that are known to work for $k \geq 5$ [19, 27]; faster ones are known for $k \leq 5$ [8, 11] and we shortly discuss them at the end of the section. Both algorithms are based on a very natural approach—the simple random walk on the space of k-graphlet occurrences.

Recall that $\mathcal{V}_k(G)$ is the set of k-graphlet occurrences of G. Consider then a new graph whose nodes are $\mathcal{V}_k(G)$. There is an edge between two graphlets if and only if the node set of one can be obtained by the node set of the other by removing one node and adding another. More precisely,

$$\mathcal{E}_k(G) = \{\{X, Y\} \mid X, Y \in \mathcal{V}_k(G) \text{ and } |X \cap Y| = k - 1\}.$$

We let $\mathcal{G}_k(G)$ be the graph $\mathcal{G}_k(G) = (\mathcal{V}_k(G), \mathcal{E}_k(G))$. When G is clear from the context, we use the notation $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$. We are interested in studying the simple random walk on $\mathcal{G}_k$. First of all, note that the connectedness of G implies the connectedness of $\mathcal{G}_k$. Furthermore, the MC associated to walk is aperiodic under mild conditions: it is sufficient that G is non-bipartite or there is $v \in G$ with $d_v \geq 3$, see e.g., Theorem 3.3 in [27]. We thus assume the chain is aperiodic. If we then let $d(H)$ denote the degree in $\mathcal{G}_k$ of a k-graphlet occurrence $H \in \mathcal{G}_k$, by standard MC theory, we have:

Observation 4.1. *The simple random walk on $\mathcal{G}_k$ converges toward the distribution where H has probability $p(H) = \frac{d(H)}{2|\mathcal{E}_k|}$.*

This observation can be used to build an unbiased estimator for the graphlet frequencies. To this end, one must ensure that each graphlet occurrence visited by the walk has the same weight in the final count. This can be achieved via rejection sampling (accepting H with probability $\frac{1}{d(H)}$) or via reweighting (counting H as a "fraction" $\frac{1}{d(H)}$ of a sample). These rejection and reweighting techniques are at the heart of all state-of-the-art graphlet-sampling algorithms based on random walks [8, 11, 19, 27]. The fundamental question now is how many steps are needed to reach stationarity, or more formally, what is the *mixing time* [17] of the random walk—the number of steps required to reach (within an ϵ-statistical error from) the stationary distribution. The remainder of this section is devoted to developing bounds on such a mixing time.

Let us start by recalling some standard notion. Given a set $W \subseteq V$ of nodes, the *volume* of W is $\text{vol}(W) = \sum_{v \in W} \deg(w)$. The *cut* induced by $W \subseteq V$ is equal to the number of edges that have exactly one endpoint in W, that is, $cut(W) = |\{e : |e \cap W| = 1\}|$. The conductance of a set of nodes $W \subseteq V$ is defined as $\phi(W) = \frac{\text{cut}(W)}{\text{vol}(W)}$. The *conductance* of G is defined as

$$\phi(G) = \min_{\substack{W \subseteq V \\ \text{vol}(W) \leq |E|}} \phi(W).$$

Consider the uniform random walk on G, where at each node $v \in V$, the next node to visit is chosen uniformly at random from among the neighbors of v. Cheeger's inequality [9] implies that the mixing time of the walk is between $\Omega(\phi(G)^{-1})$ and $O(\phi(G)^{-2} \log \frac{1}{\epsilon})$. A large conductance thus implies a small mixing time and vice versa.

Social graphs have been empirically observed to have small mixing times [16]. A natural question then is: can we give small upper bounds on the mixing time of $\mathcal{G}_k$ by using the fact that G has a small mixing time? We will show in Section 4.2 that, unfortunately, the answer to this question is negative in general: there are graphs G with very large (constant) conductance for which the corresponding $\mathcal{G}_k$ has a tiny conductance. Before addressing these lower bounds, in the next section, we give an *upper bound* on the mixing time of $\mathcal{G}_k$ that may be of use in low-degree graphs.

4.1 An Upper Bound on the Mixing Time

Theorem 4.2. *For any given G, we have $\phi(\mathcal{G}_k) \geq \Omega(n^{-1}\Delta^{-k})$, and thus the mixing time of the simple random walk on $\mathcal{G}_k$ is upper bounded by $O(n^2\Delta^{2k})$.*

Proof. Observe that, since G is connected, the degrees of nodes in $\mathcal{G}_k$ range from 1 to $k \cdot \Delta(G) = k\Delta$. We first upper bound the number of k-graphlets of G, i.e., $|\mathcal{V}|$. Note that any element $H \in \mathcal{V}$ with node set $\{v_1, \ldots, v_k\}$ can be mapped to a unique tuple $(v_1, \ldots, v_k)$, where for each $i = 2, \ldots, k$, node v_i neighbors some of $v_1, \ldots, v_{i-1}$. By bounding the number of such tuples, we can thus bound $|\mathcal{V}|$. Any such tuple can be constructed by choosing $v_1, v_2, \ldots, v_k$ in turn. We have n different choices for v_1. Once v_1 is chosen, we have at most Δ possible choices for v_2, since it must be a neighbor of v_1. Similarly, we have at most 2Δ choices for v_3, which must be a neighbor of v_1 or v_2, and so on till v_k for which we have at most $(k-1)\Delta$ choices. Therefore, $|\mathcal{V}| \leq n(k-1)! \cdot \Delta^{k-1}$.

Now, since for any $H \in \mathcal{V}_k$ the degree of H in $\mathcal{G}_k$ is at most $k\Delta$, the volume of any subset of nodes of $\mathcal{V}_k$ can be upper bounded by $k\Delta \cdot |\mathcal{V}_k| \leq k!n\Delta^k$. Furthermore, since $\mathcal{G}_k$ is connected, any non-empty and proper subset of nodes of $\mathcal{V}_k$ will have at least one edge in the cut. It follows that $\phi(\mathcal{G}_k) \geq \frac{1}{k!}n^{-1}\Delta^{-k}$. The upper bound on the mixing time of $\mathcal{G}_k$ then follows. □

4.2 Lower Bounds on the Mixing Time

We next show a mixing time lower bound by exhibiting a graph G with large conductance such that $\mathcal{G}_k$ has tiny conductance.

Definition 4.3. Let $k \in \mathbb{Z}^+$ be given. Let $\ell \in \mathbb{Z}^+$ be sufficiently large. Take ℓ disjoint paths of $2k$ nodes each; create two additional nodes a and b; for each path, add an edge between one of its endpoints and a, and an edge between its other endpoint and b. Let $G = (V, E)$ be the resulting graph.

Note that $|V| = 2\ell k + 2$; let $n = |V|$. One can easily see that the conductance of G is a constant, $\phi(G) = \Theta(1/k) = \Theta(1)$. We next prove that the conductance of $\mathcal{G}_k$ is tiny, which by Cheeger's inequality implies that its mixing time is huge, thus obtaining:

Theorem 4.4. *Let G be the graph of Definition 4.3. Then, the mixing time of G is $\Theta(1)$, and yet the mixing time of $\mathcal{G}_k$ is at least $\Omega(n^{k-1})$.*

Proof. Consider the closed ball S of radius k centered at a. Observe that it is (i) disjoint and (ii) isomorphic to the closed ball T of radius k centered at b. Now, consider the set X of k-graphlets that contain only nodes in S, and the set Y of k-graphlets that contain only nodes in T. By (ii), the subgraph that X induces on $\mathcal{G}_k$ will be isomorphic to the subgraph that Y induces on $\mathcal{G}_k$. Moreover, by (i), those two subgraphs will be disjoint. Thus, $\text{vol}(X) \leq \text{vol}(\mathcal{G}_k)/2 = |\mathcal{E}_k|$ and hence $\phi(\mathcal{G}'_k) \leq \phi(X)$. Now, $|X| = \Omega(n^{k-1})$ since there are $\Omega(n^{k-1})$ graphlets isomorphic to the star on k

nodes centered at a. Also, each such graphlet H satisfies $\deg_{\mathcal{G}_k}(H) = \Omega(n)$, since it is a neighbor of the $\Omega(k\ell)$ graphlets H' that share a and $k-2$ neighbors of a with H itself. Therefore, $\mathrm{vol}(X) \geq |X| \deg_{\mathcal{G}_k}(H) = \Omega(n^k)$. Also, $\mathrm{cut}(X) = \frac{n-2}{2k} \leq O(n)$. Therefore, we have $\phi(\mathcal{G}_k) \leq \phi(X) = \frac{\mathrm{cut}(X)}{\mathrm{vol}(X)} = O(n^{1-k})$. □

We observe that in time $O(n^k)$, one can just enumerate all the k-subsets of nodes of a graph of n nodes, check whether they form a k-graphlet and, if so, which graphlet do they form. As shown above, the random walk on $\mathcal{G}'_k$ has to run for at least $\Omega(n^{k-1})$ steps to guarantee any statistical significance of the sampled graphlet.

Next, we show that not only the random walk does not converge in $o(n^{k-1})$ steps for some graphs with constant conductance, but in fact there are constant-conductance graphs such that $o(n^{k-1})$ steps of the random walk are not even enough to see any copy of a graphlet that occurs an overwhelming fraction (i.e., $1 - o(1)$) of the times. This means we need $\Omega(n^{k-1})$ steps to see some occurrence of a graphlet appearing more than 99% of the times in the graph. We will consider a graph similar to the one in Definition 4.3.

Definition 4.5. Let $k \in \mathbb{Z}^+$ be given. Let $\ell \in \mathbb{Z}^+$ be sufficiently large. Take ℓ disjoint paths of $2k$ nodes each; create an additional node a and, for each path, add an edge between one of its endpoints and a. Construct a clique out of the ℓ other endpoints of the ℓ paths. Let $G = (V, E)$ be the resulting graph.

As before, let $n = |V| = 2\ell k + 1$ and one can show that $\phi(G) = \Theta(1/k) = \Theta(1)$. We prove:

Theorem 4.6. *Let G be the graph of Definition 4.5. Then, G has mixing time $\Theta(1)$, and the k-cliques are a $1 - o(1)$ fraction of the k-graphlets of G. However, if the random walk on $\mathcal{G}_k$ starts from any graphlet containing node a, with high probability it will require $\Omega(n^{k-1})$ steps to reach any k-clique.*

Proof. The number of k-cliques inside the clique of cardinality ℓ is equal to $\binom{\ell}{k} = \Theta(\ell^k) = \Theta(n^k)$. The number of graphlets that contain some node of the clique, and some node outside the clique is $\Theta(n^{k-1})$. The number of k-graphlets that contain a is no more than $O(n^{k-1})$. There are $\Theta(n)$ graphlets that do not contain nodes of the clique and a. Therefore, the number of k-clique graphlets is a $1 - O(1/n) = 1 - o(1)$ fraction of the total number of graphlets.

We now show that, if we start the random walk on $\mathcal{G}_k$ from any graphlet containing a, with high probability we will require $\Omega(n^{k-1})$ steps to reach any graphlet that does not contain a, and thus to reach any k-clique. Let us partition the set of graphlets that contain a into k zones $P_1, \ldots, P_k$, where a graphlet belongs to zone P_i if the maximum distance between one of its nodes and a is i. The starting graphlet is therefore in P_1. We aim to show that it takes $\Omega(n^{k-1})$ steps to reach P_k. Clearly, reaching some graphlet in P_k is necessary if we are to reach some k-clique. Consider now the walk between the P_i's. Observe that, if we are in P_i we can either remain there, or move to P_{i+1} (if $i < k$), or move back to P_{i-1} (if $i > 1$). However, the probability of moving to P_{i+1} is no more than $\frac{k^2}{\ell} = O(n^{-1})$. Moreover, if $i > 1$, then the probability of reaching, in $O(k)$ steps, P_{i-1} from P_i is at least $1 - O(1/n)$. The time required to reach P_k, then, is $\Omega(n^{k-1})$. □

4.3 Other Random Walk Techniques

We conclude by briefly discussing the random-walk algorithms of [8] and [11]. The algorithm of [8] performs a simple random walk on G, and then samples (with appropriate weights) all k-graphlets containing the last $k-1$ nodes visited by the walk if those are distinct. The idea is that the random walk on G mixes faster than that on $\mathcal{G}_k$, but the drawback is that we can only observe k-graphlets that contain a simple path of length $k-1$: for example, we can count cliques or cycles, but not stars. For $k \leq 5$, this is not a problem, as the stars are the only unobservable graphlets and their

frequency can be estimated from the frequencies of other graphlets [8]. However, the number of unobservable graphlets grows to 4 for $k = 6$ and to 33 for $k = 7$; moreover, those graphlets are among the top frequent ones in practice (see Section 6). The algorithm in [11] aims at overcoming both the high mixing time and the unobservability of graphlets. This is done by allowing the simple random walk on G to "waddle" so that it can visit the spanning tree of a graphlet H even if that spanning tree is not a path. Such an approach, however, requires to devise a "waddling strategy" for each graphlet that does not contain a spanning path and only for $k = 4$ such a strategy has been shown. Furthermore, the formal guarantees of such an approach, and in particular a bound on the variance of the estimator, seem difficult to pin down.

5 GRAPHLETS VIA COLOR CODING

In this section, we present two algorithms for graphlet counting that are based on CC, a powerful algorithmic technique introduced by Alon et al. [2]. Given the input graph $G = (V, E)$ and k, coloring coding first assigns uniformly and independently to each node of G a random label in $[k] := \{1, \ldots, k\}$, referred to as a *color*. The goal now is to count the number of *non-induced* trees of k nodes in G—called *treelets*—that are *colorful*, i.e., whose labels have no repetitions. This can be done efficiently by dynamic programming, thanks to the fact that treelets with disjoint set of labels must lie on disjoint set of nodes. Since a treelet is colorful with relatively low probability, one needs to repeat the coloring sufficiently many times in order to "hit" any given treelet.

In its original version, CC requires time $O(c^k \cdot |E|)$ and space $O(c^k \cdot |V|)$ for some $c > 1$, which has made it possible to push the task of estimating subgraph counts in the realm of graphs with millions of nodes. However, the existing algorithms only count subgraphs occurrences that are not induced, and that are either trees or "tree-like" in the sense of having small treewidth. We show that treelet counting can be extended to graphlet counting based on the observation that by counting treelets, we have counted (with high probability if repeated many times) all the spanning trees of every graphlet. To summarize, a good estimate of treelets can be translated into a good estimate of graphlets.

In this section, we first describe the two algorithms that are based on CC. The first algorithm is an extension of the algorithm of Alon et al. [2], and the second is a modification that uses less space at the expense of sampling speed. The latter algorithm is the one we use in our experiments, where we are limited by the amount of available memory. We then prove concentration on the number of colorful treelets produced by one run of (either of these) algorithms. We will use this to prove concentration on the number of colorful graphlets.

5.1 Algorithms

Here, we describe two algorithms based on CC that can count and sample colorful non-induced treelets uniformly at random. We then show how that suffices to sample colorful induced graphlets, as well. Both algorithms consist of a building phase and a sampling phase, and start with a coloring phase where each node $v \in V$ of G is assigned a color $c(v)$ chosen independently and uniform at random from $[k]$.

5.1.1 The First Algorithm (CC1). Our first algorithm, CC1, is as follows. In the building phase (see Algorithm 1), which is essentially the algorithm of Alon et al. [2], we start by creating for each node $v \in G$ a counter $C(T, S, v) = 1$, where T is the trivial graphlet on one node and S is the set of colors $\{c(v)\}$ containing just the color of v. This is the counter of the number of (non-induced) treelets isomorphic to T with color labels spanning S and rooted at v. Then, we perform a dynamic programming to count treelets of size $h = 2, \ldots, k$. For each h in turn, we consider each possible rooted tree T on $h \leq k$ nodes and each possible set $S \subseteq [k]$ with $|S| = h$. Then, for each node $v \in V$,

we can compute the number $C(T, S, v)$ of occurrences of (non-induced) treelets rooted at v that are isomorphic to T and whose colors span the set S, as follows. Split ideally T into two rooted subtrees T_1 and T_2 by removing any edge e incident to the root of T (the endpoints of the edge become the roots of the subtrees); it is easy to see that $C(T, S, v)$ satisfies the following relationship:

$$C(T, S, v) = \frac{1}{d} \sum_{(v,u)\in E} \sum_{\substack{S_1,\ S_2 \subset S \\ S_1 \cap S_2 = \emptyset}} C(T_1, S_1, v) \cdot C(T_2, S_2, u), \tag{1}$$

where d is a normalization constant that is equal to the number of rooted trees isomorphic to T_2 among the subtrees rooted in the children of the root T. To compute $C(T, S, v)$ one then just sweeps over all edges uv of G, combining the counters of u and v. The correctness and complexity of this construction are proved in [2].

ALGORITHM 1: CC1-Build

```
   input: graph G, graphlet size k
1  for v in G do
2      c(v) = color drawn u.a.r. from [k]
3      C(({v}, ∅), {c(v)}, v) = 1
4  for h = 2 to k do
5      for v in G do
6          for each T : |T| = h do
7              for each S ∈ C([k] choose h) do
8                  C(T, S, v) = d^{-1} Σ_{(v,u)∈E} Σ_{S1,S2⊂S, S1∩S2=∅} C(T1, S1, v) · C(T2, S2, u)
9  for v in G do
10     build rng() with Pr[T, v] ∝ C(T, [k], v) for each T : |T| = h do
11         for each S ∈ C([k] choose h) do
12             for each S1 ⊂ S do
13                 build rng(T1, T2, S, v) with Pr[S1, S2] ∝ C(T1, S1, v) Σ_{u:uv∈E} C(T2, S2, u)
14                 build rng(T2, S2, v) with Pr[u] ∝ C(T2, S2, u)
```

In the sampling phase (see Algorithm 2), we use the counters $C(T, S, v)$ to sample a colorful treelet uniformly at random. First, we randomly choose a node v of G and a treelet T on k nodes with probability proportional to the overall number of occurrences of treelets isomorphic to T rooted at v, i.e., to $C(T, [k], v)$. We then choose one of the $C(T, S, v)$ treelets rooted at v that are isomorphic to T and are colored with the colors in S (in this first step, $S = [k]$). To this aim, we split T into T_1 and T_2 as described above. Then, we select a pair of color subsets S_1 and $S_2 = S \setminus S_1$, with size $|S_1| = |T_1|$ and $|S_2| = |T_2|$, with probability proportional the number of T's rooted at v that are formed by T_1 and T_2 colored with S_1 and S_2; that is, proportional to $C(T_1, S_1, v) \cdot \sum_{u:(u,v)\in E} C(T_2, S_2, u)$. Next, we choose the neighbor u where to root T_2 with probability proportional to $C(T_2, S_2, u)$. Then, we recursively sample one of the $C(T_1, S_1, v)$ (resp. $C(T_2, S_2, v)$) treelets isomorphic to T_1 (resp. T_2) and colored with the colors in S_1 (resp. S_2) from v (resp. u). It is immediate to verify that this procedure yields a colorful treelet occurrence chosen u.a.r. among all those in G.

