LINC: A Motif Counting Algorithm for Uncertain Graphs

Chenhao Ma[1], Reynold Cheng[1], Laks V.S. Lakshmanan[2],
Tobias Grubenmann[1], Yixiang Fang[3]*, Xiaodong Li[1]

[1]The University of Hong Kong, [2]The University of British Columbia, [3]University of New South Wales
[1]{chma2,ckcheng,tobias,xdli}@cs.hku.hk, [2]laks@cs.ubc.ca, [3]yixiang.fang@unsw.edu.au

ABSTRACT

In graph applications (e.g., biological and social networks), various analytics tasks (e.g., clustering and community search) are carried out to extract insight from large and complex graphs. Central to these tasks is the counting of the number of *motifs*, which are graphs with a few nodes. Recently, researchers have developed several fast motif counting algorithms. Most of these solutions assume that graphs are deterministic, i.e., the graph edges are certain to exist. However, due to measurement and statistical prediction errors, this assumption may not hold, and hence the analysis quality can be affected. To address this issue, we examine how to count motifs on uncertain graphs, whose edges only exist probabilistically. Particularly, we propose a solution framework that can be used by existing deterministic motif counting algorithms. We further propose an approximation algorithm. Extensive experiments on real datasets show that our algorithms are more effective and efficient than existing solutions.

PVLDB Reference Format:
Chenhao Ma, Reynold Cheng, Laks V.S. Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. LINC: A Motif Counting Algorithm for Uncertain Graphs. *PVLDB*, 13(2): 155-168, 2019.
DOI: https://doi.org/10.14778/3364324.3364330

Keywords

1. INTRODUCTION

Due to the prevalence of graphs in various important domains (e.g., life and social sciences), graph analysis has recently received a lot of attention. Various statistical tasks, such as clustering, classification, and prediction, enable discovery of important insights from the graphs involved. In biological networks, for example, it is interesting to perform network comparison [48] and discover protein functions [44]. In social networks, common analytic tasks include studying transitivity properties [17, 18], community detection [29], and understanding of how communications unfold [24].

The analysis tasks above involve the counting of *graphlets* [11] or *motifs* [39][1], which are graphs with a few nodes and edges. A motif, generally considered as a basic building block of a large graph [26], has been recently used to facilitate graph clustering [51], community search [29], and network alignment [38]. In Fig. 1, given the graph A, the *triangle* (a 3-node motif) occurs twice. This means that two subgraphs of G ($a-b-c$ and $c-d-e$) are isomorphic to *triangle*. As another example, *2-star*, another 3-node motif, has a count of 4. In other words, we can find a maximum of 4 subgraphs from A that are isomorphic to *2-star*. [2] Table 1 lists motifs commonly studied in the literature.

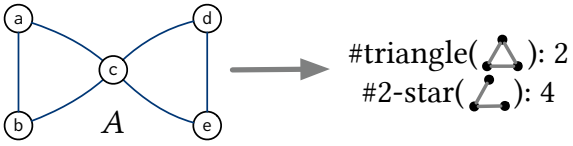


Figure 1: Motif counting

Table 1: 3-node, 4-node and 5-node motifs

	Motif	Description	Symbol	Related Work
3-node		*2-star*	$M_{3,1}$	[27, 31, 5, 37] [42, 11, 13, 48] [47]
		triangle	$M_{3,2}$	
4-node		*4-path*	$M_{4,1}$	[5, 37, 42, 11] [13, 48, 47]
		3-star	$M_{4,2}$	
		4-tailedtriangle	$M_{4,3}$	
		4-cycle	$M_{4,4}$	
		4-chordalcycle	$M_{4,5}$	
		4-clique	$M_{4,6}$	
5-node		*5-clique*	$M_{5,1}$	[42, 11, 13, 47]

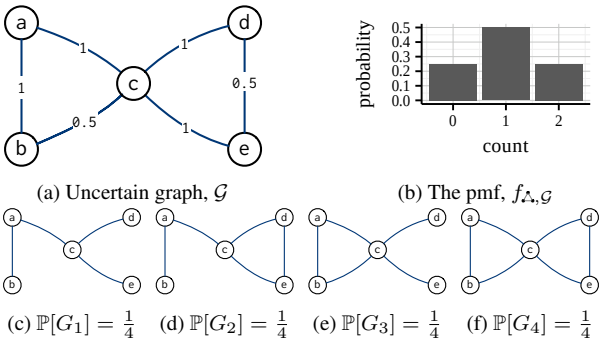
As pointed out in [19], motif counting reveals various interesting properties of a graph G. For example:
• **2-star**: if G has a larger *2-star* count, the degree distribution of G has a higher dispersion.
• **3-star**: if G has a larger *3-star* count, the degree distribution of G is more skewed.
• **triangle**: a larger *triangle* count for G indicates that G is closer to being transitive.

Motif counts are also used to find the *global clustering coefficient*, which measures the degree to which nodes tend to cluster

*Yixiang Fang is the corresponding author.
[1]In this paper, we treat "motif" and "graphlet" interchangeably.
[2]Here we assume induced subgraph semantics. In this paper, we will explain how to count motifs using the induced and non-induced subgraph semantics.

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. 13, No. 2
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3364324.3364330

together [52, 50]. This metric can be used to discover the topological and functional properties of protein-protein interaction (PPI) networks. In Fig. 1, for example, the global coefficient of graph A, which a function of count values for *triangle* and *2-star* motifs, is equal to 0.6.



(a) Uncertain graph, $\mathcal{G}$ (b) The pmf, $f_{\triangle,\mathcal{G}}$

(c) $\mathbb{P}[G_1] = \frac{1}{4}$ (d) $\mathbb{P}[G_2] = \frac{1}{4}$ (e) $\mathbb{P}[G_3] = \frac{1}{4}$ (f) $\mathbb{P}[G_4] = \frac{1}{4}$

Figure 2: The pmf of *triangle* counts

Uncertain graphs. Due to the importance of motif counting, researchers have developed algorithms to solve this problem efficiently [42, 5, 13]. Most of these approaches assume *deterministic* graphs, where the edges of the graph are certain to exist. However, this assumption may not hold in some applications. In a PPI network, where each node represents a protein compound, the existence of an edge indicates an interaction between two proteins. Whether an edge exists is not certain, because the evidence of protein interaction, based on experimental results and statistical models, can be erroneous [6]. In a social network, each user is denoted by a node, and an edge between two users indicates their relationship. These edges, which are often the result of a prediction, may not exist [10]. Uncertainty is also inherent in wireless, sensor, and road networks [3, 21]. Fig. 2a illustrates a typical *uncertain graph*, where each edge is associated with an *existential probability* to depict the chance that the edge exists.

Existing motif counting algorithms, which are not designed to handle uncertain graphs, can yield inaccurate results. Let us consider a *triangle* ($\triangle$) and the uncertain graph $\mathcal{G}$ in Fig. 2a. If the probabilistic information in the edge is not considered, we see that *two* subgraphs in $\mathcal{G}$ are isomorphic to *triangle* (i.e., the count of *triangle* is 2). Fig. 2b shows the probability mass function (pmf) of the frequency of this motif, which can be obtained by enumerating all the possible graph instances (or *possible worlds* [3, 47]). Figures 2c-f show four possible worlds and their respective probabilities. From the pmf shown, the average count of *triangle* is 1, and the probability that $\mathcal{G}$ has two *triangle* instances is only 0.25.