To reduce the complexity of the sampling phase, in CC2 we pre-build the random number generators (lines 9–14), as follows. First, for any possible treelet on k nodes and any $v \in G$, we build a generator returning the pair (T, v) with probability proportional to $C(T, [k], v)$. Then, for each node $v \in G$, we build $O(c^k)$ generators to draw the pairs of label sets and to draw a neighbor of

u. Such generators can be built in time and space linear in the size of the alphabet, and produce a sample in $O(1)$ time [26]. Thus, in our case they take overall $O(c^k|E|)$ time and space.

ALGORITHM 2: CC1-Sample

input: (T, S, v) or Null
output: colorful k-treelet chosen u.a.r. from those isomorphic to T with labels S rooted at v

```
if input = Null then
    (T, v) = rng()
    S = [k]
if |T| = 1 then
    return ({v}, ∅), {c(v)}
decompose T into T1 + e + T2
(S1, S2) = rng(T1, T2, S, v)
u = rng(T2, S2, v)
H1 = CC1-Sample(T1, S1, v)
H2 = CC1-Sample(T2, S2, u)
return H1 + vu + H2
```

From the algorithms, one can immediately obtain the following complexity bounds:

Theorem 5.1. *CC1-Build takes time and space $O(c^k|E|)$ for some $c > 0$. CC1-Sample takes time $O(k)$.*

5.1.2 The Second Algorithm (CC2). Our second algorithm, CC2 is a simple variant of CC1 that saves memory at the expense of sampling speed. In CC2, we do not precompute the random number generators that allow to (recursively) select the subtrees T_1, T_2 and the subsets of labels S_1, S_2 in time $O(1)$. Instead, we create the distributions of T_1, T_2 and S_1, S_2 at sampling time starting from the counters $C(\cdots)$. This requires to sum the counters $C(\cdots)$ over all neighbors of v, but since the degrees of the graphs we deal with are not too large, the impact on the sampling time is hopefully small. The advantage is that the space complexity of CC2 becomes $O(c^k|V|)$. This allows us to reduce the overall memory footprint of the algorithm—a determining factor in practice, since CC is typically limited by memory. Formally, one can prove:

Theorem 5.2. *CC2-Build takes time $O(c^k|E|)$ and space $O(c^k|V|)$. CC2-Sample generates a treelet sample T in time $O(c^k \sum_{v \in T} d_v)$.*

Finally, in CC2 we store only the positive counters $C(\cdots)$. In this way, the amount of memory required by CC2 is actually dictated by the number of different colored treelets that are in the graph. This can be much smaller than $c^k|V|$. For this reason, in practice the memory footprint of CC2 is not strictly proportional to the size of the graph, and on some larger graphs, we could reach larger values of k (see Section 6).

5.1.3 From Treelets to Graphlets. Once we can sample an occurrence of a colorful treelet on k nodes uniformly at random from the set of all colorful treelet occurrences in G, it is possible to also sample a colorful graphlet H by noticing that a graphlet contributes $k\sigma(H)$ to the treelet count, where $\sigma(H)$ is the number of spanning trees of H (the factor k comes from the fact that a single colorful tree contributes k to the count). Hence, to sample a colorful graphlet uniformly at random one can proceed as follows:

(i) sample an occurrence T of a treelet on k nodes from G
(ii) consider the graphlet H induced by the nodes of T, and
(iii) reject H with probability $1 - \frac{1}{\sigma(H)}$.

If the goal is to obtain an unbiased estimate of the graphlets frequency, at step (iii) instead of rejection sampling, we can simply add to the graphlet count the quantity $\frac{1}{\sigma(H)}$. This never increases the variance of the resulting estimator and, depending on the distribution, can significantly decrease it. We use this approach in our implementation. The value $\sigma(H)$ can be (pre)computed for any given H in time $O(k^\omega)$, where ω is the matrix multiplication exponent, e.g., via Kirchhoff's Matrix-Tree Theorem [24]. In our case, we compute $\sigma(H)$ the first time H is sampled, caching it for later reuse.

5.2 Concentration of Colored Graphlets

In this section, we prove concentration bounds on the number of colorful graphlets produced by CC1 and CC2 and, in fact, by any random uniform coloring of the nodes of G. We assume $k \geq 3$ and we denote by $g = |\mathcal{V}_k(G)|$ the total number of k-graphlets in G. Our goal is to show that, with high probability, the coloring preserves the distribution of graphlets. More formally, consider a subset $\mathcal{S} \subseteq \mathcal{V}_k(G)$ of the k-graphlets of G; for instance, the set of all k-cliques of G. The expected number of such graphlets that are colorful is $\sum_{H\in\mathcal{S}} \Pr[H \text{ is colorful}] = |\mathcal{S}|\frac{k!}{k^k}$; so we could recover $|\mathcal{S}|$ from an accurate estimate of such a number. Unfortunately, the actual number of colorful graphlets can fall far from $|\mathcal{S}|\frac{k!}{k^k}$. Indeed, there may be strong correlations between the colorings of different graphlets in $\mathcal{S}$; for instance, there can be $\Theta(n^{k-2})$ graphlets sharing two nodes v, v' of G, and if v and v' have the same color, then all those $\Theta(n^{k-2})$ will be simultaneously uncolorful. However, we can show that, if $\mathcal{S}$ is large enough, then there is concentration. Our main result is the following:

Theorem 5.3. *Consider any $\mathcal{S} \subseteq \mathcal{V}_k(G)$, and let $Z_\mathcal{S}$ be the random variable counting the number of $H \in \mathcal{S}$ that are made colorful by a coloring of G. Let $s = |\mathcal{S}|$ and $\mu_\mathcal{S} = \mathbb{E}[Z_\mathcal{S}]$. Then, for any $\epsilon > 0$:*

$$\Pr\Big[\, |Z_\mathcal{S} - \mu_\mathcal{S}| > \epsilon\, \mu_\mathcal{S} \,\Big] \leq e^{-\Omega(\epsilon^2 s^{1-1/k}/g^{1-2/k})}, \tag{2}$$

where the $\Omega(\cdot)$ notation hides factors that depend on k but not on g and s.

The exponent of the bound is in $\Omega(1)$ as long as $s \in \Omega(g^{1-\frac{1}{k-1}})$, i.e., as long as $\mathcal{S}$ is relatively large w.r.t. the total number of graphlets of G. For instance, when counting the most frequent graphlets (say, those appearing at least a 1% of the times) we are looking at $s \in \Omega(g)$, which is in that range. The proof of Theorem 5.3 is rather technical and can be skipped without impairing the understanding of the rest of the article; the interested reader can find it in Section 5.3.

We next prove that the bound of Theorem 5.3 is tight. Formally:

Theorem 5.4. *There exist arbitrarily large graphs G with a subset of k-graphlets $\mathcal{S}$ such that $|\mathcal{S}| = \Omega(g^{1-\frac{1}{k-1}})$ and $\Pr[Z_\mathcal{S} = 0] \geq \frac{1}{k} = \Omega(1)$.*

Proof. Consider G formed by a star on $n-1$ nodes and an additional node u' attached to a leaf node u of the star. There are $\binom{n-1}{k-1} = \Omega(n^{k-1})$ graphlets isomorphic to stars, thus $g = \Omega(n^{k-1})$. The set $\mathcal{S}$ of graphlets not spanned by stars contains all and only those graphlets containing both u and u', which are $\binom{n-2}{k-2} = \Omega(n^{k-2}) = \Omega(g^{1-\frac{1}{k-1}})$. The probability that *all* such graphlets are not colorful, and thus that $Z_\mathcal{S} = 0$, is at least $\Pr[u \text{ and } u' \text{ have the same color}] \geq \frac{1}{k}$. □

By Theorem 5.3, the distribution of colorful graphlets (which can be sampled via our algorithms CC1 and CC2) closely matches the overall distribution of graphlets, at least for graphlets that occur often enough. A formal statement is the following:

Corollary 5.5. *Let $\mu_H \in [0, 1]$ be the fraction of graphlet occurrences of G that are isomorphic to H, and let S be a graphlet drawn uniformly at random from the set of colorful graphlets of G. Then,*

for any $\epsilon > 0$*, with probability* $1 - e^{-\Omega(\epsilon^2 g^{1/k})}$:

$$\Pr\left[S \text{ is an occurrence of } H\right] \in \left[\mu_H - \frac{2\epsilon}{1+\epsilon}, \mu_H + \frac{2\epsilon}{1-\epsilon}\right]. \tag{3}$$

Proof. Let $Z_{\mathcal{S}}$ be the number of colorful elements of $\mathcal{S} = \mathcal{S}_H$, the set of graphlets isomorphic to H. Let Z be the total number of colorful graphlets in G. What we are bounding is the probability that $Z_{\mathcal{S}}/Z$ is too far from $\mathbb{E}[Z_{\mathcal{S}}]/\mathbb{E}[Z] = \mu_H$. Suppose that $\mathbb{E}[Z_{\mathcal{S}}] \geq \frac{1}{2}\mathbb{E}[Z]$. Then, $\mathbb{E}[Z_{\mathcal{S}}], \mathbb{E}[Z] \in \Theta(g)$ and by the bounds of Theorem 5.3 with probability $1 - e^{-\Omega(\epsilon^2 g^{1/k})}$, we have $Z_{\mathcal{S}} \in \mathbb{E}[Z_{\mathcal{S}}](1 \pm \epsilon)$ and $Z \in \mathbb{E}[Z_{\mathcal{S}}](1 \pm \epsilon)$. By standard calculations, it follows that $\mathbb{E}[Z_{\mathcal{S}}]/\mathbb{E}[Z] \in \left[\mu_H - \frac{2\epsilon}{1+\epsilon}, \mu_H + \frac{2\epsilon}{1-\epsilon}\right]$. If $\mathbb{E}[Z_{\mathcal{S}}] < \frac{1}{2}\mathbb{E}[Z]$, then one can obtain the bound by applying the argument above to $Z - Z_{\mathcal{S}}$. □

Finally, if one creates $\lambda \geq 1$ random colorings of G and takes the average counts of each graphlet, one gets the following improved concentration bound.

Corollary 5.6. *If we consider* λ *independent colorings of* G *and let* $Z_{\mathcal{S}} = \lambda^{-1} \sum_{i=1}^{\lambda} Z^i_{\mathcal{S}}$ *where* $Z^i_{\mathcal{S}}$ *counts the number of colorful graphlets of* $\mathcal{S}$ *in the* ith *coloring, then the bound of Theorem 5.3 becomes*:

$$\Pr\left[\, |Z_{\mathcal{S}} - \mu_{\mathcal{S}}| > \epsilon \cdot \mu_{\mathcal{S}} \,\right] \leq e^{-\Omega(\lambda \epsilon^2 s^{1-1/k}/g^{1-2/k})}.$$

Proof. It is sufficient to modify slightly the proof of Theorem 5.3 (see below). Consider λ martingale sequences, each one like the one used in the proof, associated to λ independent colorings of G. Juxtapose them to obtain a single martingale sequence of length λn, whose expectation is $\mu'_{\mathcal{S}} = \lambda \mu_{\mathcal{S}}$. We can then apply the Azuma–Hoeffding inequality to this martingale sequence. The denominator of the bound's exponent becomes $2\lambda \sum_{i=1}^{n} c_i^2$, where the c_i's are the same of the proof of Theorem 5.3. Into the numerator of the exponent, now we plug $t = \epsilon \mu'_{\mathcal{S}} = \epsilon \lambda \mu_{\mathcal{S}}$. Since t is squared at the numerator, one can check that the bound gains a factor λ at the exponent. □

5.3 Proof of Theorem 5.3

The key steps are as follows. We assume the nodes of G are colored in non-increasing order of number of graphlets they appear in (clearly any order is equivalent). We then consider the (Doob) martingale that counts the expected number of colorful graphlets in $\mathcal{S}$ given the colors assigned to the first i nodes, for each $i = 1, \ldots, n$. By applying the method of bounded differences, we get a concentration inequality whose exponent's denominator has one term for each node of G, telling how much the martingale can oscillate when we color that node. Thanks to the ordering of the nodes, bounding the vast majority of terms due to nodes that appear in a few graphlets is relatively easy; less so for the other terms, that also depend on how many graphlets can be shared by two nodes. This requires us to prove that two nodes cannot simultaneously appear in too many graphlets (one of the two must appear in asymptotically more).

Let us start with some notation. For $i = 1, \ldots, n$, let $X_i \in [k]$ be the random variable denoting the color of node i. For $j = 1, \ldots, s$ let $Y_j \in \{0, 1\}$ be the indicator random variable of the event that the jth graphlet of $\mathcal{S}$ is colorful, and let $Z = Z_{\mathcal{S}} = \sum_{j=1}^{s} Y_j$ be the total number of colorful graphlets of $\mathcal{S}$. Finally, for $i = 1, \ldots, n$ let $Z_i = \mathbb{E}[Z|X_1, \ldots, X_i]$ be the expectation of Z as a function of the colors assigned to the first i nodes, and let $Z_0 = \mathbb{E}[Z]$. The sequence $Z_0, \ldots, Z_n$ is a Doob martingale with respect to $X_1, \ldots, X_n$, and Azuma's inequality implies

$$\Pr\left[\, |Z - \mathbb{E}[Z]| > t \,\right] < 2e^{-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}}, \tag{4}$$

whenever $|Z_i - Z_{i-1}| \leq c_i$.

Let now $g_{\mathcal{S}}(u) = |\{H \in \mathcal{S} : u \in H\}|$ be the number of graphlets of $\mathcal{S}$ in which u appears. The rest of the proof is devoted to showing that, if the nodes of G are sorted in non-increasing order of

$g_{\mathcal{S}}(i)$, then $\sum_{i=1}^{n} c_i^2 = O(s^{1+\frac{1}{k}}/g^{1-\frac{2}{k}})$. Together with Equation (4), and since $\mu_{\mathcal{S}} = \frac{k!}{k^k}s = \Theta(s)$, this implies the theorem's claim for $t = \epsilon\mu_{\mathcal{S}}$. Start by breaking $\sum_{i=1}^{n} c_i^2$ in two parts:

$$\sum_{i=1}^{n} c_i^2 = \sum_{i=1}^{\ell} c_i^2 + \sum_{i=\ell+1}^{n} c_i^2, \tag{5}$$

for some ℓ to be chosen later. Note that $c_i \leq g_{\mathcal{S}}(i)$: the conditioning on X_i can alter, by at most 1, the expectation of only those Y_j associated to graphlets containing i. Also, $g_{\mathcal{S}}(i) \leq \frac{1}{i}\sum_{u=1}^{n} g_{\mathcal{S}}(u)$ by the ordering of the nodes. Finally, $\sum_{u=1}^{n} g_{\mathcal{S}}(u) = ks = O(s)$, and thus $g_{\mathcal{S}}(i) = O(\frac{s}{i})$. Hence, the second term in Equation (5) can be bounded as

$$\sum_{i=\ell+1}^{n} c_i^2 \leq \sum_{i=\ell+1}^{n} g_{\mathcal{S}}(i)^2 = \sum_{i=\ell+1}^{n} O\left(\frac{s}{i}\right)^2 = O\left(\frac{s^2}{\ell}\right). \tag{6}$$

The rest of the proof focuses on bounding $\sum_{i=1}^{\ell} c_i^2$. First of all notice, that if the graphlet associated to Y does not contain i or does not contain one among $1, \ldots, i-1$, then $\mathbb{E}[Y|X_1, \ldots, X_i] = \mathbb{E}[Y|X_1, \ldots, X_{i-1}]$. Therefore, c_i is bounded by the number of graphlets of $\mathcal{S}$ that contain both i and at least one of $1, \ldots, i-1$. Let then $g_{\mathcal{S}}(i,j) = |\{H \in \mathcal{S} : i, j \in H\}|$ and let $g(i,j) = g_{\mathcal{G}_k}(i,j)$. Clearly, $g_{\mathcal{S}}(i,j) \leq g(i,j)$. Thus, we have

$$c_i \leq \sum_{j=1}^{i-1} g_{\mathcal{S}}(i,j) \leq \sum_{j=1}^{i-1} g(i,j). \tag{7}$$

Assume now that $g(i,j) = O((g(i)+g(j))^{\frac{k-2}{k-1}})$, which we indeed prove later. Therefore, the right-hand side of the equation above is $\sum_{j=1}^{i-1} g(i,j) = O(\sum_{j=1}^{i-1}(g(i)+g(j))^{\frac{k-2}{k-1}})$. Now by the ordering of nodes $g(i)+g(j) \leq 2g(j) = O(g/j)$, hence

$$c_i \leq O\left(\sum_{j=1}^{i-1}(g(i)+g(j))^{\frac{k-2}{k-1}}\right) = O\left(\sum_{j=1}^{i-1}(g/j)^{\frac{k-2}{k-1}}\right) = O\left(g^{1-\frac{1}{k-1}} i^{\frac{1}{k-1}}\right), \tag{8}$$

with the last equality following from standard analysis. By using this bound in $\sum_{i=1}^{\ell} c_i^2$, we get

$$\sum_{i=1}^{\ell} c_i^2 = O\left(\sum_{i=1}^{\ell} g^{2-\frac{2}{k-1}} i^{\frac{2}{k-1}}\right) = O\left(g^{2-\frac{2}{k-1}} \ell^{1+\frac{2}{k-1}}\right). \tag{9}$$

Finally, by setting $\ell = s^{1-\frac{1}{k}} g^{\frac{2}{k}-1}$ in both Equations (6) and (9), we obtain $\sum_{i=1}^{n} c_i^2 = O\left(s^{1+\frac{1}{k}} g^{1-\frac{2}{k}}\right)$ as desired.

It only remains to prove:

Lemma 5.7. *For any $u, v \in G$ it holds $g(u,v) = O\left((g(u)+g(v))^{\frac{k-2}{k-1}}\right)$.*

Proof. For any $k \geq 1$ let $\mathcal{H}_k(u)$ denote the set of graphlets of size k of G that contain u (so $\mathcal{H}_1(u)$ contains only the trivial graphlet formed by u alone). For any graphlet occurrence H, let d_H be the sum of the degrees of its nodes in G; i.e., if $u_1, \ldots, u_k$ are the nodes of H then $d_H = \sum_{i=1}^{k} d_{u_i}$. To avoid ambiguities w.r.t. k, we use $g_k(u)$ instead of $g(u)$ to denote $|\mathcal{H}_k(u)|$. Note that $g_k(u) > 0$ for all k since G is connected by hypothesis, so we can safely employ Landau notation.

We start by proving that $g_k(u)$ is proportional to the sum of the degrees of the graphlets of size $k-1$ containing u:

$$\sum_{\mathcal{H}_{k-1}(u)} \frac{1}{k}\left\lceil \frac{1}{k-1}\left(d_H - 2\binom{k-1}{2}\right)\right\rceil \leq g_k(u) \leq \sum_{\mathcal{H}_{k-1}(u)} d_H.$$

The upper bound follows immediately by noting that any element of $\mathcal{H}_k(u)$ can be obtained by adding to some $H \in \mathcal{H}_{k-1}(u)$ one of the neighbors of its nodes, and those neighbors are at most d_H. Consider now any $H \in \mathcal{H}_{k-1}(u)$. Since the arcs within H are at most $\binom{k-1}{2}$, and since G is connected, there must be at least $d_H - 2\binom{k-1}{2} > 0$ arcs between H and $G \setminus H$. These arcs then lead to at least $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil$ distinct nodes, each of which can be added to obtain an element of $\mathcal{H}_k(u)$. Any element of $\mathcal{H}_k(u)$ can be obtained in this way, and from at most k elements of $\mathcal{H}_{k-1}(u)$. The lower bound then follows by summing $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil$ over all $H \in \mathcal{H}_{k-1}(u)$ and dividing by k.