Indeed, as found in [47] and our results, the expected motif count (by considering the uncertain graph) can be very different from the motif count on the graph's deterministic version (in which all its edges exist definitely). The uncertain graph model enables the derivation of statistical information about motif counts, thereby benefiting graph analysis. In [47], the authors obtained the mean and variance values about the motif counts and employed these statistics to cluster biological networks that are uncertain. They showed that these statistics could improve clustering accuracy. The same paper also discussed the use of the pmf information (e.g., Fig. 2b) to analyze and compare biological networks. The pmf of the *triangle* count can also be used to indicate how likely the transitivity of the graph is – high or low.

Challenges. We aim to investigate motif counting for uncertain graphs extensively. This problem has not been well studied. In [47], the authors proposed a solution to calculate the variance of the motif count. On a graph with $3M$ edges, their algorithm takes about 2 hours to count *triangle* motifs and yields a relative error of

Figure 3: Our proposed solutions: PGS and LINC

140% in variance in our experiments. There is thus room for improvement in both efficiency and effectiveness of motif counting. However, it is not easy to develop an algorithm that satisfies these requirements. Recall from Fig. 2 that an uncertain graph $\mathcal{G}$ can be considered as a representation of a set of possible worlds. We can aggregate the counts of the motifs in the possible worlds and obtain accurate statistics (e.g., pmf and variance). Unfortunately, this simple algorithm is prohibitive owing to its exponential running time [3].

Our solutions: PGS and LINC. To tackle the challenging problem of counting motifs on uncertain graphs, we propose two solutions based on a sampling framework, as shown in Fig. 3. Given an uncertain graph $\mathcal{G}$ and a motif M, the solutions produce statistical information (pmf, mean, and variance) about the number of subgraphs of G isomorphic to M. Our solutions support a wide range of motifs, unlike existing counting algorithms that are tailored for particular motifs [48, 5]. Our first algorithm, called *possible graph sampling* (PGS), draws a number of possible world samples from $\mathcal{G}$ and runs a deterministic motif-counting algorithm on each possible world. The solution is simple to implement, adaptive to state-of-the-art motif counting solutions, and provides statistical guarantees in the result. Particularly, PGS has a theoretical accuracy bound on the pmf of the motif count – the error is not larger than ε, with a probability of $1 - \delta$.

However, PGS is not very efficient, because the deterministic counting solution has to be executed many times. To further speed up the motif-counting process, we propose a second solution, called *linking and counting* (or LINC). This solution is based on the intuition that the possible worlds sampled from $\mathcal{G}$ are often structurally similar. Conceptually, LINC "borrows" the motif count value obtained from one possible world, and deduces the result for another one. As we will explain later, this is based on our encoding method, which embeds M and its appearances in $\mathcal{G}$ into binary representations, allowing fast bit operations on the graph structures involved. To attain further speedup, we have developed two optimization techniques: (1) an "early-stop strategy", which allows motif-counting to terminate faster when only the mean or the variance values of motif counts are required; and (2) customization for common motifs (e.g., 2-star in Table 1).

We have performed time and space complexity analysis for our solutions. We have also conducted experiments on various motif types, including 6-node motifs. While the accuracy of the state-of-the-art counting algorithm for uncertain graphs, called BMA [47], fluctuates considerably over different datasets and motifs, LINC and PGS perform consistently well. Moreover, LINC is up to three order-of-magnitude faster than both BMA and PGS.

The rest of the paper is organized as follows. We review related work in Section 2. In Section 3, we give a formal problem definition. We discuss PGS in Section 4. We give an overview of LINC in Section 5, its design details in Section 6, and the optimizations and extensions of LINC in Section 7. We present our experimental results in Section 8, and conclude in Section 9.

2. RELATED WORK

A motif, essentially a connected graph of a few nodes and edges, is often considered to be a fundamental building block of a large and complex network [39, 11, 26]. *Motif-based-analysis*, which interprets a graph based on motifs, has attracted a lot of attention

in the past few years. For example, the problem of motif-aware graph clustering has been studied in [9, 34, 51]. They utilized motif conductance, a generalization of the conductance metric for motifs, in the graph clustering process. In [24], the issues of mining communication motifs from dynamic interaction networks were investigated. The authors developed a technique called COMMIT, which converts a dynamic network into a sequence database, based on which communication motifs are discovered. Motifs have also been recently used to facilitate community search [29] and network alignment [38]. In the following, we focus on the works related to motif counting on deterministic and uncertain graphs.

Motif counting on deterministic graphs. Motif counting, which facilitates graph analysis tasks (e.g., clustering analysis [52, 50, 51], community search [29], network alignment [38], and transitivity studies [19]), have been actively studied in the last few years. To handle the increasing needs of counting motifs in graphs, researchers have designed efficient counting algorithms for motifs of three to five nodes [27, 31, 5, 37, 42]. Some of these motifs are illustrated in Table 1. For 3-node motifs, in [27], the authors proposed a fast solution to enumerate and count *triangle* instances. The authors in [31] focused on maintaining triangle counts under updates of edges. For 4-node motifs, [37] designed an enumeration algorithm to count motif-instances. The authors in [5] further developed mathematical formulas to avoid enumerating 4-node motifs that have a huge number of motif-instances. For 5-node motifs, [42] studies formulas to speed up the counting process. To enable motif counting on large graphs, approximation algorithms have been recently developed, including a color coding-based sampling method [11] and a random-walk-based solution [13].

It is worth notice that all the solutions above assume a deterministic graph, i.e., every edge in the graph exists with certainty. As we discussed before, this may lead to a poor estimation of motif counts. Using an uncertain graph in the counting process can improve accuracy significantly, as shown in our experiments.

Motif counting on uncertain graphs. This topic has not been well studied. In [48], the authors proposed an approximation method to estimate the mean of motif counts on a given uncertain graph. This algorithm only considered a limited case of the uncertain graph model, where all the graph edges are assumed to have the same existential probability. This is not true in the real-world datasets that we considered in our experiments. In more recent work, [47] proposed a method, named BM, to calculate the expectation and variance of motif counts. As found in our experiments, the solution cannot scale well beyond graphs with thousands of edges. It also cannot give more detailed statistical information (e.g., the pmf of a motif count). Motif counting over an uncertain graph can be cast as answering a query over a tuple-independent probabilistic database (PDB) [35, 28] which stores all the edges with their probabilities, but research on PDB has mainly focused on computing expectations as opposed to the pmf and variance [28]. According to the results in the Appendix of our full version [1], BM [47] performs about 40% faster than the PDB method [28] on computing expectations. This calls for a more effective, informative, and efficient motif counting algorithm, as we will discuss next.

3. PRELIMINARIES

We now give the formal definitions of motif counting for deterministic and uncertain graphs. Table 2 summarizes the notations used in this paper.