We can now prove the following crucial fact:

$$g_k(u) = \Omega\left(g_{k-1}(u)^{\frac{k-1}{k-2}}\right). \tag{10}$$

The proof is by induction on k. The claim holds trivially for $k = 2$. Let us assume it holds for some $k \geq 2$ and focus on proving it for $k+1$. Since $g_{k+1}(u) = \Omega(\sum_{H' \in \mathcal{H}_k(u)} d_{H'})$, we will show the right-hand side is in $\Omega(g_k(u)^{\frac{k}{k-1}})$. Recall that from any $H \in \mathcal{H}_{k-1}(u)$ and its neighbors in G one can create $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil = \Omega(d_H)$ graphlets of $\mathcal{H}_k(u)$; each such graphlet H' includes all the nodes of H, hence has degree $d_{H'} \geq d_H$, and may be obtained from at most k distinct graphlets H. Therefore,

$$\sum_{\mathcal{H}_k(u)} d_{H'} \geq \frac{1}{k} \sum_{\mathcal{H}_{k-1}(u)} \Omega(d_H) \cdot d_H = \Omega\left(\sum_{\mathcal{H}_{k-1}(u)} d_H^2\right). \tag{11}$$

Now,

$$\sum_{\mathcal{H}_{k-1}(u)} d_H^2 \geq \frac{1}{g_{k-1}(u)} \left(\sum_{\mathcal{H}_{k-1}(u)} d_H\right)^2 = \Omega\left(\frac{g_k(u)^2}{g_{k-1}(u)}\right), \tag{12}$$

where the first inequality follows from convexity and the second from $\sum_{\mathcal{H}_{k-1}(u)} d_H = \Omega(g_k(u))$. Now by the inductive hypothesis $g_{k-1}(u) = O(g_k(u)^{\frac{k-2}{k-1}})$, which used in the denominator of the right-hand side proves Equation (10).

We can now conclude the proof of Lemma 5.7. First of all note that any graphlet of size k containing both u and v is the union of (the sets of nodes of) two smaller graphlets: one of size h containing u (but possibly not v), and one of size $k-h$ containing v (but possibly not u), for some $h \in \{1, \ldots, k-1\}$. It follows that:

$$g(u,v) \leq \sum_{h=1}^{k-1} g_h(u) g_{k-h}(v). \tag{13}$$

Since k is a constant, we can thus choose $h \in \{1, \ldots, k-1\}$ such that $g_h(u)g_{k-h}(v) \geq \Omega(g(u,v))$. If $h = 1$, since $g_1(u) = 1$, then $g_{k-1}(v) = \Omega(g(u,v))$, and $g_k(v) = \Omega((g(u,v))^{\frac{k-1}{k-2}})$ by Equation (10). Similarly, if $h = k-1$, we obtain $g_k(u) = \Omega((g(u,v))^{\frac{k-1}{k-2}})$.

Assume then $h \in \{2, \ldots, k-2\}$. If $g_h(u) = \Omega(g(u,v)^{\frac{h-1}{k-2}})$, by Equation (10) $g_k(u) = \Omega(g_h(u)^{\frac{k-1}{h-1}}) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$. Otherwise $g_{k-h}(v) = \Omega(g(u,v)/g_h(u)) = \Omega(g(u,v)^{\frac{k-h-1}{k-2}})$, but then Equation (10) implies $g_k(v) = \Omega(g_{k-h}(v)^{\frac{k-1}{k-h-1}}) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$. In any case, $g(u) + g(v) = g_k(u) + g_k(v) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$, which concludes the proof. □

6 EXPERIMENTS

In this section, we compare the practical performance of CC2 and of its main competitor, the Pairwise Subgraph Random walk (PSRW) of [27]—as discussed in Section 4.3, this is the only

Table 1. The Graph Datasets Used in Our Experiments (Largest Connected Component Only)

Name	Nodes	Edges	k	Source dataset
WordAssoc	10, 6K	63, 8K	8	LAW, wordassociation-2011
Facebook	63, 4K	0, 8M	8	MPI-SWS, Facebook New Orleans
Amazon	0, 3M	0, 9M	8	SNAP, com-amazon.ungraph
DBLP	0, 3M	1, 0M	7	SNAP, com-dblp.ungraph
Yelp	0, 2M	1, 3M	7	YLP, Yelp
Road-PA	1, 1M	1, 5M	6	SNAP, roadnet-PA
Road-CA	2, 0M	2, 8M	7	SNAP, roadnet-CA
BerkStan	0, 7M	6, 6M	5	SNAP, web-BerkStan
Skitter	1, 7M	11, 1M	5	SNAP, as-skitter
Patents	3, 8M	16, 5M	6	SNAP, cit-Patents
Road-US	24, 9M	28, 9M	see note	NDR, inf-road-usa
LiveJournal	5, 4M	49, 5M	6	LAW, ljournal-2008
Hollywood	1, 9M	114, 3M	6	LAW, hollywood-2009
Twitter	41, 7M	117, 2M	see note	LAW, twitter-2010
Orkut	3, 1M	223, 5M	5	MPI-SWS, orkut-2007

Here, k is the largest graphlet size for which we could successfully run algorithm CC2 within the memory resource limits of our machines. LAW: http://law.di.unimi.it/, [4, 5]. MPI-SWS: http://socialnetworks.mpi-sws.org/data-wosn2009.html, [25]. SNAP: https://snap.stanford.edu/, [15, 31]. YLP: https://www.yelp.com/dataset_challenge/.

random-walk technique available that can be employed for $k > 5$. We note that PSRW (like other similar random walk techniques) has been developed with the primary goal of minimizing the number of nodes of G visited by the walk; in the present article, however, we investigate it in terms of samples taken, running time, and accuracy.

6.1 Setup

We implemented CC2 and PSRW in a multi-threading fashion. For CC2, in the building phase at each level of the dynamic program each thread takes care of merging a subset of counters, while in the sampling phase each thread executes the sampling algorithm independently. For PSRW, each thread runs a single random walk independently. We note that PSRW's running time is dominated by enumerating the possible transitions from the current graphlet occurrence H. We implemented this routine by intersecting, for each $u \in H$, the set of neighbors of the connected components left in H by removing u; this reduced the running time by as much as 100× w.r.t. the naive implementation. Our code is written in Java and based on the WebGraph library.[2] It is publicly accessible at https://github.com/Steven--/graphlets. Our platform was a commodity machine equipped with 64GiB of main memory and 32 Intel Xeon CPU cores at 2.5GHz with 30MB of L3 cache, using Oracle's Java Virtual Machine (JVM) (version 1.8.0).

Experiments were executed on 15 graphs that appeared as largest instances in previous work; Table 1 shows the graphs and the largest k for which CC2 ran successfully, i.e., within the available memory limits. Each graph was made undirected, and only the largest connected component was kept, to ensure the correctness of PSRW (as the walk cannot reach one component from another). For Twitter and Road-US, just loading the graph exceeded the available main memory, and we could not run either PSRW or CC2; as a sanity check, we tried a high-end machine with 240GiB

[2] http://webgraph.di.unimi.it/.

memory, and we could run CC2 on Twitter for $k = 4$ and on Road-US for $k = 7$. Those two graphs are therefore omitted from now on.

Concerning the ground-truth graphlet frequencies, we operated as follows. For $k = 5$, for all graphs except Hollywood we could successfully run the exact algorithm of [18] and obtain the precise count of all graphlets. In each other case, we took the average of 50 independent runs of CC2, for a total of five million samples; the empirical low variance of the estimates (see below) strongly suggests that such an average is indeed very close to the true distribution.

6.2 Color Coding Versus Random Walks

We measured the accuracy of CC2 and PSRW as a function of the number of samples and of the running time. For PSRW, we performed 25 independent runs; each execution picks a random initial node in the graph, then simulates 32 random walks in parallel from that node (each walk using a different number generator seed). For CC2, we took the 50 independent runs used to compute the ground truth. In both the cases, we stopped at $100k$ graphlet samples, and along the way we measured the elapsed wall-clock time returned by the operating system, excluding the time to load the graph. Note that running times shall be taken with caution; our is a specific implementation, and times may change as a result of optimizations or hardware modifications.[3] Moreover, small times are affected by artefacts such as the warm-up of the JVM.

Accuracy versus sample size. Let us start with the accuracy versus the number of samples. For each single execution, we computed the accuracy of the estimates, meant as the 1-norm of the residual between the ground truth and the distribution estimated by the algorithm. Then, for each instance (graph, k, number of samples taken), we computed the average the distance over all the executions. For each k, for each input graph, we computed the mean and standard deviation of this 1-norm accuracy over all runs, for both PSRW and CC2. This procedure was repeated after $1k$ samples, $10k$ samples, and $100k$ samples taken by the algorithms. Figure 1 summarizes the results. CC2 appears always at least as accurate as PSRW, and in fact on many instances its accuracy is better by orders of magnitude. Note that, while for $k > 5$ the ground truth is the average of all CC2 runs, which may be in favor of CC2, for $k = 5$, we are using the exact graphlet count.

Figure 2 shows the accuracy of PSRW and CC2 as a function of the number of samples. We observe that CC2 starts converging immediately, while PSRW often exhibits a "bootstrap" phase where the distance from the ground truth remains virtually unchanged. This makes sense: CC2 immediately starts sampling from the distribution of colorful graphlets (hopefully close to the ground truth), while PSRW needs to reach mixing time first. It also suggests that our worst-case lower bounds on the mixing time (Section 4.2) might be not so far from reality after all.

Single-graphlet accuracy. Next, we measured the accuracy of PSRW and CC2 in estimating the frequencies of single graphlets. Note that any sampling-based method has an intrinsically poor accuracy for graphlets that occur very rarely; if we take s samples, a reasonable threshold is to consider only graphlets with ground-truth frequency at least $1/s$. Furthermore, since the number of k-graphlets is already in the hundreds for $k = 6$, we need an aggregate measure of accuracy. Consistently with past work, we used the normalized root-mean-square error (NRMSE). Denote by $f_H(G)$ the ground-truth relative frequency of H in G. Denote by $\hat{f}_H^i(g)$ the frequency estimated obtained from the ith run of the algorithm. Then, the normalized mean-square error is $NMSE(H, G) = r^{-1} \sum_{i=1}^{r} (\hat{f}_H^i(g)/f_H(G) - 1)^2$. The NRMSE$(H, G)$ is simply the square root of $NMSE(H, G)$. Table 2 shows the average NRMSE of PSRW and CC2 over all runs, all graphs,

[3]For instance, because CC2 accesses vast memory regions in a random fashion, while PSRW exploits more the CPU and its data cache.

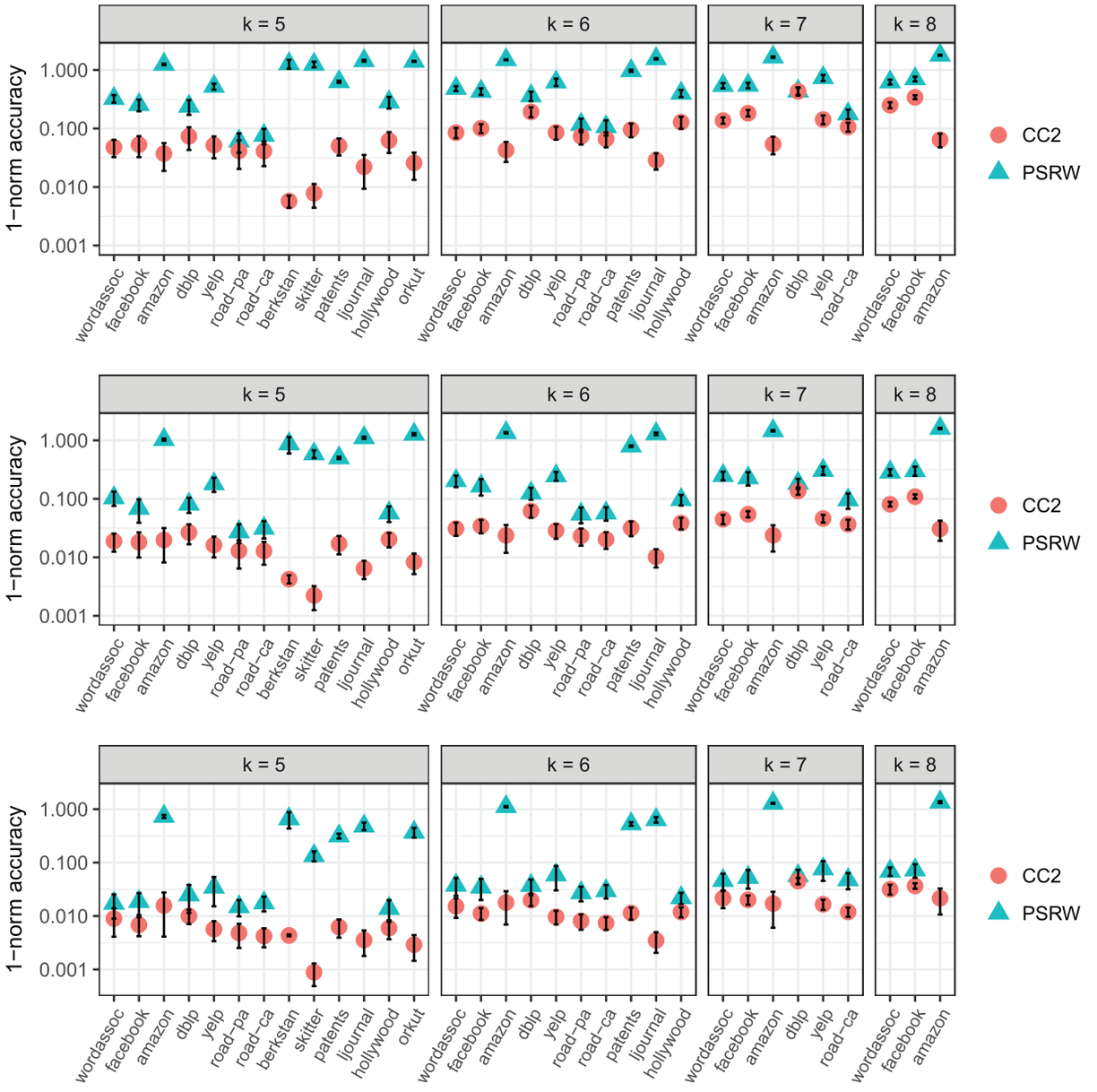


Fig. 1. Accuracy of CC2 and PSRW: average distance from ground truth, ± one standard deviation, after $1k$ samples (top), $10k$ samples (middle), and $100k$ samples (bottom).

and all graphlets, for $s = 1k$, $10k$, and $100k$. The NRMSE of CC2 is always significantly lower than that of PSRW—and the gap increases with the number of samples. Note that NRMSE penalises large errors, so these results confirm that CC2 is reliably accurate on all graphs. Observe also that, while the NRMSE of CC2 increases monotonically with s, the NRMSE of PSRW *decreases* with s. This looks counterintuitive, but can be again be explained by high mixing time—if the walk gets "stuck" in a part of the graph where a graphlet is particularly scarce, then the estimates will progressively worsen.

Accuracy versus time. Finally, we look at the accuracy versus the running time. Note that there are some subtleties in this comparison. On the one hand, PSRW starts sampling immediately, while CC2 first performs an (expensive) building phase. On the other hand, when sampling, CC2 immediately produces unbiased samples, while PSRW must wait until the walk mixes. Keeping this in mind, it makes sense to compare the accuracy of the two algorithms after the same elapsed

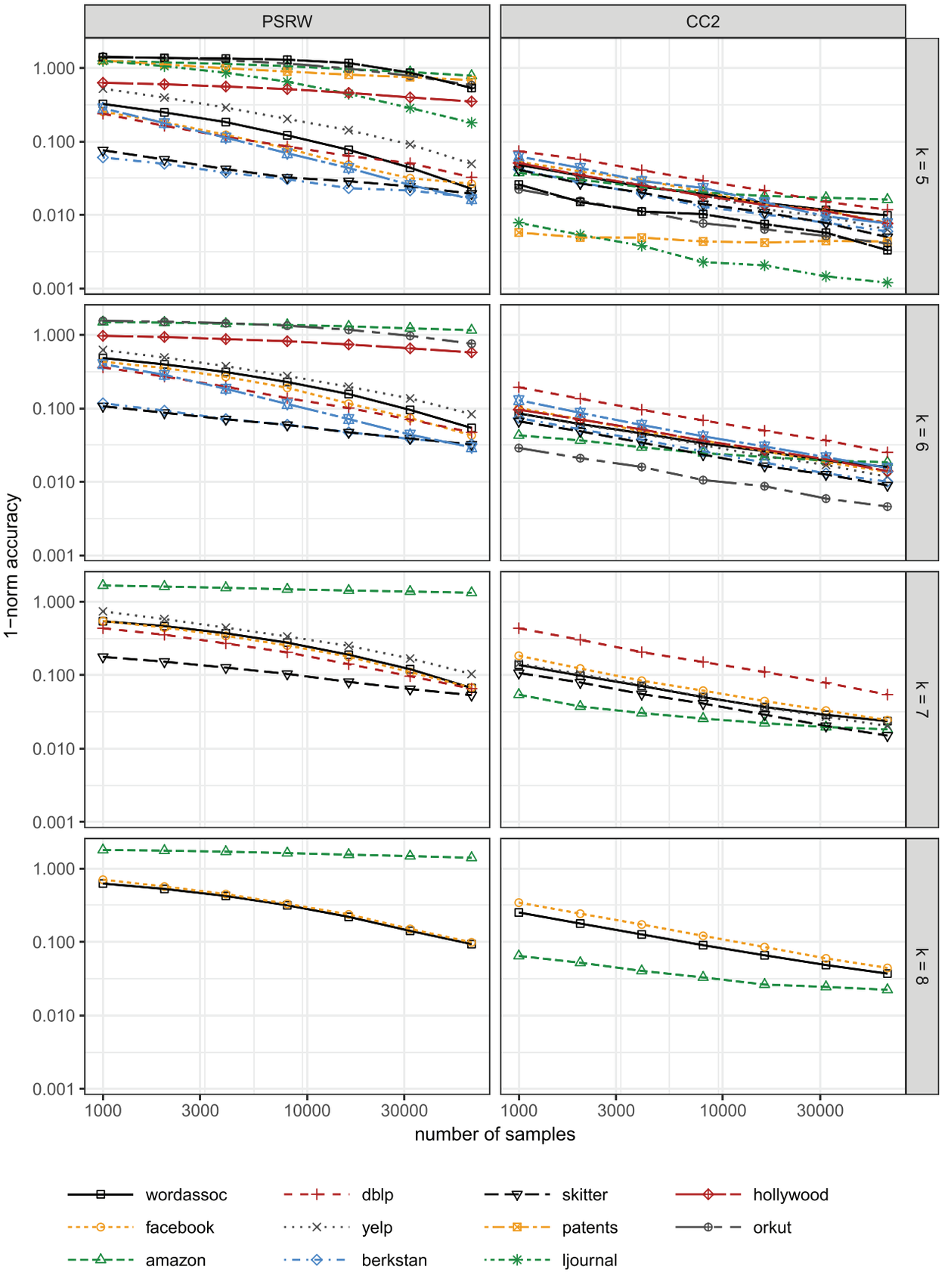


Fig. 2. Convergence of PSRW (left) and CC2 (right) to the ground truth distribution.

Table 2. Accuracy of Individual Graphlet Estimates: Average NRMSE of PSRW and CC2

	$1k$ samples				$10k$ samples				$100k$ samples			
	$k=5$	$k=6$	$k=7$	$k=8$	$k=5$	$k=6$	$k=7$	$k=8$	$k=5$	$k=6$	$k=7$	$k=8$
PSRW	3.84	1.93	1.61	1.23	5.52	2.11	1.98	3.11	5.74	2.61	2.25	3.76
CC2	0.19	0.27	0.44	0.47	0.11	0.15	0.26	0.29	0.07	0.09	0.15	0.17

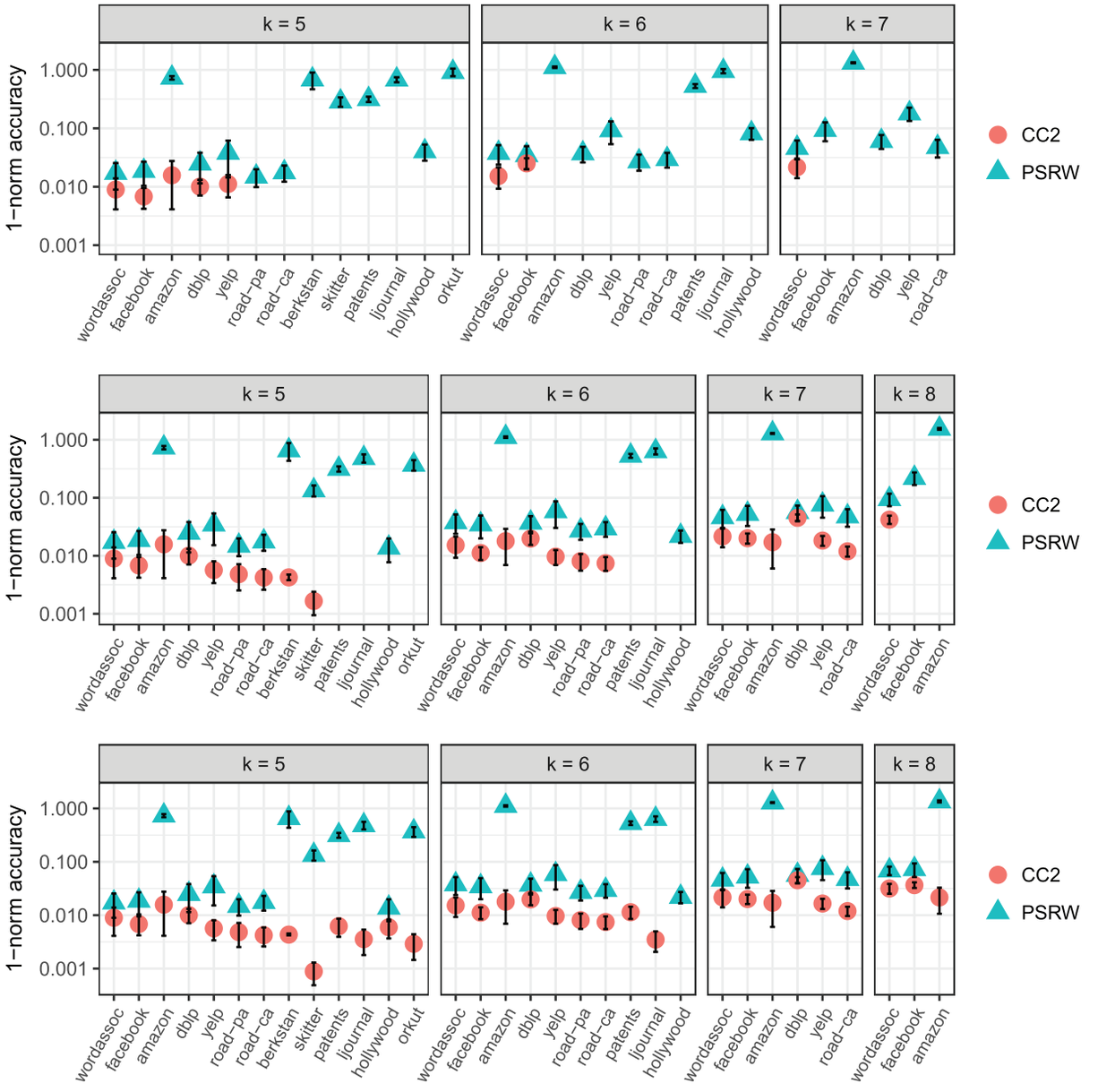


Fig. 3. Accuracy of CC2 and PSRW: average distance from ground truth, ± one standard deviation, after running for 10 seconds (top), 100 seconds (middle), 1,000 seconds (bottom). For CC2, the time includes the building phase.

running time. Figure 3 shows the average distance from ground truth, ± one standard deviation, measured after 10 seconds (top), 100 seconds (middle), and 1,000 seconds (bottom). For CC2, the time includes the building phase. Missing data means no run of the algorithm had produced at least 1,000 samples at that time. As we expected, in many cases CC2 takes longer than PSRW to produce samples. However, when it does so, it consistently provides higher accuracy. In fact, it

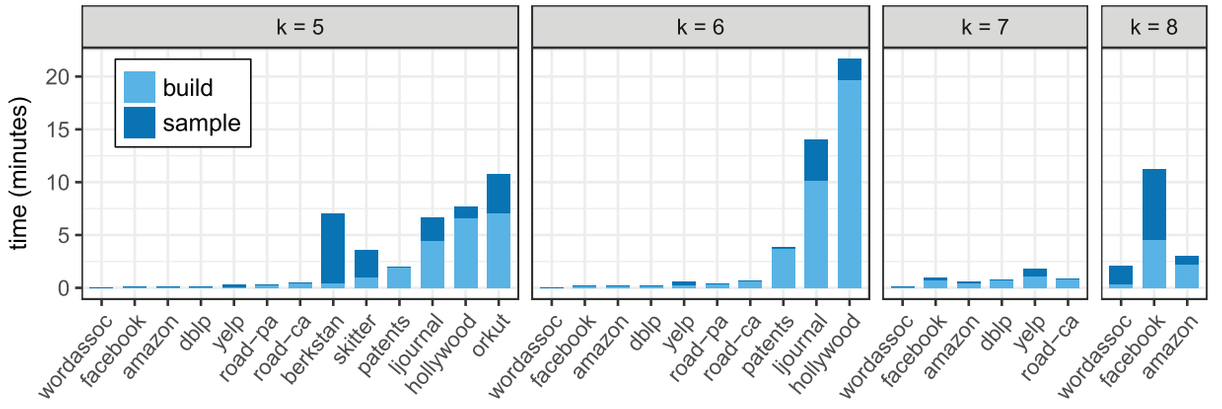


Fig. 4. Running time of CC2 (10^5 samples, average of 50 runs).

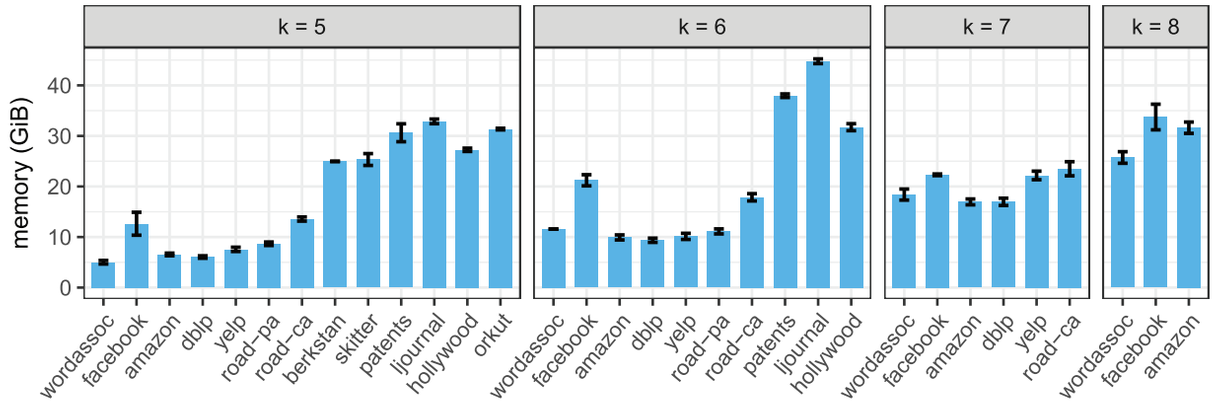


Fig. 5. Memory footprint of CC2 (10^5 samples, average ± one standard deviation of 50 runs).

seems that the instances where CC2 takes longer are those where PSRW yields more inaccurate estimates.

6.3 Performance Analysis of CC2

We analyze the performance of CC2 in terms of running time, memory footprint, and sampling speed. Running time was measured as described above, separately for the building phase and the sampling phase. The sampling speed is simply the ratio of the total number of samples, $100k$, to the sampling time. For memory, we report the value returned by `Runtime.getRuntime().totalMemory()` just after drawing the last sample; this is the JVM heap size used in that moment by our process. We note that the memory footprint depends on the behavior of JVM's Garbage Collector, and that one can reduce it at the expense of time. The measurements are summarized in Figures 4–6.

Two observations are in order. First, consistently with the complexity of CC2, the build time grows with the number of edges in the graph. Similarly, memory grows with the size of the graph, but in a more complex way (recall that the memory used by CC2 actually depends on the number of colorful graphlets). Furthermore, for the instances on which it ran successfully, CC2 managed to take $100k$ samples in a matter of minutes; and this includes instances with tens or hundreds of millions of edges, like `LiveJournal` or `Orkut`, for $k = 5$ and $k = 6$, and many others for $k > 6$. Even a non-optimized implementation of CC2, then, allows us to scale graphlet counting to a larger k than it was possible before, and on just a commodity machine; and an optimized rewriting in a

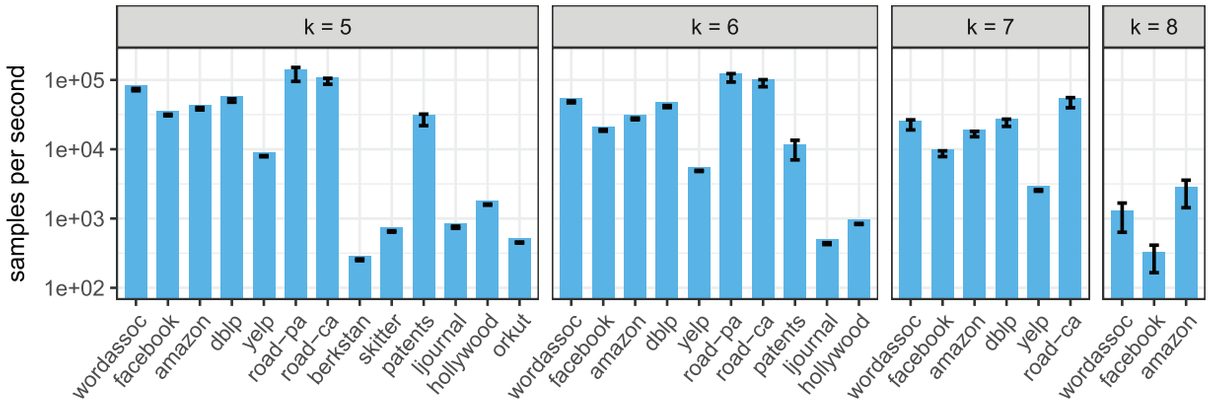


Fig. 6. Sampling speed of CC2 (10^5 samples, average ± one standard deviation of 50 runs).

memory-efficient language (e.g., C++) could lead to significantly better performance. Second, the performance of CC2 appears very stable. The relative standard deviation of the building phase time was below 0.1 for all instances except `WordAssoc` on $k = 5$ and $k = 6$, for which however the average building time was below 1.5seconds and thus inevitably noisy. The relative standard deviation of the memory footprint was less than 0.1 in all cases except `Facebook` for $k = 5$. The relative standard deviation of the sampling time was below 0.3 save for five instances.

Finally, a fact that may look surprising is the large variation in the sampling speed across different graphs, even for the same k (Figure 6). We believe this has to do with how we sample colored treelets in the graphs. Recall that, in our implementation, we sample a treelet by building on-the-fly the distribution of colorful subtrees over the neighbors of a node. If a graph contains a high-degree node, thus, sampling treelets containing it will be rather inefficient. In addition, that node will be likely included in a large fraction of all the graphlets, and therefore we will encounter it often in the sampling phase. Therefore, it is likely that sampling is slower in graphs of higher degree.

6.4 Scaling Graphlet Sampling Beyond Five Nodes

We conclude by showing the distribution of k-graphlets, for $k = 5, 6, 7, 8$, according to our ground truth (see above). Figure 7 shows the distributions, graphlets sorted from the highest to the lowest average frequency. For readability, we include only the 15 most frequent graphlets, and we exclude the distribution of the road graphs, `Road-CA` and `Road-PA`, which are similar and contain mostly trees. Figure 7 tells some interesting facts. First, the most frequent graphlets on average are K_{k-1} (a star) and the graphlet K^+_{k-2} obtained by attaching an extra node to a leaf of K_{k-2}. The three top 5-graphlets, the six top 6-graphlets, the eight top 7-graphlets, and the 14 top 8-graphlets are trees. In addition, many of these trees have a single *branching* node, i.e., a single node with degree more than two, which we conjecture is mapped to a high-degree node (a *hub*) of the graph. These properties might be rooted in social phenomena. Take for instance `LiveJournal` and `Amazon`; these two graphs contain many more copies of K_{k-1} than of K^+_{k-2}. A possible explanation is that the readers of the most frequently read blogs will tend not to know each other and, since the `LiveJournal` graph was made undirected, the neighbors of the high-degree nodes will tend to not induce many edges. The same can hold for `Amazon`, which is a co-purchasing network—in some sense, in both graphs users are buying products. On the other hand, look at the `Facebook` graph (containing only the users from the New Orleans area). Facebook imposes an upper bound on the number of friends of a user. This may increase the likelihood that two neighbors of a node are actually friends, so the ego-network of most nodes will tend to have a significant number of edges. This clearly decreases the fraction of induced K_{k-1}'s that can be found in this graph. Finally,

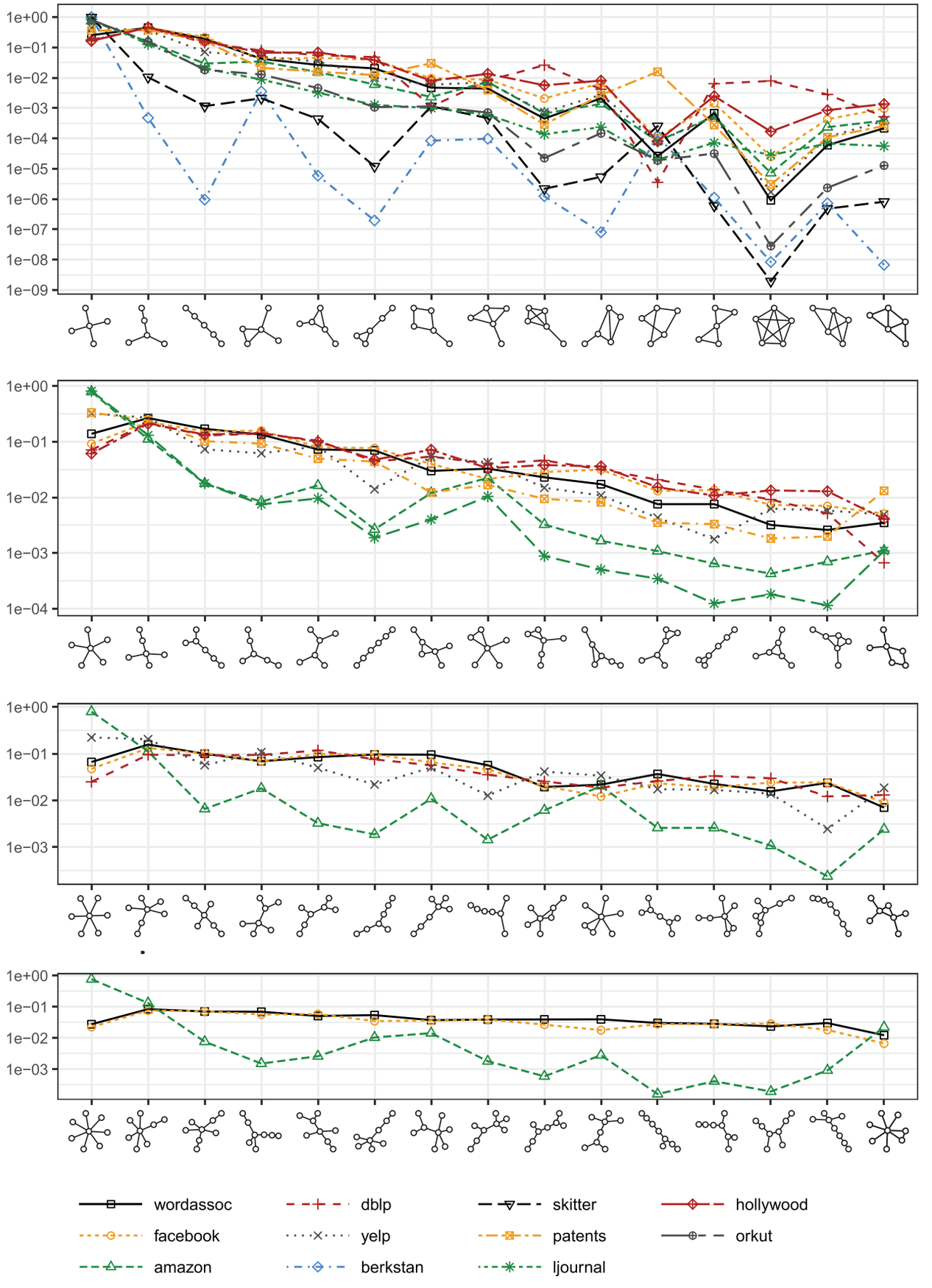


Fig. 7. Estimated frequency distribution of k-graphlets, from $k = 5$ (top) to $k = 8$ (bottom).

the Hollywood graph is the union of cliques, since the actors who starred in a movie are pairwise friends; hence, this graph does not contain many induced K_{k-1}'s.

Second, there seem to be different families of distributions followed by different graphs. This is especially evident in the 6-graphlets distribution, where on the one side, we have Amazon and LiveJournal, with skewed distributions that are strikingly similar; on the other hand, we have the remaining graphs, with much flatter distributions that are again quite close to each other. An intriguing question then one can explain this separation in terms of differences between the processes that have formed the networks.

Third, a surprising fact is that the graphlet distributions of WordAssoc and Facebook are extremely close for all $k = 5, 6, 7, 8$; but Facebook is a social graph, while WordAssoc is a graph resulting from a "free word association" experiment in psychology. Understanding whether such an almost perfect overlapping is just a casual correlation would be an interesting research direction.

Finally, in relation to the algorithm of [8], we note that 4/15 of the top 6-graphlets, 11/15 of the top 7-graphlets, and 14/15 of the top 8-graphlets do not contain a simple path on $(k-1)$ nodes and therefore would be unobservable with their algorithm (see Section 4.3).