3.1 Deterministic Graphs

A *deterministic graph* G is a pair (V, E), where V is a set of nodes, and $E \subseteq V \times V$ is a set of edges. A *motif* $M = (V_M, E_M)$ is a connected graph, which is often regarded as a building block of modern networks. In the literature of motif-based-analysis, G is an undirected simple graph, and M is "small" (i.e., from three to seven nodes) [44, 17, 24, 42, 5, 48]. We also adopt these assumptions, noting that our solutions can support motifs of arbitrary sizes. Table 1 shows 3-node and 4-node motifs. An *induced subgraph* $S = (V_S, E_S)$ of $G = (V, E)$ is a subgraph satisfying $V_S \subset V$ and $E_S = (V_S \times V_S) \cap E$, meaning that S contains all edges of G whose endpoints are in V_S.

Table 2: Notations

Notation	Description
$\mathcal{G} = (G, P)$	An uncertain graph
$G = (V, E)$	The backbone graph of $\mathcal{G}$
$G_i \sqsubseteq \mathcal{G}$	G_i is a possible world from $\mathcal{G}$
d	The maximum degree among all nodes V
M	A motif M
$\mathcal{M}$	The motif closure of M
$S \simeq_G M$	S is a motif-instance of M in G
$\mathcal{I}_{M,G}$ (or $\mathcal{I}_M$)	The set of all motif-instances of M in G
$C_{M,G}$	The motif count of M in G
$C_{M,\mathcal{G}}$	The motif count of M in $\mathcal{G}$
$f_{M,\mathcal{G}}$	The pmf of $C_{M,\mathcal{G}}$
$k_{\max}$	The maximum value $C_{M,\mathcal{G}}$ can take
L (or L_e)	L-tables (or the L-table on the edge e)
B	The bit-string dictionary

Definition 1. A **motif-instance [5]** of $M = (V_M, E_M)$ in G is an induced subgraph $S = (V_S, E_S)$ in G which is *isomorphic* to M, i.e., there exists a bijection $h : V_S \to V_M$ such that $\forall u, v \in V_S, (u, v) \in E_S \iff (h(u), h(v)) \in E_M$.

We use $S \simeq_G M$ to denote that $S \in G$ is a motif-instance of M. Also, $\mathcal{I}_{M,G}$ is the set of all motif-instances of M in G, and $C_{M,G} = |\mathcal{I}_{M,G}|$ is the *number of motif-instances* of M in G.

Induced and non-induced subgraph semantics. In Definition 1, we define a motif-instance as an induced subgraph. This is also an assumption commonly used in the motif counting literature [5, 13, 11, 30]. As for *non-induced subgraph* semantics, a motif-instance need not be an induced subgraph. As discussed in [30], the frequency of a given motif under non-induced subgraph semantics is the linear combination of the induced counts [30].

Example 1. Under induced subgraph semantics, there are three *2-star* (∴) in G_2 of Fig. 2d, due to the subgraphs induced by {a, b, c}, {a, c, d}, and {a, c, e}. For non-induced subgraph semantics, the number of *2-star* motifs is 6 (including the 3 subgraphs mentioned above, and 3 more *2-star* subgraphs derived from the same node set {c, d, e}. □

In the rest of the paper, we will focus on the induced subgraph semantics. We will also explain our solutions can be extended to handled non-induced subgraph semantics.

3.2 Uncertain Graphs

In this paper, we study the motif counting on an uncertain graph model based on the definitions in [43, 14]:

Definition 2. An **uncertain graph** is a pair $\mathcal{G} = (G, P)$, where $G = (V, E)$ is a deterministic graph called the *backbone graph* [14] of $\mathcal{G}$, and $P : E \to (0, 1]$ is a function that assigns to each edge $e \in E$ an *existential probability* $P(e)$.

This uncertain graph model is used in various domains, such as wireless networks [21], biological networks [46], and reliability graphs [4].

Fig. 2a illustrates $\mathcal{G}$. Each edge has an existential probability. For instance, edge (b, c) exists with a probability of 0.5.

Following the *Possible World Semantics* (PWS) [2, 15], $\mathcal{G}$ can be interpreted as the set $\{G_i = (V, E_{G_i}) | E_{G_i} \subseteq E\}$ of all $2^{|E|}$ possible deterministic graphs G_i. Each possible graph G_i represents one possible outcome when randomly determining whether a certain edge should exist or not. We use $G_i \sqsubseteq \mathcal{G}$ to denote that G_i is a possible deterministic graph extracted from $\mathcal{G}$. The existential probability $\mathbb{P}[G_i]$ of a possible world $G_i = (V, E_{G_i}) \sqsubseteq \mathcal{G}$ can be computed by:

$$\mathbb{P}[G_i] = \prod_{e \in E_{G_i}} P(e) \prod_{e \in E \setminus E_{G_i}} (1 - P(e)) \,. \tag{1}$$

Example 2. Fig. 2d illustrates G_2, a possible world extracted from $\mathcal{G}$ of Fig. 2a. By Eq. 1, the existential probability of G_2 is $\mathbb{P}[G_2] = 0.25$. □

Given $\mathcal{G}$ and M, our goal is to evaluate the occurrence statistics of M – **pmf**, **mean**, and **variance**. Since each G_i exists with a probability, the number of motif-instances of M in $\mathcal{G}$, $C_{M,\mathcal{G}}$, is a **random variable**. We use $f_{M,\mathcal{G}}(k)$ to denote the probability mass function (pmf) of $C_{M,\mathcal{G}}$, where k is a non-negative integer ranging over all possible values $C_{M,\mathcal{G}}$ can take. We use $k_{\max}$ to denote the maximum value of $C_{M,\mathcal{G}}$, i.e., $k_{\max} = \max_{G_i} C_{M,G_i}$. In other words, $f_{M,\mathcal{G}}(k)$ is the probability that $C_{M,\mathcal{G}} = k$. The **pmf** $f_{M,\mathcal{G}}$ can then be described as:

$$f_{M,\mathcal{G}}(k) = \sum_{\substack{G_i \sqsubseteq \mathcal{G} \\ C_{M,G_i} = k}} \mathbb{P}[G_i], \quad 0 \le k \le k_{\max} \,. \tag{2}$$

Example 3. Fig. 2b shows a pmf of $C_{M_{3,2},\mathcal{G}}$. $M_{3,2}$ is a *triangle* motif and $\mathcal{G}$ is the uncertain graph of Fig. 2a. Figs. 2c to 2f show all four possible worlds of $\mathcal{G}$ with corresponding existential probabilities. Using Eq. 2, we can compute $f_{M_{3,2},\mathcal{G}}(0) = \mathbb{P}[G_1] = 0.25$, $f_{M_{3,2},\mathcal{G}}(1) = \mathbb{P}[G_2] + \mathbb{P}[G_3] = 0.5$ and $f_{M_{3,2},\mathcal{G}}(2) = \mathbb{P}[G_4] = 0.25$. □

Using the pmf, the **mean** $\mathrm{E}[C_{M,\mathcal{G}}]$ and the **variance** $\mathrm{Var}[C_{M,\mathcal{G}}]$ can then be computed easily. According to the pmf $f_{M_{3,2},\mathcal{G}}$ in Fig. 2b, we can derive the mean $\mathrm{E}[C_{M_{3,2},\mathcal{G}}] = 1$ and the variance $\mathrm{Var}[C_{M_{3,2},\mathcal{G}}] = 0.5$.

From the discussions above, we can see that once we obtain the pmf $f_{M,\mathcal{G}}$ (by Eq. 2), then we can get the mean and variance accordingly. However, Eq. 2 is prohibitively expensive, because of the $2^{|E|}$ possible worlds of $\mathcal{G}$, which means, for a motif M, we need to count motif-instances in an exponential number of deterministic graphs. Thus, it is infeasible to compute the accurate pmf on even moderately large graphs.

THEOREM 1. *It is #P-hard to compute $f_{M,\mathcal{G}}$.*

PROOF. It is proved that it is #P-hard to check for the presence of a path of length 2 (i.e., a *2-star*) in an uncertain graph, in Example 3.3 of [45]. The *2-star* presence problem can be reduced to the motif counting problem for *2-star* in polynomial time because we can get the probability that at least one *2-star* exists in the uncertain graph using the pmf of the *2-star* count. □

Our goal is thus to approximate $f_{M,\mathcal{G}}$ effectively and efficiently. Next, we present our two solutions, namely PGS and LINC, which utilize the sampling framework shown in Fig. 3.

4. POSSIBLE GRAPH SAMPLING (PGS)

Our first solution called the *Possible Graph Sampling* (PGS), computes an approximate pmf of a motif M for a given uncertain graph $\mathcal{G}$. As shown in Fig. 4, it first samples N randomly chosen possible worlds G_i of $\mathcal{G}$. For each G_i, a deterministic motif counting algorithm is used to compute the frequency of M. The counts of M in the sampled graphs are then aggregated to obtain the statistics of M.

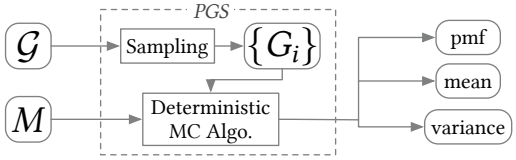


Figure 4: PGS

In detail, each G_i is obtained by Monte-Carlo sampling, based on the existential probabilities of the edges of $\mathcal{G}$. We then use a motif counting algorithm for deterministic graphs, e.g., `Escape` [42], to calculate the occurrences of instances of M in G_i. The number of motif-instances of M in the N sampled graphs are aggregated, yielding the approximate pmf, $\hat{f}_{M,\mathcal{G}}(k)$:

$$\hat{f}_{M,\mathcal{G}}(k) = \frac{|\{G_i | C_{M,G_i} = k, 1 \le i \le N\}|}{N} \,. \tag{3}$$

PGS can be easily extended to support non-induced counts; we just need to use a non-induced version of `Escape`. We denote with `PGS` the induced and with `PGS-n` the non-induced version of PGS.

Next, we study the important problem of deciding the lower bound of N for attaining accuracy guarantees. Then, in Section 4.2, we discuss the deterministic motif-counting algorithms that can be used in PGS, and its complexity.

4.1 Sample Size Analysis

We now study the sample size N needed to ensure that the absolute error of each value $\hat{f}_{M,\mathcal{G}}(k)$ is less than ε with probability $1 - \delta$. We first focus on a particular value of k and its corresponding $\hat{f}_{M,\mathcal{G}}(k)$ value. Let us define N random variables, $X_1, X_2, \ldots, X_N$, where $X_i = 1$ if $C_{M,G_i} = k$, otherwise $X_i = 0$. Because $G_1, G_2, \ldots, G_N$ are drawn randomly from $\mathcal{G}$, $X_1, X_2, \ldots, X_N$ are independent identically distributed (i.i.d.) Bernoulli random variables and $X_i = 1$ with probability $f_{M,\mathcal{G}}(k)$, according to Eq. 2. Then, Eq. 3 can be rewritten as:

$$\hat{f}_{M,\mathcal{G}}(k) = \frac{\sum_{i=1}^{N} X_i}{N} \tag{4}$$

Based on Hoeffding Inequality [25], the probability that the absolute error of $\hat{f}_{M,\mathcal{G}}(k)$ is larger than ε, i.e., $\mathbb{P}\left[|\hat{f}_{M,\mathcal{G}}(k) - p| \ge \varepsilon\right]$, satisfies:

$$\mathbb{P}\left[|\hat{f}_{M,\mathcal{G}}(k) - p| \ge \varepsilon\right] \le 2e^{-2\varepsilon^2 N} \le \delta. \tag{5}$$

Thus, the mean of N sample observations, $X_1, X_2, \ldots, X_N$ approximates $f_{M,\mathcal{G}}(k)$, whose absolute error is less than ε with probability $1 - \delta$. Given ε and δ, we have $N \ge \frac{\ln \frac{2}{\delta}}{2\varepsilon^2}$ for $\hat{f}_{M,\mathcal{G}}(k)$.

To extend this to all possible values of k in the approximate pmf, i.e., bounding $\mathbb{P}\left[\bigwedge_{0 \le k \le k_{\max}} |g(k)| < \varepsilon\right]$, where $g(k) = \hat{f}_{M,\mathcal{G}}(k) - f_{M,\mathcal{G}}(k)$ and $k_{\max} = \max_{G_i} C_{M,G_i}$, we apply the Fréchet Inequality [20] on all $k_{\max} + 1$ values of the pmf. For t possibly dependent random events $X_1, \ldots, X_t$, the probability, $\mathbb{P}[\bigwedge_{1 \le i \le t} X_i]$, that all events occur satisfies:

$$\mathbb{P}\left[\bigwedge_{1 \le i \le t} X_i\right] \ge 1 - \sum_{i=1}^{t} (1 - \mathbb{P}[X_i]). \tag{6}$$

If $\mathbb{P}[|g(k)| \geq \varepsilon]$ is less than $\delta' = \frac{\delta}{k_{\max}+1}$ for every k between 0 and $k_{\max}$, then $\mathbb{P}\left[\bigwedge_{0\leq k\leq k_{\max}} |g(k)| < \varepsilon\right] > 1-\delta$ will be satisfied, according to the Fréchet Inequality. Applying the Hoeffding Inequality again with δ', we can obtain that

$$N \geq \frac{\ln \frac{2(1+k_{\max})}{\delta}}{2\varepsilon^2} \tag{7}$$

random samples are sufficient to bound the absolute error of the approximate pmf within ε with probability at least $1-\delta$. It is hard to obtain the exact value of $k_{\max}$ in Eq. 7. Thus, we will give an upper bound of $k_{\max}$ in Section 5.

4.2 Complexity of PGS

In PGS, the motif M in each possible world G_i is counted from scratch. Counting the number of motif-instances in a possible world can be very expensive, e.g., the counting algorithm given by [5] needs $O(|E_{G_i}|d^2)$ time to count 4-node motifs in one possible world, where d denotes the maximum degree among all vertices. `Escape` [42] extends the motif counting on deterministic graphs up to 5-node motifs and designs a more elegant enumeration algorithm, but its worst time complexity is $O(|E_{G_i}|d^{|V_M|-2})$ and its worst space complexity is $O(|V|+|E|+|\mathcal{I}_T|)$, where $|\mathcal{I}_T|$ refers to the number of *triangle* instances in G.[3] Note that for some motifs, the algorithm does not need the $O(|\mathcal{I}_T|)$ memory space, e.g., $O(|V|+|E|)$ space is needed for counting *2-star* motif (∴). Thus, the overall time complexity of PGS is $O(N \cdot |E|d^{|V_M|-2})$.