7 CONCLUSIONS

In this article, we compared random walks and CC as the two most powerful algorithmic methods available to efficiently count graphlets in massive graphs. Our theoretical mixing time analysis cautions the blind use of random walks on real graphs, if statistical accuracy is paramount; on the other hand, we show that CC can be extended into a graphlet-sampling algorithm with statistical guarantees. In our experiments, CC appears to outperform the state-of-the-art random walk methods, yielding accurate counts for graphlets on up to eight nodes. Investigating the properties of graphs that lead to high mixing times for random walks is an interesting direction of future research. For CC, it will be interesting to see if the dynamic program table can somehow be compressed without sacrificing statistical guarantees much, which would make the method applicable for even larger graphlets; however, such an endeavour appears very challenging.

REFERENCES

[1] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Proceedings of the 15th IEEE International Conference on Data Mining (ICDM'15)*. 1–10. DOI: http://dx.doi.org/10.1109/ICDM.2015.141

[2] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *Journal of the ACM* 42, 4 (Jul. 1995), 844–856. DOI: http://dx.doi.org/10.1145/210332.210337

[3] Mansurul A. Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. GUISE: Uniform sampling of graphlets for large graph analysis. In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM'12)*. 91–100. DOI: http://dx.doi.org/10.1109/ICDM.2012.87

[4] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. IW3C2, Geneva, Switzerland, 587–596. DOI: http://dx.doi.org/10.1145/1963405.1963488

[5] P. Boldi and S. Vigna. 2004. The webgraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*. IW3C2, Geneva, Switzerland, 595–602. DOI: http://dx.doi.org/10.1145/988672.988752

[6] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting graphlets: Space vs time. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM'17)*. ACM, New York, NY, 557–566. DOI: http://dx.doi.org/10.1145/3018661.3018732

[7] V. T. Chakaravarthy, M. Kapralov, P. Murali, F. Petrini, X. Que, Y. Sabharwal, and B. Schieber. 2016. Subgraph counting: Color coding beyond trees. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. 2–11. DOI: http://dx.doi.org/10.1109/IPDPS.2016.122

[8] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John C. S. Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment* 10, 3 (Nov. 2016), 253–264. DOI: http://dx.doi.org/10.14778/3021924.3021940

[9] F. Chung. 2007. Four proofs for the Cheeger inequality and graph partition algorithms. In *Proceedings of the Integrated Community Case Management (ICCM'07)*.

[10] Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. 2015. Detecting and counting small pattern graphs. *SIAM Journal of Discrete Mathematics* 29, 3 (Feb. 2015), 1322–1339. DOI: http://dx.doi.org/10.1137/140978211

[11] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *Proceedings of the 16th IEEE International Conference on Data Mining (ICDM'16)*. 181–190.

[12] Mark Jerrum and Kitty Meeks. 2015. The parameterised complexity of counting connected subgraphs and graph motifs. *Journal of Computer and System Sciences* 81, 4 (Nov. 2015), 702–716. DOI: http://dx.doi.org/10.1016/j.jcss.2014.11.015

[13] Madhav Jha, C. Seshadhri, and Ali Pinar. 2013. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, New York, NY, 589–597. DOI: http://dx.doi.org/10.1145/2487575.2487678

[14] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. IW3C2, Geneva, Switzerland, 495–505. DOI: http://dx.doi.org/10.1145/2736277.2741101

[15] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD'05)*. ACM, New York, NY, 177–187. DOI: http://dx.doi.org/10.1145/1081870.1081893

[16] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. IW3C2, Geneva, Switzerland, 695–704. DOI: http://dx.doi.org/10.1145/1367497.1367591

[17] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. 2009. *Markov Chains and Mixing Times*. American Mathematical Society. xviii+371 pages.

[18] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW'17)*. IW3C2, Geneva, Switzerland, 1431–1440. DOI: http://dx.doi.org/10.1145/3038912.3052597

[19] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding network motifs using MCMC sampling. In *Proceedings of the 6th Workshop on Complex Networks (CompleNet'15)*. Springer International Publishing, 13–24.

[20] G. M. Slota and K. Madduri. 2013. Fast approximate subgraph counting and enumeration. In *Proceedings of the 42nd International Conference on Parallel Processing (ICPP'13)*. 210–219. DOI: http://dx.doi.org/10.1109/ICPP.2013.30

[21] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. IW3C2, Geneva, Switzerland, 607–614. DOI: http://dx.doi.org/10.1145/1963405.1963491

[22] Ngoc Hieu Tran, Kwok Pui Choi, and Louxin Zhang. 2013. Counting motifs in the human interactome. *Nature Communications* 4 (Aug. 2013), Article 2241.

[23] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. 2009. DOULION: Counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, New York, NY, 837–846. DOI: http://dx.doi.org/10.1145/1557019.1557111

[24] W. T. Tutte. 2001. *Graph Theory*. Cambridge University Press. https://books.google.it/books?id=uTGhooU37h4C.

[25] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM Workshop on Online Social Networks (WOSN'09)*. ACM, New York, NY, 37–42. DOI: http://dx.doi.org/10.1145/1592665.1592675

[26] M. D. Vose. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering* 17, 9 (Sep. 1991), 972–975. DOI: http://dx.doi.org/10.1109/32.92917

[27] Pinghui Wang, John C. S. Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data* 9, 2 (Sep. 2014), Article 8, 27 pages. DOI: http://dx.doi.org/10.1145/2629564

[28] Pinghui Wang, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Junzhou Zhao, Jing Tao, and Xiaohong Guan. 2016. A fast sampling method of exploring graphlet degrees of large directed and undirected graphs. arXiv:1604.08691

[29] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2018), 73–86. https://doi.org/10.1109/TKDE.2017.2756836

[30] Virginia Vassilevska Williams and Ryan Williams. 2013. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing* 42, 3 (2013), 831–854. DOI: http://dx.doi.org/10.1137/09076619X

[31] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (Jan. 2015), 181–213. DOI: http://dx.doi.org/10.1007/s10115-013-0693-z

[32] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe. 2010. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *Proceedings of the 39th International Conference on Parallel Processing (ICPP'10)*. 594–603. DOI: http://dx.doi.org/10.1109/ICPP.2010.67

Received June 2017; revised February 2018; accepted February 2018

EXTRACT and REFINE: Finding a Support Subgraph Set for Graph Representation

Kuo Yang
University of Science and Technology of China
Hefei, China
yangkuo@mail.ustc.edu.cn

Zhengyang Zhou*
Suzhou Institute for Advanced Research, University of Science and Technology of China
Suzhou, China
zzy0929@mail.ustc.edu.cn

Wei Sun
University of Science and Technology of China
Hefei, China
sunwei3@mail.ustc.edu.cn

Pengkun Wang
Suzhou Institute for Advanced Research, University of Science and Technology of China
Suzhou, China
pengkun@mail.ustc.edu.cn

Xu Wang
University of Science and Technology of China
Hefei, China
wx309@mail.ustc.edu.cn

Yang Wang*
University of Science and Technology of China
Hefei, China
angyan@ustc.edu.cn

ABSTRACT

Subgraph learning has received considerable attention in its capacity of interpreting important structural information for predictions. Existing subgraph learning usually exploits statistics on predefined structures e.g., node degrees, occurrence frequency, to extract subgraphs, or refine the contents via only capturing label-relevant information with node-level sampling. Given diverse subgraph patterns, and mutual independence with local correlations on graphs, current solutions on subgraph learning still have two limitations in extraction and refinement stages. 1) The universality of extracting substructure patterns across domains is still lacking, 2) node-level sampling in refinement will distort the original local topology and none explicit guidance eliminating redundant information contribute to inefficiency issue. In this paper, we propose a unified subgraph learning scheme, Poly-Pivot Graph Neural Network (P2GNN) where we designate the centric node of each subgraph as the pivot. In the extraction stage, we present a general subgraph extraction principle, i.e., *Local Asymmetry* between the centric and affiliated nodes. To this end, we asymmetrically model the similarity between each pair of nodes with random walk and quantify mutual affiliations in Affinity Propagation architecture, to extract subgraph structures. In the refinement, we devise a subgraph-level exclusion regularization to squash the target-independent information by considering mutual relations across subgraphs, cooperatively preserving a support set of subgraphs and facilitating the refinement process for graph representation. Empirical experiments on diverse web and biological graphs reveal 1.1%~7.3% improvements against best baselines, and visualized case studies prove the universality and interpretability of our P2GNN.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Learning latent representations**.

KEYWORDS

Graph Neural Network; Subgraph Extraction; Local Asymmetry; Subgraph Refinement

ACM Reference Format:
Kuo Yang, Zhengyang Zhou, Wei Sun, Pengkun Wang, Xu Wang, and Yang Wang. 2023. EXTRACT and REFINE: Finding a Support Subgraph Set for Graph Representation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599339

1 INTRODUCTION

Graph-structured data is ubiquitous across various real-world scenarios, ranging from social networks [30, 46], citation networks [12, 50], to molecular graphs [17, 42]. Recently, Graph Neural Networks (GNNs) have achieved great success to materialize diverse downstream tasks through a sparse message-passing process. However, there is a growing recognition that the message-passing paradigm has inherent limitations [22], i.e., the expressiveness of message passing in traditional GNNs is upper bounded by the first order Weisfeiler-Leman (1-WL) isomorphism test [39]. The limitation of GNNs motivates researchers to explore more expressive architecture. Most notably, subgraph-level explanations are more intuitive and useful, as subgraphs are simple building blocks of complex graphs and concerned with the functionalities of graphs. As a result, there is a growing trend to extract the subgraph patterns from original data for more efficient and accurate predictions [15].

Even flourishing, there remains two open issues in subgraph-based methods. 1) How to clearly extract major substructure patterns in a graph which mostly explain the predictions [32, 47], 2)

*Yang Wang and Zhengyang Zhou are corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '23, August 6–10, 2023, Long Beach, CA, USA.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599339

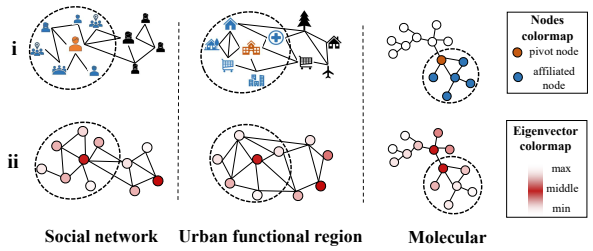


Figure 1: Examples of Local asymmetry in three different scenarios by contrast observation and spectral theory. (i) The asymmetric status between the pivot nodes and the affiliated nodes forms a stable substructure. (ii) The low-frequency eigenvectors often resonate with central nodes in local structures.

how to remove non-useful information, and obtain subgraphs that maximally benefit final prediction for better generalization [18, 26]. Therefore, finding a local structure descriptor to accommodate majority of subgraphs and reducing the target-detrimental information to achieve the minimal but sufficient subgraphs, are two main targets of subgraph-based methods.

Regarding substructure extractions, the mainstream solutions often predefined the subgraph structure with prior knowledge, such as EgoNets in social networks [41, 46], high-frequency functional groups in molecular graphs [19, 20]. However, these criterions of subgraph discovery require knowledge and experiences on specific domains, and consequently hampers the universality in extracting different substructural patterns across various domains [3, 4, 19, 36, 48, 49]. Concerning information refinement, the pioneering Graph Information Bottleneck (GIB) explores the information theory to squash and refine information with edge-wise sampling or node-level dropping [11, 37, 43, 44]. Unfortunately, these existing refinement solutions suffer two limitations. First, refining the whole graph on node levels inevitably distorts the original local topology and corrupt the node-wise correlations, leading to the intervention on following refinements. Then, without an explicit guidance for redundant information elimination, existing refinement solutions only exploit the labels to preserve the minimal information, which is inefficient for their convergence processes. To this end, we can summarize two issues that hinder the existing subgraph learning from achieving universal and robust graph representations, i.e., 1) the predefined substructure derived from domain knowledge usually **lacks universality in subgraph extraction**, 2) sampling on node levels and lacking guidance of redundance elimination introduce **the distorted and inefficient refinement process**. Therefore, how to find a support subgraph set that is minimal but sufficient for learning tasks, is still an open challenge. Fortunately, the following two observations can potentially facilitate to tackle above challenges.

Firstly, to address the universality limitation of previous methods, we discover that there exists a special common pattern of *Local Asymmetry*, shared across the majority subgraphs. As shown in Figure 1(i), subgraphs in web social networks are usually forged by an EgoNet centering with Internet celebrity, which also obeys asymmetric influencer-follower relations, urban functional regions are dominant by the density of major functional POIs but also with several affiliated sites, and the functional groups in molecules are usually identified with the statistical frequencies where it also can be interpreted by anisotropical inter-atomic attractions. Despite the diversity of subgraph formation principles, we can still summarize a common property that contributes to subgraphs, i.e., the *Local Asymmetry* between the centric and affiliated nodes. Quantitatively, consider the well-known property of *Spectral Theorem* discriminating between different graph structures and substructures [22], we exploit eigenvalues to prove our observations. In practices, the k-smaller eigenvectors depict smooth encoding coordinates and reveal a comprehensive electric potential field of neighboring nodes [13, 35]. Such principle motivates us to obtain Figure 1(ii), which theoretically supports our *Local Asymmetry*. More details are provided in Appendix C.

Secondly, the content refinement is usually realized by Information Bottleneck [37], which is prone to be fragile and inefficient as it performs on node levels and lacks guidance for actively eliminating redundance. Therefore, we have two observations about the refinement of subgraphs. 1) nodes in a subgraph exhibit clustering effects and a large real-world graph usually consists of multiple subgraphs with each subgraph accounting for specific properties [2, 5, 48]. 2) on inter-subgraph relationships, each subgraph should inherently reveal remarkable independence where nodes in a specific subgraph can be deemed as a collectivity and such individual collectivity is informative to represent all its members. These two observations, which are considered as the local dependence within subgraphs and inter-independence among subgraphs [37], can advance the refinement towards a more robust and efficient manner. Motivated by the local dependence, simultaneously discovering a bag of subgraphs and improving sampling from node to subgraph levels can promisingly avoid the corruptions of local topology with redundant subgraphs removed integrally. Considering the inefficiency induced by none guidance of information elimination, the inter-independence potentially provides a gathering and dispersion principle for excluding redundant information and thus facilitating the sampling process.

Present work. In this paper, to address the challenges of both general subgraph pattern extraction and subgraph-level refinements, we propose a novel and general subgraph learning scheme, Poly-Pivot Graph Neural Network (P2GNN) where we designate the centric node of each subgraph as the pivots in graphs. Our P2GNN composes of an extractor and a refiner, which extract substructure and refine subgraph information, respectively. For the extractor, we propose a substructure extraction strategy *Hitpath* based on the principle of *Local Asymmetry*. Firstly, we design an improved random walk, to measure the node-wise asymmetry by the differences between pairwise random walk distances, preserving both local topology and feature correlation. Secondly, to adaptively discover diverse subgraphs centered with different pivots, we take Affinity Propagation (AP) clustering as a basic framework and receives node-wise asymmetric similarity from random walk, where the AP clustering enjoys the nice property of modeling interactive relationships between pivot nodes and underlying affiliated points without pre-defining the number of subgraphs. In refiner, we refine information on the subgraph level to maximally avoid the corruption of

local topology, and devise a novel Exclusion-based regularization to actively obtain support subgraphs. We also design *Information Shift Method (ISM)* to verify whether the subgraphs learned from P2GNN are with the nice support property for prediction. Specifically, we propose to rank all the subgraphs based on sampling confidence and realize the information shift with probability reassignment. With our proposed *ISM*, we can easily exploit the prediction performance trends to explore the support property of discovered subgraph set.

Our main contributions are summarized as:

• We emphasize the universality and efficiency of subgraph learning, and summarize a general principle of substructure extraction *Local Asymmetry*, which is further justified by spectral theory.

• We present a two-stage subgraph learning scheme P2GNN, which composes of an extractor and a refiner. The analysis of *Hitpath* provides theoretical guarantee on modeling the asymmetric relationship between nodes.

• Extensive experimental results reveal that our P2GNN excels best baselines, where our *Local Asymmetry* can exactly cover subgraph patterns across datasets. A novel evaluation method *ISM* is designed to verify the support property of refined subgraphs.

2 PRELIMINARIES AND DEFINITIONS

Graph. Let $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ denote a graph with $|\mathcal{V}|$ nodes. In particular, $\mathcal{V} = \{v_1, v_2, ..., v_{|\mathcal{V}|}\}$ is the node set, $\mathcal{E} = \{e_{ij} : \langle v_i, v_j \rangle\}$ is the edge set with each pair of v_i and v_j connected, and $\boldsymbol{W} = \{w_{v_i v_j}\}$ denotes the weight of the edge $\langle v_i, v_j \rangle$. Besides, $d(v_i)$ denotes the degree of node v_i and $\mathcal{N}(v_i)$ is the set of v_i neighbor nodes.

Subgraph. Given a graph G, it can be decomposed into a set of M subgraphs $G_S = \{G_S^1, G_S^2, \ldots, G_S^M\}$, where M is adaptively variable to different G. Each disentangled subgraph is denoted as $G_S^i = (\mathcal{V}_S^i, \mathcal{E}_S^i)$. All subgraphs in G_S must satisfy the following two conditions: (1) $\mathcal{V} = \mathcal{V}_S^1 \cup \mathcal{V}_S^2 \cup \ldots \cup \mathcal{V}_S^M$, (2) $\mathcal{V}_S^i \cap \mathcal{V}_S^j = \emptyset$ $(\forall i, j \in [1, M],\ i \neq j)$.

Random Walk. It is a time-reversible Markov chain [24]. Given a graph G, and a starting node v_i, in the random walk, v_i will first jump to one of its neighbors with a transition probability, i.e., $p(v_i \rightarrow v_j) = \frac{1}{d(v_i)}(v_j \in \mathcal{N}(v_i))$, and then it will successively jump to high-order neighbors with corresponding probability. The sequence of nodes v_i walking through is defined as the walk path on the graph, and $\mathcal{H}(v_i, v_j)$ is denoted as walking distance from v_i to v_j, which is the expected distance that v_i first arrives at v_j.

Problem Definition. The goal of our work is to perform a systematical subgraph learning that finds a support subgraph set for graph representation. Given a graph G, first, we are expected to learn a general subgraph extraction model $P_\theta(G_S|G)$ to obtain a subgraph set $G_S = \{G_S^1, G_S^2, \ldots, G_S^M\}$. Second, we further impose the subgraph-level refinement model $P_\phi(G_S^*|G_S)$ on G_S to derive a support subgraph set G_S^* that consists of minimal but sufficient subgraphs, simultaneously improving the performance and interpretation of graph-level classification.

3 POLY-PIVOT GRAPH NEURAL NETWORK

Our P2GNN is a two-stage subgraph learning scheme, consisting of a Local Asymmetry-based substructure extractor and a subgraph-level information refiner via mutual exclusions. The whole technical process is illustrated in Figure 2.

3.1 Substructure Extractor via Local Asymmetry

The goal of the first stage is to generalize the *Local Asymmetry* principle to extract subgraph structure, and to disentangle original graph into a subgraph set. Inspired by the analogy between node clustering and subgraph extraction, we propose a subgraph extractor based on AP algorithm, which employs HitPath as the asymmetric similarity measurement.