Fortunately, it turns out that we can vastly reduce the runtime of the algorithm by making use of some additional data structures which help capture just the difference in the motif count when switching from one possible world to another. In Section 5, we will introduce such an algorithm, LINC, which exploits the similarity between different possible worlds and maintains the differences in motif counts between them.

An alternative baseline. We remark that a straightforward alternative for PGS is to first compute all motif-instances of any motif in the motif closure of M in the backbone graph G. (The motif closure is conceptually the 'superset' of motif M. For example, in Fig. 5, the motif closure of motif $M_{4,5}$ contains $M_{4,5}$ and $M_{4,6}$ shown in the figure. We will discuss motif closure in detail in Section 5.) Then, we repeatedly sample N possible graphs G_i, with $1 \leq i \leq N$, and iterates over the list of motif-instances to see which of them are the motif-instances of M in G_i. This can be done by checking whether the edges of the motif-instance exist in G_i following Definition 1. We call this method *backbone sampling* (or BS). BS has the same worst-case time complexity with PGS. We will evaluate PGS and BS in Section 8. Similar to PGS, BS can also be extended to support non-induced counts, by computing the linear combination of the induced counts, as discussed in Section 3.1. We denote with `BS` the induced and with `BS-n` the non-induced version of the BS algorithm.

5. LINKING AND COUNTING (LINC)

The main problem of PGS is that it counts motifs in each sampled possible world from scratch. This yields a high computational overhead. To overcome this issue, we propose LINC (Linking and Counting), which computes the difference in the motif count between two different possible worlds efficiently. The algorithm performs motif counting in two steps. In the first phase, a linking table (L-table) is built to capture the influence of each edge on the motif count. In the second phase, the algorithm identifies for each sample the difference in edges with respect to the previous sample and updates the count accordingly.

[3]Refer to [5, 42] for a more detailed complexity analysis for the deterministic part of the PGS method.

To understand the mechanics of LINC, we next introduce the notion of a *motif closure*.

Definition 3. **Motif Closure.** The *motif-closure* $\mathcal{M}$ of a motif $M = (V_M, E_M)$ is the set of all motifs $M' = (V_{M'}, E_{M'})$ s.t. $E_M \subseteq E_{M'}$ and $V_M = V_{M'}$.

Intuitively, the motif closure $\mathcal{M}$ of a motif M is the set of all possible supergraphs of M which can be obtained by adding more edges but without adding more vertices. For our algorithm LINC, we need to find the motif closure $\mathcal{M}$ of M because each instance of a motif $M' \in \mathcal{M}$ in the backbone graph G has the potential to degenerate into a motif-instance of M in one possible world (because some of the edges are removed in the specific possible world). Example 4 illustrates this process. This also means that $\sum_{M'\in\mathcal{M}} C_{M',G} = \sum_{M'\in\mathcal{M}} |\mathcal{I}_{M',G}|$ provides an upper bound for $k_{\max}$ discussed in Section 4.1.

Example 4. Fig. 5 depicts the process of obtaining the motif closure for different 4-node motifs. For a given motif M (e.g., $M_{4,4}$), the motif closure $\mathcal{M}$ can be obtained by following the arrows until the *4-clique* (⊠) is reached. All motifs M' encountered during a traversal of this directed graph (including start and end points) form the closure of M. □

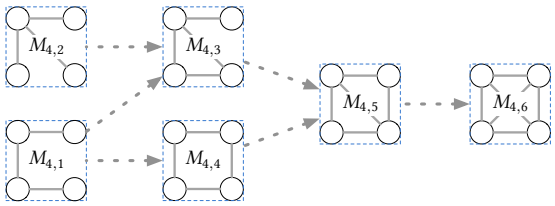


Figure 5: Computing Motif Closures

Fig. 6 illustrates the workflow of LINC. We will first describe the linking phase, illustrated by the top part of Fig. 6. As a first step, we construct the motif closure $\mathcal{M}$ of M, and the backbone graph G of $\mathcal{G}$. Next, we have to find all instances of each motif $M' \in \mathcal{M}$. Finally, we can build the L-table which links an edge e to all those motif-instances of $M' \in \mathcal{M}$ which contain edge e.

In the second phase of LINC, illustrated by the bottom part of Fig. 6, we sample N different possible worlds $G_1, \ldots, G_N$ of $\mathcal{G}$ and identify for each G_i the set $E_{\Delta i} = (E_{G_i} - E_{G_{i-1}}) \cup (E_{G_{i-1}} - E_{G_i})$, with $G_0 := G$. Using $E_{\Delta i}$ and the L-table from the first phase, we can quickly identify the motif-instances of motifs $M' \in \mathcal{M}$ which are affected by the addition and removal of the edges when switching from G_{i-1} to G_i. To count the instances of M, we will successively investigate each edge $e \in E_{\Delta i}$. For each edge e we identify whether motif-instances of M will appear or dissapear because edge e is added or removed, respectively, and update the count accordingly. Example 5 illustrates this update process. We will discuss the efficient implementation of LINC in detail in Section 6.

Example 5. Fig. 7 illustrates how the count of the motif M is updated for the possible world G_i based on its count for G_{i-1}. For the given motif M (⊡), $\mathcal{M}$ consists of three different motifs (⊡, ⊟, ⊠). On the left-hand side of Fig. 7, we illustrate the graphs for G_{i-1} and G_i. Note that there is one edge, e_1, which dissapears when switching from G_{i-1} to G_i, and another edge, e_2, which appears. Hence, $E_{\Delta i} = \{e_1, e_2\}$.

As a first step, LINC updates the count according to the dissapearance of e_1 (the order is arbitrary): as e_1 disappears, we gain one ⊡({`B, C, E, F`}), which is degenerated from a motif-instance of ⊟.

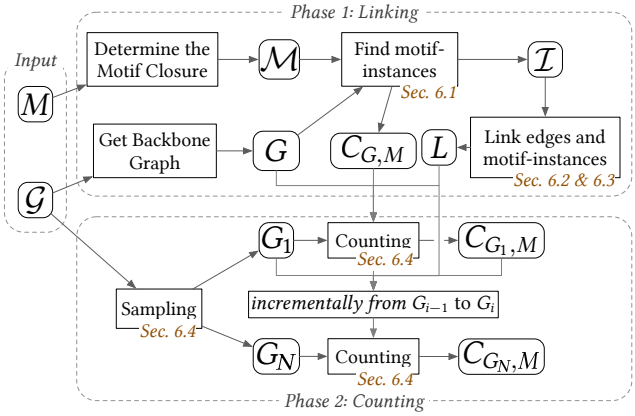


Figure 6: Flowchart of LINC

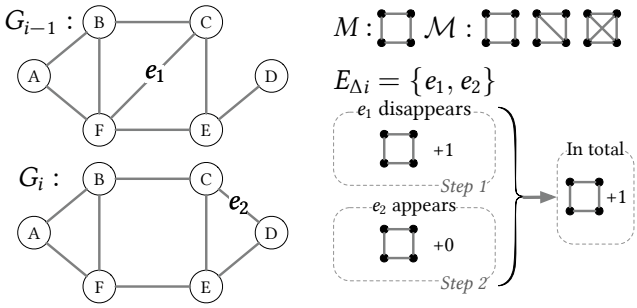


Figure 7: Example of updating and counting in LINC

In the second step, LINC updates the count according to the appearance of e_2 (after e_1 is already removed): as e_2 appears, there is no motif-instance of M( ) appearing or disappearing.