As shown in Figure 1, we discover that almost all subgraphs, even graphs across domains, share two rules, i.e., 1) members in a subgraph tend to be physically neighboring, and 2) there is the disparity of statuses among different members where some pivots are prone to attract affiliated nodes to formulate a subgraph community. We summarize such common properties as *Local* and *Asymmetry* and formalize *Local Asymmetry* as a novel principle for subgraph discovery. Given these observations, it is not proper to just mimic previous predefined substructure extraction. We then introduce an approach on how to implement the *Local Asymmetry* principle in graph.

A vital cognition about *Local Asymmetry* is that the cluster formed by the asymmetric attraction of the pivot nodes to the member nodes constitutes the subgraph, which is analogous to clustering of nodes. However, our task is significantly more complex, i.e. irregularity of graph structure leads to uncertain numbers of subgraphs, the determination of the pivot nodes is complicated, and asymmetric metrics are difficult to quantify. These factors hinder applying traditional symmetric clustering strategy to extract subgraphs. Fortunately, Affinity Propagation (AP) clustering algorithm based on message passing mechanism has many excellent properties, which lights up our idea. AP algorithm does not need to prespecify the number of clusters, can find the centers adaptively, and more importantly supports asymmetric similarity matrix. Note that these satisfactory properties all depend on appropriate similarity criteria. Given these evidences, the core challenge is how to design a similarity criteria according to *Local Asymmetry* principle.

Technically, *Local* can be easily implemented by calculating the hop distance $hop(v_i, v_j)$ between pairwise nodes $\langle v_i, v_j \rangle$, while *Asymmetry*, which is expected to quantify the status disparity among members, is too abstract to describe. Random Walk enjoys the nice property of modeling node-wise asymmetric statuses [28, 29]. Nodes in a random walk will stochastically jump to one of its neighbors with a transition probability inversely proportional to the node degree. In this way, the asymmetric statuses induced by the disparate local topology can be exactly captured. However, we argue that traditional random walk suffers two limitations when adopted in our subgraph discovery. First, since node features usually play significant roles in forming subgraphs, these solutions of Random Walk only consider the topology but fail to involve the semantic correlations induced by node features. Second, computing the expectation of walking distances introduces the inefficiency issue [33]. Therefore, we expand a weighted random walk, *HitPath*, by two modifications on the traditional one.

First, to accommodate the semantic similarity of node features, the walking path in our *HitPath* is not only determined by the node-wise connectivity, but also the disparity between node-specific representations. Specifically, consider a walking path $p = (v_{p_0}, v_{p_1}, \ldots)$

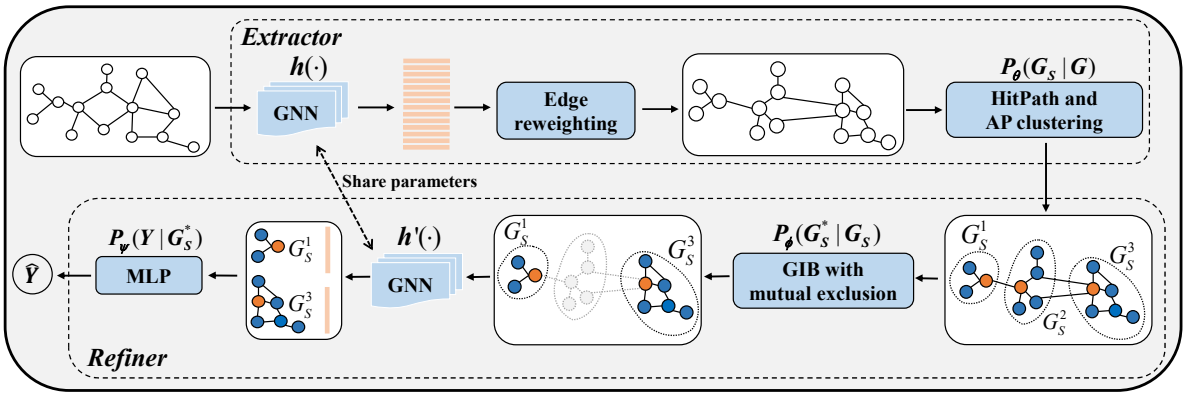


Figure 2: The architecture of our two-stage P2GNN. In the substructure extraction stage, the node-wise associations on both topology and features are explored by *HitPath* to obtain the re-weighted G. Then all nodes are disentangled into a subgraph set G_S. In the subgraph refinement stage, the subgraph-level exclusions are imposed to GIB for facilitating learning process. Finally, P2GNN outputs a set of support subgraphs G_S^* and performs graph classifications. Note that $h(\cdot)$ and $h'(\cdot)$ share the same GNN encoder. The orange nodes are pivots of subgraphs while the blue nodes are affiliated member points.

in graph G , each neighboring transition pair $\langle v_{p_i}, v_{p_{i+1}} \rangle \in \mathcal{E}$ consists of the sequential walking steps. Then we revise the degree-based transition probability into a weighted distance. For node v_i, the probability jumping to v_j in its neighborhood is,

$$P(v_i \to v_j) = \frac{w_{v_i v_j}}{\sum\limits_{v_k} w_{v_i v_k}} (v_k, v_j \in \mathcal{N}(v_i)) \tag{1}$$

where $w_{v_i v_j}$ is instantiated as the Euclidean distance between their representations learned by a GNN-based encoder $h(\cdot)$,

$$w_{v_i v_j} = ||h(v_i) - h(v_j)||^2 \tag{2}$$

For the transition probability $P(v_i \to v_j)$, the number of summed terms in the denominator implies the node degree of v_i and thus capturing the topological property, while $w_{v_i v_j}$ serves as a node-wise weighted distance, encapsulating the proximity of feature information. Due to the potentially disparate local neighborhood environments (both topology and feature distributions over neighbors) of different nodes, the node-wise asymmetric relationship can be well characterized by our modified transition probability.

Second, to alleviate the inefficiency issue, a restart strategy along with truncated walking steps is devised to approximate the walking distance $\mathcal{H}$. Given the path p starting at v_{p_0}, the hitting distance from v_{p_0} to any other node $v_{p_k} \in p$ is truncated by a fixed step of l, i.e., walking distance makes sense only when v_{p_0} is accessible to v_{p_k} within l steps, and thus p is modified as $p = (v_{p_0}, v_{p_1}, \ldots, , v_{p_l})$. We then also impose a fixed restart times K, to accommodate the tradeoff between performance and efficiency in *HitPath*.Therefore, we have the one-time hitting distance $H(v_0, v_k)$ from v_{p_0} to any other node v_{p_k} by summing step-wise feature-based distances along the path p,

$$H(v_0, v_k) = \begin{cases} \sum\limits_{v_i \in (v_0, \ldots, v_{k-1})} w_{v_i v_{i+1}}, v_k \in p \\ \sigma, v_k \notin p \end{cases} \tag{3}$$

where a large upper limit σ will be set as the hitting distance if v_k is out of range of path p. By repeating walking for K times, we can obtain the modified random walking distance in our *HitPath* (directional walking distance) $\tilde{\mathcal{H}}$ between nodes v_i and v_j,

$$\tilde{\mathcal{H}}(v_i, v_j) = HitPath(v_i, v_j) = \frac{1}{K} \sum_{k=1}^{K} H_k(v_i, v_j) \tag{4}$$

Such step truncation in *HitPath* can naturally eliminate the influences from nodes within high-order neighborhoods, and preserve the localization constraint to satisfy *Local Asymmetry* principle.

Furthermore, in the process of subgraph discovery, what we actually focus on is not distinguishing the exact nodes with higher status, but to identify a clustering of nodes with locally asymmetric relationships. In this way, we can leverage the difference between the asymmetric $HitPath(v_i, v_j)$ and $HitPath(v_j, v_i)$ to characterize the disparity of statuses between v_i and v_j.

Finally, considering the *Local* property with hop distances, we derive the quantitative node-wise relationship between v_i and v_j under the *Local Asymmetry* principle,

$$s(v_i, v_j) = \gamma \cdot \mathrm{e}^{-\mathrm{hop}(v_i, v_j)} + (1 - \gamma) \cdot ||HitPath(v_i, v_j) - HitPath(v_j, v_i)||^2 \tag{5}$$

This $s(v_i, v_j)$ can also be viewed as a node-wise similarity measurement, where γ balances the importance between the former term for localization and the latter one for asymmetry. Noted that there are two major hyper-parameters in *HitPath*. We empirically let the truncated step l be $\frac{|\mathcal{V}|}{2}$, while the balance coefficient γ be 0.8, and then fine-tune these settings with experiments. More settings can be found in Appendix B.

3.2 Mutual Exclusion-based Subgraph Refiner

In this subsection, we present our subgraph refinement model $P_\phi(G_S^*|G_S)$ as well as the classifier $P_\psi(Y|G_S^*)$ for downstream tasks, where G_S^* is the set of support subgraphs refined from G_S. Recall that we have pointed out that conventional node-level refinement will potentially distort the original local topology and lead to intervention on following refinements [32]. What's more, existing refinement solutions only exploit the label information to preserve the

minimal information but neglect any other guidance for redundant information elimination, leading to their inefficient convergence.

To break up above two dilemmas, we devise a subgraph-level refiner. From the perspective of statistics, we are expected to obtain a series of independent and label-relevant subgraphs, which shares similar goals of GIB [37]. However, GIB-based refinements usually seek for one subgraph in a given graph by node-level sampling [43, 44], which are trivially dropped into the topology corruption. In this work, we make modifications to GIB on two aspects, 1) expand the sampling unit of nodes into subgraphs and 2) impose a mutual exclusion regularization to decentralize each subgraph and facilitate the learning process.

Given a graph G, a discovered subgraph set G_S and the label Y of G, our subgraph-level GIB is expected to find a support subgraph set G_S^*, which not only squashes the subgraph information towards labels but also constrains the least number of reserved subgraphs. It is formulated as,

$$\min_{G_S^*} -I(Y, G_S^*) + \beta I(G_S, G_S^*) \tag{6}$$

where β is set as 1 according to common practice [37, 43, 44]. We then elaborate the implementation of above GIB. For the first term $I(Y, G_S^*)$, we exploit the tractable lower bound obtained by [44], and realize it with the standard cross-entropy loss. In particular, this cross-entropy loss is also minimized to materialize the classifier $P_\psi(Y|G_S^*)$ for downstream tasks. For the second term $I(G_S, G_S^*)$, we introduce a variational estimator $q(G_S)$ for the marginal distribution $p(G_S)$, and derive the variational upper bound with KL-divergence, i.e., $I(G_S, G_S^*) \leq KL\left(P_\phi(G_S^*|G_S)||q\left(G_S\right)\right)$ [1]. Actually, since there is no premise or prior knowledge and further and an inaccurate $q(G_S)$ will terribly mislead the refinement and deteriorate the efficiency, finding an accurate variational approximation of $p(G_S)$ is intractable. To explicitly guide eliminating redundant information on subgraph-level sampling, besides a KL-divergence objective, we further introduce another principle of mutual exclusion on subgraphs. This principle interprets a generally recognized but less exploited knowledge on subgraph refinement, i.e., the node representations within the same subgraph should be as close as possible while the representation disparity among different subgraphs should be large. Then we consider this mutual exclusion as a regularization term in our objective to jointly optimize a decentralized and effective set of subgraphs. Technically, let $p_{sub}(\{v_*\} \in \widetilde{G}_S^t)$ be the probability that all the nodes v_* in a tentative $\widetilde{G}_S^t$ forming an exact subgraph, and $p_{sub}(\widetilde{G}_S^p, \widetilde{G}_S^q)$ be the joint occurrence probability of both subgraphs $\widetilde{G}_S^p$ and $\widetilde{G}_S^q$. Therefore, the support subgraph set G_S^* considering the mutual exclusion are expected to obtain the maximal $\sum_{G_S^i \in G_S^*} p_{sub}(G_S^i)$ and the minimal $\sum_{G_S^i \in G_S^*} \sum_{G_S^-} p_{sub}(G_S^i, G_S^-)$, where G_S^- denotes any other subgraph except G_S^i. We can formally derive the regularization objective of mutual exclusion as,

$$\begin{aligned} ME(G_S^*) &= \mathbb{E}_{p_{sub}(G_S^i)} \mathbb{E}_{p_{sub}(G_S^-)} [p_{sub}(G_S^i, G_S^-) - p_{sub}(G_S^i)] \\ &= \mathbb{E}_{p_{sub}(G_S^i)} \mathbb{E}_{p_{sub}(G_S^-)} [\log \sum_{G_S^i \subset G_S^*} \exp(h(G_S^i)^T h(G_S^-)) \\ &\quad - \log \sum_{(v_k, v_m) \sim G_S^i} \exp(h(v_k)^T h(v_m))] \end{aligned} \tag{7}$$

In this way, the second term $I(G_S, G_S^*)$ can be realized by integrating both variational approximation of KL-divergency and mutual exclusion regularization,

$$I(G_S, G_S^*) = KL\left(P_\phi(G_S^*|G_S)||q(G_S)\right) + ME(G_S^*) \tag{8}$$

Optimization objective. Considering the mutual exclusion regularization, we jointly optimize the integrated objectives of P2GNN,

$$\min_{\theta,\phi,\psi} -\mathbb{E}(P_\psi(Y|G_S^*)) + \beta \mathbb{E}\left[KL\left(P_\phi(G_S^*|G_S)||q\left(G_S\right)\right) + ME(G_S^*)\right] \tag{9}$$

So far, we can finally obtain the refined subgraph set G_S^* from the discovered subgraph set G_S.

3.3 Detailed Implementation of P2GNN

P2GNN includes an extractor $P_\theta(G_S|G)$, a refiner $P_\phi(G_S^*|G_S)$ and a classifier $P_\psi(Y|G_S^*)$ for prediction. Besides, a support property evaluation *ISM* is also designed to explore whether the discovered subgraph set G_S^* satisfies the expected support property. We introduce the detailed implementation of P2GNN as follows:

Substructure extractor $P_\theta(G_S|G)$. Our substructure extractor is instantiated as the combination of *HitPath* and AP clustering. The AP clustering exploits two metrics of "Responsibility" and "Availability" [16] to jointly measure the interactive relationships between pivot nodes and potentially affiliated points by considering influences from other nodes, which opportunely match our *Local Asymmetry* principle. Concretely, consider the potential pivot node as v_k and other affiliated point as v_i. The "Responsibility" $r\left(v_i, v_k\right)$ reflects the accumulated evidence for how well-suited the node v_k is to serve as the pivot for v_i, while "Availability" $a\left(v_i, v_k\right)$ reflects the accumulated evidence for how appropriate v_i chooses v_k as its pivot. In this way, by exploiting the *Local Asymmetry*-based node-level relationship measurement $s(\cdot,\cdot)$, the responsibility $r\left(v_i, v_k\right)$ and availability $a\left(v_i, v_k\right)$ in AP clustering can be respectively formalized by,

$$r(v_i, v_k) = \begin{cases} s(v_i, v_k) - \max\limits_{k' \neq k} \{a(v_i, v_{k'}) + s(v_i, v_{k'})\}, v_i \neq v_k \\ s(v_i, v_k) - \max\limits_{k' \neq k} \{s(v_i, v_{k'})\}, v_i = v_k \end{cases} \tag{10}$$

$$a(v_i, v_k) = \begin{cases} \min\{0, r(v_k, v_k) + \sum\limits_{v_j \neq v_i} \max\{r(v_j, v_k), 0\}\}, v_i \neq v_k \\ \sum\limits_{v_j \neq v_k} \max\{r(v_j, v_k), 0\}, v_i = v_k \end{cases} \tag{11}$$

Based on the responsibility and availability iteratively updated by Equation 12, we can obtain a set of pivot nodes $\{v_{c_1}, v_{c_2}, ..., v_{c_M}\}$, and subsequently construct the corresponding subgraphs $G_S = \{G_S^1, G_S^2, \ldots, G_S^M\}$. The node set $\mathcal{V}_S^i$ consists of the node membership in i-th subgraph.

$$\max_{v_k} \; r(v_i, v_k) + a(v_i, v_k) \tag{12}$$

Subgraph refiner $P_\phi(G_S^*|G_S)$. Subgraph refiner consists of two submodules, a subgraph-level GIB and a subgraph-level mutual exclusion for complementing the lacking of prior knowledge on subgraph clustering. Concretely, the first term in Equation 6 is implemented by the cross-entropy loss for squashing the subgraph information towards labels, and the second term constrained by the KL-divergence and mutual exclusion is devised to preserve the minimal but sufficient information. Given a graph $G = \{G_S^1, ..., G_S^M\} \sim P_G$, we learn the sampling probability p_i at the subgraph level, where G_S^i

will be removed if $p_i = 1$. Inspired by [26], we impose $q_i \sim Bern(t)$ [1] to introduce the stochasticity and further define the variational distribution as

$$\begin{aligned} q(G_S) &= \sum_{G} P(G_S|G)P_G(G) \\ &= P(G_S^1, G_S^2, \cdots, G_S^M|G) \cdot P_G(G) \\ &= P(G_S^1|G) \cdot P(G_S^2|G) \cdots P(G_S^M|G) \cdot P_G(G) \\ &= P(q_1) \cdot P(q_2) \cdots P(q_M) \cdot P_G(G) \\ &= P_G(G) \prod_{i=1}^{M} P(q_i) \end{aligned} \tag{13}$$

Considering $P_G(G)$ is a constant that can be ignored in optimization, the KL-divergence for the variational approximation is formalized as

$$KL(P_\phi(G_S^*|G_S)||q(G_S)) = \sum_{G_S^i \in G_S} p_i \log \frac{p_i}{t} + (1-p_i) \log \frac{1-p_i}{1-t} \tag{14}$$

Therefore, Equation 14 and Equation 7 jointly constitute of the optimization objective for our g_ϕ.

Classifier $P_\psi(Y|G_S^*)$. The classifier receives the support subgraph set G_S^*, which includes a GNN block with an MLP layer. The GNN block separately encodes each subgraph into corresponding subgraph-level representation where each block shares the same parameters. We then exploit a sum-pooling strategy on the refined set of subgraphs by element-wise addition [39]. The MLP layer imposes linear transformations on compressed subgraph-level representation and finally outputs the categorical predictions for the downstream classification task.