After the steps described above, LINC has the newly updated count for M. As it turns out, the count increases by 1 when G_{i-1} is changed to G_i. □

The intuition behind computing the motif counts incrementally from one sample graph to the next is that the two successively sampled possible worlds tend to be similar to each other:

THEOREM 2. *Let $G_{i-1} = (V, E_{G_{i-1}})$ and $G_i = (V, E_{G_i})$ be two successively sampled possible worlds during the sampling process, with $i > 1$ and $\bar{P} = \frac{\sum_{e \in E} P(e)}{|E|}$ be the average existential probability over all edges in $\mathcal{G}$. Let $E_{\Delta i} = (E_{G_i} - E_{G_{i-1}}) \cup (E_{G_{i-1}} - E_{G_i})$ be the relative edge change, or the set of edges inserted or deleted from $E_{G_{i-1}}$ to become E_{G_i}. Then, $\mathrm{E}[\frac{|E_{\Delta i}|}{|E|}]$, the expected relative edge change over all the sampled possible worlds, satisfies $\mathrm{E}\left[\frac{|E_{\Delta i}|}{|E|}\right] \leq 2(\bar{P} - \bar{P}^2) \leq \frac{1}{2}$.*

PROOF. The probability that $e \in E_{\Delta i}$ is $P[e \in E_{\Delta i}] = 2 \cdot P(e) \cdot (1 - P(e))$. Thus,

$$\mathrm{E}\left[\frac{|E_{\Delta i}|}{|E|}\right] = \frac{1}{|E|}\mathrm{E}\left[|E_{\Delta i}|\right] = \frac{1}{|E|}\sum_{e \in E} P[e \in E_{\Delta i}]$$
$$= \frac{1}{|E|}\sum_{e \in E} 2\left(P(e) - P(e)^2\right) \leq 2(\bar{P} - \bar{P}^2) \leq \frac{1}{2}.$$

□

Notice that only when $\forall e \in E, P(e) = \frac{1}{2}$, $\mathrm{E}[\frac{|E_{\Delta i}|}{|E|}]$ can be equal to $\frac{1}{2}$. In the datasets we tested, the expected relative edge change $\mathrm{E}[\frac{|E_{\Delta i}|}{|E|}]$ is much less than $\frac{1}{2}$ as reported in Table 3.

6. IMPLEMENTATION OF LINC

We are now ready to discuss the implementation of LINC. As an overview, it consists of 3 steps: (1) generate motif instances from the backbone graph of G (Section 6.1); (2) encode the motif instances into L-tables (Sections 6.2 and 6.3); and (3) perform incremental counting on the L-tables (Section 6.4). In Section 6.5, we present the full algorithm of LINC.

6.1 Finding Motif-instances

The first step in our LINC algorithm is to find all instances of motifs in $\mathcal{M}$, denoted by $\mathcal{I}_{\mathcal{M}}$. A motif-instance $S \in \mathcal{I}_{\mathcal{M}}$ may have multiple isomorphisms to a motif $M' \in \mathcal{M}$. Thus, when finding the motif-instances in G via isomorphisms, we should either avoid finding duplicated motif-instances or have an efficient method to remove duplicates. For this, we resort to *symmetry breaking* (SimB) [23], which validates exactly one isomorphism among several isomorphisms from $V_{M'}$ to V_S for each instance S. With the help of SimB, we can prune some replicates early during the instance searching process and avoid generating the replicated instances.

3-node and 4-node motifs are more commonly used in different domains, e.g., in social and life sciences. 3-node motif counts are treated as important structural features of social networks [22, 18]. In biology, scientists usually focus on 3-node or 4-node motifs to find functional units of biological processes in cells [48]. To address this, we have developed two efficient motif-instance searching algorithms (named `MF3` and `MF4`) for 3-node and 4-node motifs, respectively, which integrates the aforementioned symmetry breaking technique and makes use of the structural features of 3-, 4-node motifs. In `MF3` and `MF4`, we search the 3-node and 4-node motif-instances by enumerating each edge and the neighbors of its endpoints. If the subgraph induced by the endpoints and their neighbor(s) for the corresponding edge is isomorphic to the motif and it obeys the SimB conditions, it is recorded as a valid motif-instance. We give the algorithm details in the appendix of our full version [1]. The time complexity of the algorithm is $O(|E|d^2)$ for `MF4` and $O(|E|d)$ for `MF3`, where d is the maximum vertex degree. The space complexity is $O(|\mathcal{I}_{\mathcal{M}}|)$, i.e., the number of motif-instances of motifs in $\mathcal{M}$.

To find all instances of a motif $M' \in \mathcal{M}$ with more than 4 nodes in a graph G, we resort to state-of-the-art subgraph isomorphism algorithms (e.g., `VF3` [12]) with minor modification. The outputs of `VF3` are all isomorphism mappings, $O \subset V_{M'} \times V$, while we only validate one isomorphism mapping to avoid indexing duplicate motif-instances. Therefore, we perform the symmetry breaking check by using SimB on the outputs of `VF3` to avoid duplicates. Table join algorithms [41, 49] provides the alternative options for `VF3`, which share the same worst-case complexity as `VF3`. Here, we choose `VF3` because `VF3` is designed and optimized for graphs.

6.2 Encoding of Motif-instances

As described in Section 5, we need to track the number of motif-instances of the motif M. To facilitate this process, we encode each motif-instance into a bit-string according to the existing statuses of its edges.

To define which bit should correspond to which edge for a motif-instance $S \simeq M' \in \mathcal{M}$, we define an arbitrary bijection $l : V_S \to \mathbb{N}_{[1,|V_S|]}$, i.e., every node $u \in V_S$ is mapped to a distinct integer in $\mathbb{N}_{[1,|V_S|]}$. With this transformation, every pair $(u, v) \in V_S \times V_S$

where $l(u) > l(v)$ can be mapped to a distinct position in the bit-string s by the following formula, where $l(u) > l(v)$:

$$s[\binom{l(u)-1}{2} + l(v)] = \begin{cases} 1 & \text{if } (u,v) \in E_S, \\ 0 & \text{if } (u,v) \notin E_S. \end{cases} \quad (8)$$

The intuition behind this encoding formula is that we successively consider more nodes: first, we encode the edge between the first and second node, then we encode the edges between the first and third, and between the second and third node, and so on. Whenever we consider a new node u, we have to encode all the pairs (u, v) with all previous nodes v. When encoding the pairs for the $l(u)$-th node, we already encoded $\binom{l(u)-1}{2}$ pairs before and hence, we will use this number as an offset for encoding the $l(v)$ new pairs which include node u. Example 6 illustrates this process.

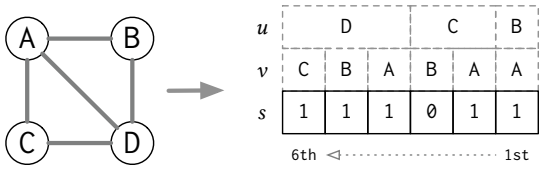


Figure 8: Example of encoding motif $M_{4,5}$

Example 6. Fig. 8 illustrates the process of encoding a motif-instance of $M_{4,5}$. For this, we arbitarily define $l : V_S \to \mathbb{N}_{[1,4]}$ as $\{\texttt{A} \to 1, \texttt{B} \to 2, \texttt{C} \to 3, \texttt{D} \to 4\}$. The first step is to encode the pair (A, B) (first and second node). Since $\binom{l(B)-1}{2} = \binom{1}{2} = 0$ there is no offset which we have to consider, yet. Moreover, $l(A) = 1$ and hence, the pair (A, B) is assigned to the first bit. Next, we will consider the pairs which include node C (the third node) as the node with larger image under l. As we already encoded $\binom{l(C)-1}{2} = \binom{2}{2} = 1$ pairs, we now have an offset of 1. Hence, the pairs (A, C) and (B, C) will be assingned to the positions $1+l(A) = 1+1 = 2$ and $1+l(B) = 1+2 = 3$ in the bit-string s. Finally, we consider the pairs which include node D via the process mentioned above. The pairs (A, D), (B, D), and (C, D) will be assingned to the positions 4, 5, and 6, respectively, in the bit-string s. □

Once we have encoded each motif-instance $S \simeq M' \in \mathcal{M}$, we need a way to quickly decide whether S degenerates to an instance of the motif M in one of the possible worlds. For this, we flip the bit of an edge in the bit-string of S, whenever this edge appears/disappears. The new subgraph $S^\star$ can be a motif-instance of the motif M but does not have to be. We use an additional data-structure, named *M-code*, to store all possible bit-strings which correspond to the motif M. Using the M-code, we can decide if $S^\star \simeq M$ by simply looking up the bit-string of $S^\star$ in this data structure.

To construct the M-code, we need to enumerate all possible bijection mappings l from $V_{M'}$ to $\mathbb{N}_{[1,|V_{M'}|]}$ and use Eq. 8 to get all possible different bit-strings which relate to the same motif M'. There are $|V_{M'}|!$ different mapping functions l in total. However, some of the mappings might yield the same bit-string.

The astute reader may have noticed that if we change the bijection, the encoded bit-string s of a motif-instance may also change. For instance, if we change l to $\{\texttt{A} \to 1, \texttt{B} \to 4, \texttt{C} \to 3, \texttt{D} \to 2\}$ in Example 6, s will change to 011111. Thus, the first concern is whether we can uniquely identify one specific motif for a given bit-string s. The answer is "yes":

THEOREM 3. *Let $S_1 \simeq M_1$ and $S_2 \simeq M_2$ be instances of two different motifs and let $l_1 : V_{S_1} \to \mathbb{N}_{[1,|V_{S_1}|]}$, $l_2 : V_{S_2} \to \mathbb{N}_{[1,|V_{S_2}|]}$ be two arbitrary bijections and s_1, s_2 be the bit-strings of S_1 and S_2, respectively. The following proposition holds:*

$$M_1 \neq M_2 \Rightarrow s_1 \neq s_2 \; \forall l_1, l_2. \quad (9)$$

PROOF. We will prove this by contraposition. Let $S_1 \simeq M_1$ and $S_2 \simeq M_2$ be two motif-instances which yield the same bit-string s under bijections l_1 and l_2, respectively. Let us further denote with $s_{(u,v)}$ the bit in s which encodes whether there is an edge between u and v. Consider the following bijection $f = l_2^{-1} \circ l_1 : V_{S_1} \to V_{S_2}$. Now (u, v) is an edge in S_1 if and only if $s_{(u,v)} = 1$ and $(f(u), f(v))$ is an edge in S_2 if and only if $s_{(f(u),f(v))} = 1$. But $s_{(u,v)} = s_{(f(u),f(v))}$ since they refer to the same bit string and hence, f is a graph-isomorphism. □

6.3 Building the L-tables

The L-tables are data structures which help us to quickly identify which motif-instances are affected by the appearance/disappearance of an edge. We denote with $L_{(u,v)}$ the L-table associated to edge (u, v). It stores the linking relationships between (u, v) and those motif-instances containing (u, v). Each pair $\langle id, p\rangle$ in $L_{(u,v)}$ indicates that edge (u, v) contributes to the p-th edge in the id-th motif-instance. The bit-strings of the motif-instances are stored in a simple dictionary, the *bit-string dictionary* B, recording the mapping between id and the bit-string of the corresponding motif-instances.

Example 7. Fig. 9 illustrates the L-table for the edge (b, c) w.r.t. potential motif-instances[4] of the *2-star* motif $M_{3,1}$ (∴) with id=2, 3, 4. Fig. 9a shows the L-table, Fig. 9b, with an additional column V (for illustration only, not in the actual implementation) to indicate to which motif-instance the bit-string belongs, shows the bit-string dictionary, and Fig. 9c shows the M-code.

The L-table (Fig. 9a) for edge (b, c) indicates that three different subgraphs (potential motif-instances) are affected by this edge (id=2, 3, and 4). The position (column p) indicates which bit of the corresponding motif is affected. The bits in a bit-string are counted from right to left and from 1 up. For example, when the edge (b, c) disappears, the third bit of the bit-string of the motif with $id = 3$ in the dictionary B (Fig. 9b) should be flipped to 0 (101 → 001), and when the edge appears, the bit should be flipped to 1 (001 → 101). Once the bits are flipped, we can use the M-code (Fig. 9c) to quickly identify whether the new subgraph is a motif-instance of the motif M. For example, the subgraph with $id = 3$ is an instance of $M_{3,1}$ if edge (b, c) exists (bit-string 101), but as soon as edge (b, c) disappears, the subgraph is not anymore an instance of the motif $M_{3,1}$ (bit-string 001). □

6.4 Sampling and Counting

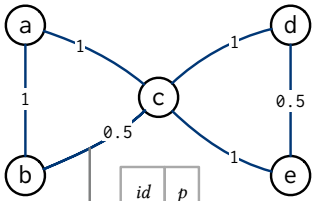
To calculate the approximate pmf $\hat{f}_{M,\mathcal{G}}$, the Monte-Carlo estimator draws N possible graphs and then computes the motif count on every possible graph. Finally, it aggregates the counts to get the approximate pmf using Eq. 3. To generate a sample graph from uncertain graph $\mathcal{G}$, we need to draw $|E|$ random numbers to determine the existence or non-existence of all $|E|$ edges of $\mathcal{G}$.

After obtaining the sample graphs, we can incrementally update the count of the motif M as described in Section 5. Using the encoding of motif-instances (Section 6.2) and the L-tables (Section 6.3) can speed up this process by exploiting bit-wise operations and table-lookups.

6.5 Algorithm

In this section, we discuss the complete algorithm for LINC, listed in Algorithm 1.