Support property evaluation. To explore whether the discovered subgraph set G_S^* satisfies the expected support property, we devise an *Information Shift Method (ISM)* to reveal the performances of P2GNN when G_S^* is under diverse shifts. In practice, the encapsulated information in our model can be quantified as the number of subgraphs. Specifically, the core idea of *ISM* to impose interventions on G_S^* by decreasing or increasing subgraphs that input into classifier P_ψ, thus we can further verify the refined subgraph set G_S^* perfectly supports the prediction. This information quantification can lead to one issue, i.e., our learning system will be considered as experiencing the same information variation if only the number of subgraph shifts is the same. To this end, a pricinple guiding the ordering of information variation is required. As the learned p_i from P2GNN indicates the removal probability, then it is less confident with its removal decision when it becomes closer to 0.5, thus subgraphs with 0.5 removal probability should be first analyzed. With this insight, we can exploit p_i to rank the priority of each subgraph G_S^i and take 0.5 as the threshold to determine the direction of information shift. Therefore, we divide the G_S^* into two categories and rank them as,

$$S^+ =< G_1^+, G_2^+, ..., G_l^+ > \tag{15}$$

$$S^- =< G_1^-, G_2^-, ..., G_k^- > \tag{16}$$

where $0.5 \le p(G_1^+) \le p(G_2^+) \le, ..., \le p(G_m^+), p(G_k^-) \le, ..., \le p(G_2^-) \le p(G_1^-) < 0.5$. When extending information, we employ the order of S^+ for subgraph selection to progressively increase information for shift. Otherwise, the order of S^- will be utilized to decrease the information. To perturb the input information to the classifier P_ψ, *ISM* devises a simplified strategy, which modifies the removal probability p_i of these selected subgraphs into equal ones, i.e., 0.5. Specifically, we can formulate this operations by,

$$\text{ISM}(G_S^*, t) = \begin{cases} p(G_1^+) = p(G_2^+) = p(G_t^+) = 0.5, shift =' +' \\ p(G_1^-) = p(G_2^-) = p(G_t^-) = 0.5, shift =' -' \end{cases} \tag{17}$$

where t denotes the number of selected new subgraphs (information), $'+'$ indicates information expanding while $'-'$ is information decreasing. With our proposed *ISM*, we can easily select the subgraphs for information shifts and obtain the prediction performance trends to explore the support property of G_S^*. The detailed algorithm of P2GNN is provided in Algorithm 1.

Time complexity analysis. For each graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, P2GNN has two parts of calculation: subgraphs extraction and refinement. The execution time of the first stage main comes from calculating the walk distance $HitPath(v_i, v_j)$ and AP clustering. Their corresponding complexities are $O(\frac{|\mathcal{V}|}{2}|\mathcal{V}|)$ and $O(T|\mathcal{V}|^2)$, respectively, where $\frac{|\mathcal{V}|}{2}$ is truncated step, T is the number of iterations before convergence or max iterations. The reason for not including the hop distance calculation is that it has been calculated before training. In the subgraphs refinement stage, exploring the attention of each subgraph will take $O(M)$ time, where M is the number of subgraphs from subgraphs extraction. Therefore, the time complexity of P2GNN is $O((T + \frac{1}{2})|\mathcal{V}|^2 + M)$.

3.4 Theoretical Analysis

In our work, the asymmetric relationship between nodes is not directly measured by the ordering of status, but is measured with the relative status disparity. In this section, we will discuss two manners of status disparity measurement and justify the rationality of our solution with theoretical analysis.

Firstly, according to literature [24], we have two principles to describe the relationships among the walking distances across different nodes, i.e., Lemma 1 and Lemma 2.

Lemma 1. Given any three nodes v_i, v_j and v_k in graph G, the pairwise walking distances can be described as

$$\mathcal{H}(v_i, v_j) + \mathcal{H}(v_j, v_k) \ge \mathcal{H}(v_i, v_k) \tag{18}$$

Lemma 2. (Symmetric walk distances) Given two looped walk paths, the summations of walking distances on these two paths preserve invariant,

$$\mathcal{H}(v_i, v_j) + \mathcal{H}(v_j, v_k) + \mathcal{H}(v_k, v_i) = \mathcal{H}(v_i, v_k) + \mathcal{H}(v_k, v_j) + \mathcal{H}(v_j, v_i) \tag{19}$$

Proposition 1. Given $\mathcal{H}(v_i, v_j) > \mathcal{H}(v_j, v_i)$ that the statues v_i precedes v_j, this ordering is ambiguous in the whole graph, i.e., the ordering *Asymmetry* in graph is not transitive, while the relative difference between the walking distance $\mathcal{H}(v_k, v_i) - \mathcal{H}(v_i, v_k)$ can exactly capture the asymmetry of node-wise statuses, and quantify the status disparity between two nodes.

Proof. First, consider the three nodes v_i, v_j and v_k where v_i precedes v_j and v_j precedes v_k i.e. $\mathcal{H}(v_i, v_j) > \mathcal{H}(v_j, v_i)$ and $\mathcal{H}(v_j, v_k) > \mathcal{H}(v_k, v_j)$. We dissect the order of v_i and v_k through progressively deriving the following inequalities,

$$\mathcal{H}(v_i, v_j) + \mathcal{H}(v_j, v_k) > \mathcal{H}(v_j, v_i) + \mathcal{H}(v_k, v_j) \tag{20}$$

$$\mathcal{H}(v_i, v_j) + \mathcal{H}(v_j, v_k) > \mathcal{H}(v_k, v_i) \tag{21}$$

[1] Parameter t controls the sampling probability in Bernoulli distribution.

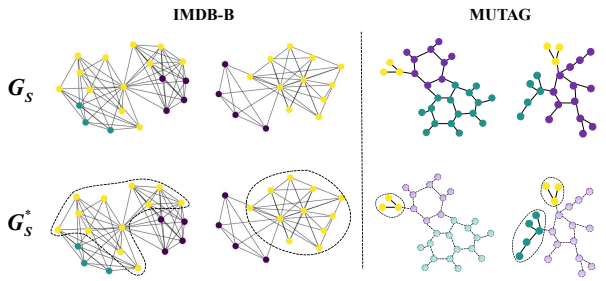


Figure 3: The universality and effectiveness of P2GNN on social networks and bioinformatics. Nodes with the same color belong to the same subgraph, and the captured support subgraphs are circled by the dashed line.

$$\mathcal{H}(v_i, v_k) + \mathcal{H}(v_k, v_j) + \mathcal{H}(v_j, v_i) > 2\mathcal{H}(v_k, v_i) \tag{22}$$

$$\mathcal{H}(v_i, v_k) - \mathcal{H}(v_k, v_i) > \mathcal{H}(v_k, v_i) - \mathcal{H}(v_k, v_j) - \mathcal{H}(v_j, v_i) \tag{23}$$

Since $\mathcal{H}(v_k, v_i) \leq \mathcal{H}(v_k, v_j) + \mathcal{H}(v_j, v_i)$ (Lemma 1), we have that $\mathcal{H}(v_i, v_k) > \mathcal{H}(v_k, v_i)$ is not always hold on, demonstrating the status relationship between v_i and v_k is ambiguous. Therefore, we can conclude that *Asymmetry* in graphs does not have the property of transitivity.

Second, considering v_k as a fixed intermediate node and assuming that v_i precedes v_j where these two nodes are both anchored on v_k, we can get the following derivations,

$$\mathcal{H}(v_i, v_k) - \mathcal{H}(v_k, v_i) \geq \mathcal{H}(v_j, v_k) - \mathcal{H}(v_k, v_j) \tag{24}$$

$$\mathcal{H}(v_i, v_k) + \mathcal{H}(v_k, v_j) \geq \mathcal{H}(v_j, v_k) + \mathcal{H}(v_k, v_i) \tag{25}$$

$$\mathcal{H}(v_i, v_j) \geq \mathcal{H}(v_j, v_i) \tag{26}$$

To this end, the relative difference can lead to a unique asymmetric ordering between v_i and v_j. Therefore, $\mathcal{H}(v_k, v_i) - \mathcal{H}(v_i, v_k)$ can definitely represent the *Asymmetry* in graphs.

Remark. Since the ordering of status relationships is not transmissible, only the relative walking distance can characterize the status disparity among nodes. In our implementation, we instantiate $||\mathcal{H}(v_i, v_j) - \mathcal{H}(v_j, v_i)||^2$ to measure the status disparity, where $\mathcal{H}$ is alternatively approximated by *HitPath* proposed in Sec 3.1.

4 EXPERIMENT

4.1 Experimental Settings

4.1.1 ***Datasets.*** The experimnents are conducted on eight public datasets regarding bioinformatics and social networks. More details of these datasets are summarized in Table 2 of Appendix A.

• **Bioinformatics datasets:** MUTAG [10], PROTEINS [6], and MOLHIV [38].

• **Social network datasets:** IMDB-B [40] and IMDB-M [40], REDDIT-B [40] and REDDIT-M [40], COLLAB [40].

4.1.2 ***Baselines.*** Our baselines are three-fold, including GNN backbones, subgraph learning methods, and interpretable models. • **Backbone baselines:** GCN [21], GraphSAGE [17], GIN [39] and PNA [9].

• **Subgraph methods:** GNN-AK [48], GIB [44] and SUGAR [32].

• **Interpretable models:** GSAT [26] and CAL [31].

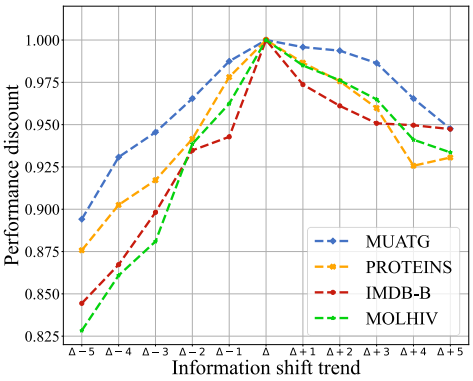


Figure 4: The performance discount with shifting the information of support subgraphs. Δ represents the information of G_S^*.

4.1.3 ***Our Setups.*** We provide some important training hyperparameters and metrics for different datasets.

• **Hyper-parameters.** We take GIN as the backbone of P2GNN. The balance coefficient γ is set to 0.8, and the walking step l is set to $\frac{|\mathcal{V}|}{2}$. The parameter sensitivity analysis is performed in Sec 4.4, with more detailed implementations presented in the Appendix B.

• **Metrics.** Following the common practice [39, 48], we report ROC-AUC on MOLHIV while present classification accuracy for all other datasets.

4.2 Result Analysis

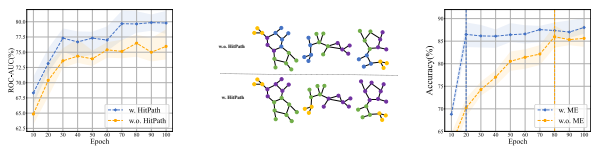
Table 1 shows the performance comparisons across different methods, and we can obtain the following three **Obs**ervations.

Obs 1: The methods based on subgraph discovery consistently outperform the traditional backbone models on all datasets. In backbone methods, GIN and PNA have achieved competitive performance, GIN (89.9%) on Reddit-B and PNA (79.1%) on MOLHIV could even obtain the expression ability of subgraph learning methods. In fact, both of them are often used as the backbone in subgraph learning methods. This observation demonstrates that subgraph learning can indeed improve the graph representation ability. Almost all subgraph methods greatly improve the prediction accuracy, however, we obtain the counter-examples on two complex social network datasets, such as Reddit-M and COLLAB. The maximal performance drop among subgraph learning baselines is 6.9% on Reddit-M, and PNA and GIB achieved comparable prediction accuracy on COLLAB. This phenomenon reveals the common shortcomings of subgraph learning methods on some social network datasets.

Obs 2: Compared with other subgraph learning models, our P2GNN achieves the most competitive results where we achieve the SOTA on four datasets. Specifically, our P2GNN outperforms best baselines by 6.3% and 1.3% respectively on IMDB-B and MUTAG, and encouragingly, P2GNN has significantly improved over other methods on most scenarios except three complex social network datasets. Such performance superiority can be explicitly attributed to the coupling effects of both two objectives, i.e., asymmetry-based subgraph extraction and robust refinement.

We believe that the two-stage subgraph learning can exactly better extract label-relevant subgraphs. Further, it is worth noting that our P2GNN does not have the best performance on the REDDIT-B, REDDIT-M and COLLAB. The underlying reasons are that 1) unlike the edges of friends, the interactions of users in REDDIT-B and REDDIT-M are more random thus it is less discriminative on QA community and discussion-based community, and due to the superior performance of CAL on them, we consider that the causal perspective may be able to further solve the common problem of subgraph discovery learning. 2) subgraph-level refinement may lead to information loss if the local pattern is not well captured. Therefore, given the large number of nodes in graphs, extraction and refinement on large-scale heterogeneous graphs with noisy or non-robust edges are still under explored.

Obs 3: The support subgraph set has the most sufficient and minimal information for prediction. We verify the support property of our discovered subgraphs using visualization method and *ISM* . Figure 3 visualizes the discovered subgraph set G_S and the support subgraph set G_S^* with our P2GNN. On IMDB-B, different patterns of the ego-network centering with various actors/actresses are bond to determine the genre of the movie [40]. Encouragingly, our support subgraphs captured by P2GNN can reveal prominent consistency with such ego-network. For one graph of MUTAG, P2GNN tends to take these two functional groups $-NO_2$ and $-NH_2$, or some substructures containing these functional groups, as support subgraphs. By looking into the chemical explanations in literature [25], we find that the ground-truth interpretation corresponds to our empirical results on MUTAG. Based on information theory, we also quantitatively analyze the support property of our discovered subgraphs via designing *ISM*. Centered on the support subgraph set G_S^*, we respectively explore the performance trends of set when it expands and decreases information. Figure 4 visualizes the results of *ISM* on three datasets. Reducing the information content of the support subgraph set G_S^* will obviously lead to a decrease in the prediction power. Also, we are more surprised to find that continuously expanding G_S^* doesn't improve the prediction ability and even hurt the accuracy of the prediction.



(a) Performance comparison on MOLHIV. (b) Comparison of the discovered subgraphs on MOLHIV. (c) Comparison of convergence speed on IMDB-B.

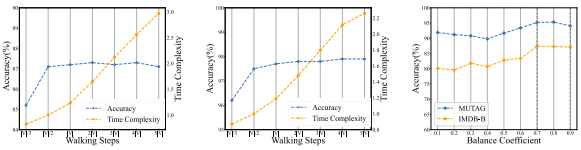
Figure 5: Ablation studies on *HitPath* and mutual exclusion in P2GNN.

4.3 Ablation Study

Ablation studies consist of two aspects. 1) We replace the *HitPath* with a symmetrical subgraph discovery measure with hop distance to verify the effectiveness of asymmetry measures. 2) We ablate the mutual exclusion (*ME*) principle in GIB to demonstrate whether our solution can exactly facilitate the learning convergence. Due to the limited space, we only illustrate studies on two datasets, more comprehensive evaluations can be found in Appendix D.

Figure 5(a) and 5(b) respectively illustrate the performances and visualized discovered subgraphs (with and without *HitPath*) on MOLHIV. Subgraph discovery based on asymmetry measures obviously reveals the better performances than hop-based measures, where solution w.o. *HitPath* only focuses on the topology and neglects the feature information. And most subgraphs discovered by solution w.o. *HitPath* are prone to be in a small size, probably due to the lacking connections of feature correlations. We can conclude that the asymmetric topology and feature correlations modeled by *HitPath* is superior to original random walk for subgraph discovery.

Figure 5(c) shows the comparison of convergence speeds between our P2GNN and variant without *ME*. Intuitively, we have a faster convergence speed with the mutual exclusion, verifying that the mutual exclusion principle can be viewed as a prior knowledge, to provide effective guidance for subgraph sampling and makes it easier to obtain label-relevant representation. It should not be ignored that our performance is also better than w.o. *ME*, which suggests that the mutual exclusion also plays an important role in the downstream tasks for better representation.



(a) Variations of walking steps l on IMDB-B. (b) Variations of walking steps l on MUTAG. (c) Variations of balance coefficient γ.

Figure 6: Parameter sensitivity analysis.

4.4 Parameter Sensitivity Analysis

The main hyper-parameters in P2GNN are two-fold. 1) The walking step l in *HitPath*, and 2) the topology-feature balance coefficient γ in Equation 5.

Figure 6 shows the performance and training time with different l on various datasets. With increasing l, especially when $l > \frac{|\mathcal{V}|}{2}$, the increased training time haven't led to better performance. Thus, to simultaneously balance efficiency and accuracy, we set $l = \frac{|\mathcal{V}|}{2}$. We analyze that the walking distance l should be neither set too small nor a fixed value per graph. When the value of walking distance l is too small, the randomness of walking makes the process may be limited to a few nodes, resulting in local information cannot be captured. Fixed l is often unreasonable for graphs of different sizes. Therefore, although the purpose of $HitPath$ is to encode local information, we still walking more than half of the number of nodes $\frac{|\mathcal{V}|}{2}$.

Figure 6(c) shows the fluctuation of performances under different γ. Obviously, we get stable results at $\gamma \in [0.7, 0.9]$. Then we set γ as 0.8 in our experiments. More hyper-parameter settings are carefully described in Table 3 in Appendix B.

Table 1: Performance comparisons. The best result is in bold across all methods and the second best is underlined.

	MUTAG	PROTEINS	IMDB-B	IMDB-M	Reddit-B	Reddit-M	COLLAB	MOLHIV
GCN	74.3±11.0	74.2±3.1	70.0±0.9	51.5±3.2	85.5±2.1	48.6±2.3	69.6±2.1	75.5±1.6
Graph-SAGE	74.3±7.7	73.0±4.5	70.9±4.1	47.6±3.5	84.3±1.9	50.0±1.3	71.6±1.5	74.8±3.4
GIN	89.4±5.6	77.0±4.3	75.6±3.7	48.5±3.3	89.9±1.9	56.1±1.7	73.9±1.7	75.6±1.4
PNA	89.6±5.3	76.7±4.2	79.8±4.5	48.0±2.0	81.7±6.1	54.2±1.2	74.2±2.1	79.1±1.3
GNN-AK	92.3±6.8	77.1±5.1	75.2±3.1	53.2±1.2	**94.6±1.0**	54.8±2.1	78.5±1.8	79.2±1.1
GIB	83.9±6.4	77.2±3.4	73.7±7.0	51.4±2.1	90.3±1.9	49.2±2.0	74.5±2.0	76.4±2.7
SURGAR	92.4±2.1	81.0±2.4	80.1±2.8	51.1±1.9	89.4±2.1	50.1±2.4	77.4±1.7	77.0±3.1
GSAT	94.1±2.1	76.2±1.4	72.6±4.4	54.5±3.2	85.4±2.0	56.2±1.7	81.8±1.4	78.1±2.0
CAL	89.9±8.3	76.9±3.3	74.1±5.2	52.6±2.4	91.2±2.3	**57.1±1.9**	**82.7±1.3**	78.1±2.0
P2GNN	**95.4±2.9**	**82.1±3.1**	**87.4±3.9**	**56.2±2.1**	91.4±2.4	56.8±2.0	81.6±1.9	**79.8±1.4**

5 RELATED WORK

Graph representation model. There is a growing interest in exploring more expressive graph learning model. Most previous works designed more powerful node-level learning methods, such as GCN [21], GraphSAGE [17], GAT [34] and GIN [39], which rely on stronger Aggregate, Combine and Readout functions. Along another line of research, subgraph-based representation strategy has been proposed to provide more intuitive understanding and better interpretability. These methods expect to obtain subgraphs or network motifs which are simple building blocks of complex networks and determine the functionalities of graphs. More details, we categorize these strategies into two classes: extracting substructure patterns to capture certain topological structure of subgraphs, and removing non-useful information to capture subgraphs that affect the model predictions the most. The former focuses on extracting the structure patterns of subgraph from the perspective of topology, and the latter aims to refine the most valuable subgraphs for learning tasks from the perspective of information.

Substructure extraction. Most previous works on substructure extraction tend to determine substructure patterns based on domain knowledge or directly adopt ego-net structure. [14, 27] propose density-based subgraph discovery methods for some specific areas, including network science, biological analysis, and graph databases. GNN-AK [48] directly applies star-pattern subgraph and convolves all subgraphs with a base GNN as kernel, which produce multiple rich subgraph-node embeddings. SubGNN [2] decouples the graph topology to three property-aware channels, and designs three structure patterns subgraph which capture position, neighborhood, and structure. XGNN [45] trains a graph generator to interpret GNNs by edge-wise sampling, which provides an understanding of which parts contributing to final predictions. However, these methods reveal great limitations. First, scenario-based subgraph discovery models greatly lack generality. Then, the substructure discovery methods of ego-net still follow MPNNs' local neighbor aggregation, which don't take advantage of subgraph representation learning.

Information refinement. Recent works borrow the information theory to develop the Graph Information Bottleneck (GIB). These GIB-based solutions squash the original graph into one minimal sufficient subgraph by removing the redundant nodes [43, 44]. Unfortunately, given the personalized structures of subgraphs across domains, and the entangled correlations within subgraphs, existing subgraph learning scheme reveals two limitations. First, there still lack a general principle to accommodate various subgraph structures across multiple domains. Second, refinement performing on node levels without explicit guidance for eliminating redundant information tend to distort the local topology and lead to an inefficiency convergence process.

6 CONCLUSION

In this paper, we present a two-stage subgraph learning architecture P2GNN, which performs general subgraph extract and efficient refinement. In the extraction stage, *Local Asymmetry* is proposed to accommodate diverse domain-specific subgraphs discovery tasks. We design a novel node-wise asymmetry measurement *HitPath*, and achieve subgraph extraction via *AP* clustering. Theoretical analysis of *Hitpath* is also provided to verify its asymmetry measuring capacity. In the refinement stage, we propose the principle of mutual exclusion regularization to explicitly guide eliminating redundant information and thus boost the efficiency of refinement. Further, we empirically verify the superiority on universality and effectiveness via experiments and propose *ISM* to prove the support property of discovered subgraph set.

Limitations: In the subgraphs extraction stage, personalizing the number of subgraphs according to domain knowledge may result in better predictive power, which remains unexplored. And in the subgraphs refinement stage, the analysis of our refinement method from the perspective of causality can be further studied.

ACKNOWLEDGMENTS

This paper is partially supported by the National Natural Science Foundation of China (No.62072427, No.12227901), the Project of Stable Support for Youth Team in Basic Research Field, CAS (No.YSBR-005), Academic Leaders Cultivation Program, USTC.

REFERENCES

[1] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. 2016. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410* (2016).

[2] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8017–8029.

[3] Yunsheng Bai, Derek Xu, Yizhou Sun, and Wei Wang. 2021. Glsearch: Maximum common subgraph detection via learning to search. In *International Conference on Machine Learning*. PMLR, 588–598.

[4] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. 2021. Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems* 34 (2021), 25280–25293.

[5] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. 2022. Equivariant Subgraph Aggregation Networks. In *International Conference on Learning Representations*.

[6] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[8] Fan Chung and S-T Yau. 2000. Discrete Green's functions. *Journal of Combinatorial Theory, Series A* 91, 1-2 (2000), 191–214.

[9] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems* 33 (2020), 13260–13271.

[10] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.

[11] Huiqi Deng, Qihan Ren, Hao Zhang, and Quanshi Zhang. 2021. DISCOVERING AND EXPLAINING THE REPRESENTATION BOTTLENECK OF DNNS. In *International Conference on Learning Representations*.

[12] Lun Du, Xiaozhou Shi, Qiang Fu, Xiaojun Ma, Hengyu Liu, Shi Han, and Dongmei Zhang. 2022. GBK-GNN: Gated Bi-Kernel Graph Neural Networks for Modeling Both Homophily and Heterophily. In *Proceedings of the ACM Web Conference 2022*. 1550–1558.

[13] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* (2020).

[14] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks VS Lakshmanan, and Xuemin Lin. 2019. Efficient algorithms for densest subgraph discovery. *arXiv preprint arXiv:1906.00341* (2019).

[15] Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. 2022. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems* 35 (2022), 31376–31390.

[16] Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *science* 315, 5814 (2007), 972–976.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[18] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard TB Ma, Hongzhi Chen, and Ming-Chang Yang. 2022. Measuring and improving the use of graph information in graph neural networks. *arXiv preprint arXiv:2206.13170* (2022).

[19] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2020. Hierarchical generation of molecular graphs using structural motifs. In *International conference on machine learning*. PMLR, 4839–4848.

[20] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2020. Multi-objective molecule generation using interpretable substructures. In *International conference on machine learning*. PMLR, 4849–4859.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems* 34 (2021), 21618–21629.

[23] A Lasota and James A Yorke. 1982. Exact dynamical systems and the Frobenius-Perron operator. *Transactions of the american mathematical society* 273, 1 (1982), 375–384.

[24] László Lovász. 1993. Random walks on graphs. *Combinatorics, Paul erdos is eighty* 2, 1-46 (1993), 4.

[25] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. *Advances in neural information processing systems* 33 (2020), 19620–19631.

[26] Siqi Miao, Mia Liu, and Pan Li. 2022. Interpretable and Generalizable Graph Learning via Stochastic Attention Mechanism. In *International Conference on Machine Learning*. PMLR, 15524–15543.

[27] Dung Nguyen and Anil Vullikanti. 2021. Differentially private densest subgraph detection. In *International Conference on Machine Learning*. PMLR, 8140–8151.

[28] Giannis Nikolentzos and Michalis Vazirgiannis. 2020. Random walk graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 16211–16222.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[30] Christoph Schweimer, Christine Gfrerer, Florian Lugstein, David Pape, Jan A Velimsky, Robert Elsässer, and Bernhard C Geiger. 2022. Generating Simple Directed Social Network Graphs for Information Spreading. In *Proceedings of the ACM Web Conference 2022*. 1475–1485.

[31] Yongduo Sui, Xiang Wang, Jiancan Wu, Min Lin, Xiangnan He, and Tat-Seng Chua. 2022. Causal attention for interpretable and generalizable graph classification. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1696–1705.

[32] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. 2021. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proceedings of the Web Conference 2021*. 2081–2091.

[33] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *Sixth international conference on data mining (ICDM'06)*. IEEE, 613–622.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[35] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. 2022. Equivariant and stable positional encoding for more powerful graph neural networks. *arXiv preprint arXiv:2203.00199* (2022).

[36] Pengkun Wang, Chuancai Ge, Zhengyang Zhou, Xu Wang, Yuantao Li, and Yang Wang. 2021. Joint Gated Co-attention Based Multi-modal Networks for Subregion House Price Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[37] Tailin Wu, Hongyu Ren, Pan Li, and Jure Leskovec. 2020. Graph information bottleneck. *Advances in Neural Information Processing Systems* 33 (2020), 20437–20448.

[38] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical science* 9, 2 (2018), 513–530.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[40] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1365–1374.

[41] Carl Yang, Mengxiong Liu, Vincent W Zheng, and Jiawei Han. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 47–52.

[42] Nianzu Yang, Kaipeng Zeng, Qitian Wu, Xiaosong Jia, and Junchi Yan. 2022. Learning substructure invariance for out-of-distribution molecular representations. In *Advances in Neural Information Processing Systems*.

[43] Junchi Yu, Jie Cao, and Ran He. 2022. Improving subgraph recognition with variational graph information bottleneck. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19396–19405.

[44] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. 2020. Graph information bottleneck for subgraph recognition. *arXiv preprint arXiv:2010.05563* (2020).

[45] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 430–438.

[46] Yanfu Zhang, Hongchang Gao, Jian Pei, and Heng Huang. 2022. Robust Self-Supervised Structural Graph Neural Network for Social Network Prediction. In *Proceedings of the ACM Web Conference 2022*. 1352–1361.

[47] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2022. Protgnn: Towards self-explaining graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9127–9135.

[48] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. 2021. From stars to subgraphs: Uplifting any GNN with local structure awareness. *arXiv preprint arXiv:2110.03753* (2021).

[49] Zhengyang Zhou, Yang Wang, Xike Xie, Lianliang Chen, and Hengchang Liu. 2020. RiskOracle: a minute-level citywide traffic accident forecasting framework. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 1258–1265.

[50] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2021. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11168–11176.

A DETAILS OF DATASETS

We provide the details and statistics of datasets in this section, where the statistics of these eight datasets are illustrated in Table 2.

• **MUTAG** [10] is a binary dataset of molecular property, where nodes are atoms and edges are chemical bonds. Each graph is associated with a binary label based on its mutagenic effect.

• **PROTEINS** [6] is a dataset of proteins that are classified as enzymes or non-enzymes. Nodes represent the amino acids and two nodes are connected if they are less than 6 Angstroms apart.

• **IMDB-B** and **IMDB-M** [40] are two movie collaboration datasets, where nodes represent actors/actress. There is an edge between nodes if they appear in the same movie.

• **REDDIT-B** and **REDDIT-M** [40] are two datasets of social networks, where nodes represent users. There is an edge between nodes if the comment interaction has appeared between them.

• **COLLAB** [40] is a scientific collaboration dataset, derived from 3 public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics.

• **MOLHIV** [38] is a molecular property dataset, where nodes are atoms and edges are chemical bonds. Each molecule has a binary label, which depends on whether the molecule can inhibit HIV virus replication or not.

Algorithm 1 P2GNN

Input: Graph G, the number of learning epochs N, the maximum iterations of AP clustering T.
Output: The prediction $\hat{Y}$ of graph G.

Initialization: θ, ϕ, ψ, GNN encoder $h(\cdot)$.
for n = 0 to N **do**
  $w_{v_i v_j} \leftarrow ||h(v_i) - h(v_j)||^2$
  for t = 0 to T **do**
    /* $P_\theta(G_S|G)$ */
    AP clustering to disentangle G and obtain G_S:
    if convergence **then**
      break
    end if
  end for
  /* $P_\phi(G_S^*|G_S)$ */
  Sample a subgraph set G_S^* from G_S .
  /* $P_\psi(Y|G_S^*)$ */
  Calculate $\mathcal{L}$ with Equation 9.
  $\theta, \phi, \psi \leftarrow \text{Adam}(\theta, \phi, \psi)$.
end for
Predict by support subgraph set $\hat{Y} = P_\psi(Y|G_S^*)$.
return $\hat{Y}$

B DETAILS OF THE CONFIGURATION

To ensure fair comparisons on all datasets, we keep the same configurations on all of them and present Table 3 to summarize the detailed configurations.

C ANALYSIS OF SPECTRAL THEOREM

In this paper, we employ *Spectral Theorem* to verify the observation of Local Asymmetry, and our asymmetric metric criterion (*HitPath*) using random walk is also inspired by spectral decomposition. Therefore, it is necessary to construct a comprehensive understanding of *Spectral Theorem*, and we provide a detailed analysis in this section.

Table 2: Statistics of datasets.

Dataset	Graphs	Classes	Avg. Nodes	Avg. Edges
MUTAG	188	2	17.93	19.79
PROTEINS	1,113	2	39.06	72.82
IMDB-B	1,000	2	19.77	96.53
IMDB-M	1,500	3	13.00	65.94
REDDIT-B	2,000	2	429.63	497.75
REDDIT-M	4,999	5	508.52	594.87
COLLAB	5,000	3	74.49	2457.78
MOLHIV	41,127	2	25.50	27.50

The idea of *Spectral Theorem* is to study graph structure via its Laplacian operator of graph. According to different interpretation strategies of eigenvectors, many works have achieved great success.

C.1 Position encoding with Spectral Theorem

In this subsection, we provide an analysis of the application of *Spectral Theorem* to the field of Position encoding (PE), and demonstrate the observation of Local Asymmetry from our understanding perspective as shown in Figure 1(ii).

Given a graph G, $d(v_i)$ denotes the degree of the node v_i in G, and let $\boldsymbol{W}$ denote adjacency matrix. We perform the following analysis.

In electromagnetic theory, the Green's function of the Laplacian [8] shows the electrostatic potential of a given charge. This understanding inspired the PE of nodes on the graph. Consider the Laplacian G and can be computed by its eigenfunctions,

$$G(j_1, j_2) = d_{j_1}^{-\frac{1}{2}} d_{j_2}^{\frac{1}{2}} \sum_{i>0} \frac{(\alpha_{i,j_1}\alpha_{i,j_2})^2}{\hat{\lambda}_i} \tag{27}$$

Further, researchers [7] use the interaction between two heat kernels to define in Equation 27 the diffusion distance d_D between nodes j_1, j_2,

$$d_D^2(j_1, j_2) = \sum_{k>0} e^{-2t\lambda_i}(\alpha_{i,j_1} - \alpha_{i,j_2})^2 \tag{28}$$

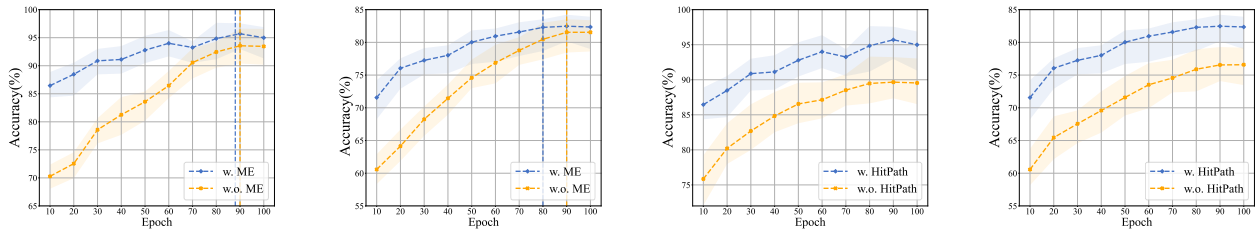
Inspired by it, the biharmonic distance d_B was proposed as a better measure of distances [22],

$$d_B^2(j_1, j_2) = \sum_{i>0} \frac{(\alpha_{i,j_1} - \alpha_{i,j_2})^2}{\lambda_i^2} \tag{29}$$

Equation29 shows that smaller frequencies/eigenvalues are more heavily weighted when determining distances between nodes. Thus, we study the 5-th lowest eigenvectors and draw heat maps Figure 1(ii). We are excited to find that there is a perfect match for our proposed Local Asymmetry. Therefore, we successfully prove Local Asymmetry via exploiting *Spectral Theorem*.

C.2 Random walk with Spectral Theorem

In this subsection, we analyze random walks from the perspective of spectral theory, and subsequently supplement the preliminary knowledge of **Lemma 1** and **Lemma 2**.



(a) Performance comparison on MUTAG. (b) Performance comparison on PROTEINS. (c) Performance comparison on MUTAG. (d) Performance comparison on PROTEINS.

Figure 7: Ablation study on *ME* and *HitPath*.

Table 3: Configurations in P2GNN.

Config	Description	Value
Backbone	The backbone of P2GNN	GIN
Smoothing coefficient α_r	α_r in Algorithm 1	0.5
Smoothing coefficient α_a	α_a in Algorithm 1	0.5
Balance coefficient β	β in Equation 9	1
Balance coefficient γ	γ in Equation 5	0.8
Walking steps l	Walking steps of *HitPath*	$\frac{\lvert\mathcal{V}\rvert}{2}$
K	Restart times of *HitPath*	100
σ	Distance between inaccessible nodes	$4\lvert\mathcal{V}\rvert$
Batchsize	Number of graphs in a batch	128
Split	Train set/Validation set/Test set	8/1/1
Hidden dimension	Hidden dimension of backbone	64
Base learning rate	Initial learning rate	1e-3
Dropout	Dropout rate of MLP	0.3

Actually, the cornerstone of our theoretical analysis is *Spectral Theorem*, which draws the topological characteristics of graphs from spectral perspective [22]. Here, we will briefly describe the connection between *Spectral Theorem* and random walks.

In the traditional strategy, the walking distance is calculated by the transition matrix $\boldsymbol{M} = \boldsymbol{DW}$, where M_{ij} represents the probability that node v_i jumps to v_j in one step. For a high-order formation, we let $\boldsymbol{M}_{ij}^t$ stand for the transition probability from v_i to v_j at the step t. Thus, the expected walking distance $\mathcal{H}$ can be calculated by,

$$\mathcal{H} = \sum_{t=1}^{\infty} t \cdot M^t \tag{30}$$

In contrast, exploiting *Spectral Theorem* to calculate the walking distance has more elegant properties where we can arrive a closed form of our distance. Consider the matrix $\boldsymbol{N} = \boldsymbol{D}^{1/2}\boldsymbol{W}\boldsymbol{D}^{1/2} = \boldsymbol{D}^{-1/2}\boldsymbol{M}\boldsymbol{D}^{1/2}$. Due to the symmetry of $\boldsymbol{N}$, it can be written in the spectral form,

$$\boldsymbol{N} = \sum_{k=1}^{n} \lambda_k \alpha_k \alpha_k^T \tag{31}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ are the eigenvalues of $\boldsymbol{N}$ and $\alpha_1, \cdots, \alpha_n$ are corresponding eigenvectors of unit length. By Frobenius-Perron Theorem [23], we have $\lambda_1 = 1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq -1$, and

$$\boldsymbol{M}^t = \boldsymbol{D}^{1/2}\boldsymbol{N}^t\boldsymbol{D}^{-1/2} = \sum_{k=1}^{n} \lambda_k^t \boldsymbol{D}^{1/2}\alpha_k\alpha_k^T\boldsymbol{D}^{-1/2} \tag{32}$$

Further, we can obtain walking distance from v_s to v_t,

$$\mathcal{H}(v_s, v_t) = 2m \sum_{k=2}^{n} \frac{1}{1-\lambda_k}\left(\frac{\alpha_{kt}^2}{d(t)} - \frac{\alpha_{ks}\alpha_{kt}}{\sqrt{d(s)d(t)}}\right) \tag{33}$$

Therefore, Equation 33 can exactly provide a closed-form solution to obtain the walking distance, which is superior to traditional strategy. By this exact solution, the principles in **Lemma 1** and **Lemma 2** can be easily derived.

D DETAILED ABLATION STUDY

We provide the detailed ablation studies on all datasets, which are shown in Figure 7. For ablations on *HitPath*, it is worth noting that the performance of solution with *HitPath* is still going to increase even achieving 100 epochs, manifesting that *HitPath* can better capture the correlation between nodes with ever-increasing epochs. For ablations on *ME*, we observe that the convergence speeds of these two solutions are similar on MUTAG and MOLHIV. We speculate that the subgraphs G_S discovered by P2GNN in the bioinformatic datasets are naturally mutually exclusive, so the regularization of *ME* does not significantly improve the convergence speeds. Even so, the performance of our method is still better than that w.o. *ME*.

E BROADER IMPACTS

As shown in Figure 4, our P2GNN automatically captures label-relevant functional groups on the chemical datasets. However, we also mentioned that P2GNN's performance on the social network datasets drops slightly.

We consider the main reason of such suboptimal performance on social networks is the edge diversity property, which is crucial for subgraph extraction. The node-wise relations in social networks can be classified into friendship, collaboration, and common interests, etc. Even, the edges of certain collaboration also tend to contain diverse information. Unfortunately, such unavailable edges features in our datasets lead to much difficulty in subgraph extraction. But in contrast, the edges in the molecular graph described by interatomic force are homogeneous. Then such edge type homogeneity can contribute to successful subgraphs extraction. Therefore, we will explore the homogenous relationship between the nodes of the graph to facilitate the in-depth study of asymmetry.