[4]The subgraphs are motif-instances of some possible world G_i but not necessarily motif-instances of the current possible world we are investigating.



id	p
2	1
3	3
4	3

(a) L-table on (b, c), $L_{(b,c)}$

id	s	V
1	111	{c,d,e}
2	111	{a,b,c}
3	101	{b,c,d}
4	101	{b,c,e}
5	101	{a,c,d}
6	101	{a,c,e}

(b) Bit-string dictionary, B

s	110	101	011
M	$M_{3,1}$	$M_{3,1}$	$M_{3,1}$

(c) M-code for $M_{3,1}$(∴)

Figure 9: Example of L-table, bit-string dictionary

As input, the algorithm needs an uncertain graph $\mathcal{G} = (G, P)$, a motif M, error tolerance ε and a confidence coefficient δ. Given these inputs, the algorithm will produce an approximate probability mass function (pmf) of the motif count of M. We split up the algorithm into two steps. The first step, *linking* (Lines 1–12), preprocesses the graph to speed up the second step, *counting* (Lines 13–31).

We will first discuss the linking step (Lines 1–12). First, the motif closure of M has to be calculated (Line 1). Then, we use either `VF3` (if the motif M has more than 4 nodes) or `MF3`/`MF4` (if the motif has 3 or 4 nodes) to find all instances of any motif $M' \in \mathcal{M}$ (Line 2) and store them in $\mathcal{I}$. In Line 3, we initialize some data structures needed for the algorithm. We then go through each motif-instance (Lines 4–12) and link them with its edges. For this, we iterate through all edges $e \in E_S$ for a given motif-instance $S \in \mathcal{I}$ (Lines 8–11), determine the edge's position in the bit-string (Line 9) according to some arbitrary bijection l (Line 6), initialize the respective bit to 1 (Line 10)[5], and add this information to the L-table of e (Line 11).

To sample and count the motifs $M' \in \mathcal{M}$, we first initialize a vector T (Line 13), whose length equals the number of edges in G, to indicate that the previous state of the edges was `true`, which means that the edges existed in the previous possible world (which is the backbone graph G for the first sample). We determine the number of samples N according to Eq. 7 (Line 14). After this, we can do the actual sampling of possible worlds and aggregate the counts (Lines 15–21). For this, we iterate through each edge $e \in E$ of the backbone graph (Lines 16–20) and draw a uniform random number between 0 and 1 to determine whether the edge should exist in the current possible world (Line 17). If the random number t is smaller than the existential probability $P(e)$, the edge will exist in the current possible world. We compare the outcome of this comparison with the previous existence state $T(e)$ of edge e to determine whether the existence state changes when switching from the possible world G_{i-1} to G_i (Line 18). If this is the case, we change the status of $T(e)$ (Line 19) and update the counts using the function `UpdateCount` (Line 20). Finally, we update the pmf of M by using the count of the motif M (Line 21).

[5]The count of the first possible world G_1 will be based on the backbone graph G, where all edges exist. Hence, we initialize the bit-string with 1s.

The function `UpdateCount` updates the motif count $C_{M,G}$ when an edge e changes its status. First, the function indentifies the L-table L_e to retrieve all the $\langle id, p\rangle$ pairs which are affected by the change (Line 24). For each $\langle id, p\rangle$ pair, the function uses the corresponding bit-string (Line 25) to check whether its motif type is M according to the M-code via the `M-code` function (Line 26) and subtracts 1 from the respective motif count if the condition is satisfied (Line 27). Afterwards, the function updates the bit-string to reflect the change in the edge's existence status (Line 28), checks whether M is the motif type of the new bit-string s (Line 29), and adds 1 to the count of M if the condition is satisfied (Line 30). Finally, the bit-string dictionary B is updated with the new bit-string s (Line 31).

Algorithm 1: LINC

```
   Input: uncertain graph G = (G, P), motif M, error tolerance ε,
          confidence coefficient δ
   Output: approximate pmf f̂_{M,G}
   /* Step 1: Linking                                          */
1  M ← motif closure of M;
2  I ← VF3(G, M) or MF3(G, M) or MF4(G, M);
3  B ← ∅, L ← ∅, C_{M,G} ← |I_M|, id ← 0;
4  foreach S ∈ I do
5      s ← 0⃗;
6      id ← id + 1;
7      l ← choose an arbitrary bijection;
8      foreach e ∈ E_S do
9          p ← e's encoded position in s according to l;
10         s[p] ← 1;
11         add pair ⟨id, p⟩ to L_e;
12     append s to B;
   /* Step 2: Counting                                         */
13 T ← (True,...,True);
14 N ← (ln 2(1+|I|)/δ) / (2ε²);
15 for i ← 1 to N do
16     foreach e ∈ E do
17         t ← random number between [0, 1];
18         if (t > P(e)) ≠ T(e) then
19             T(e) ← ¬T(e);
20             UpdateCount(e);
21     update f̂_{M,G} using C_{M,G};
22 ──────────────────────────────────────────────
23 Function UpdateCount(e):
24     foreach ⟨id, p⟩ ∈ L_e do
25         s ← B[id];
26         if M-code(s, M) then
27             C_{M,G} ← C_{M,G} − 1;
28         s[p] ← 1 − s[p];
29         if M-code(s, M) then
30             C_{M,G} ← C_{M,G} + 1;
31         B[id] ← s;
```

Analysis. LINC only needs to find all motif-instances once in the backbone graph G, which yields a **time complexity** of $O(|E| \cdot d^{|V_M|-2})$ for the first phase, where d is the maximum vertex degree in the backbone graph G. Also, LINC incurs $O(|\mathcal{I}_\mathcal{M}|)$ time in each iteration in the second phase, which yields an overall time complexity of $O(|E| \cdot d^{|V_M|-2} + N \cdot |\mathcal{I}_\mathcal{M}|)$ for the whole algorithm. Although the time complexity of LINC is almost the same with PGS in the worst case, the number of motif-instances ($|\mathcal{I}_\mathcal{M}|$) is considerably smaller than the number $|E| \cdot d^{|V_M|-2}$. The reason is that during the $O(|E| \cdot d^{|V_M|-2})$ motif searching process, some unrelated subgraphs are also accessed and a motif-instance

may be traversed multiple times due to graph automorphism[6]. Besides, during the incremental updating process from one sample to the next, the number of updated instances is smaller than $|\mathcal{I}_{\mathcal{M}}|$, because $\mathrm{E}[|E_\Delta|] \leq \frac{1}{2}|E|$ according to Theorem 2. Thus, we expect that LINC will run much faster than PGS. The **space complexity** of LINC is $O(|V| + |E| + |\mathcal{I}_{\mathcal{M}}|)$.

Supporting non-induced subgraph semantics. Algorithm 1 can be easily extended to support motif counting under non-induced semantics. This is because, in that algorithm, we record all the motif instances of $M' \in \mathcal{M}$. Thus, their frequencies, under induced semantics, can be derived. The number of instances of M, under non-induced semantics, is the linear combination of the number of instances of all $M' \in \mathcal{M}$ (c.f. Sec 3.1). We use `LINC-n` (`LINC`) to denote the non-induced (induced) version of LINC.

7. OPTIMIZATIONS OF LINC

We now discuss how to extend and improve LINC. First, we study how LINC can terminate earlier under some conditions. Then, we show how the counting of certain motifs can be made faster. Finally, we discuss how to update LINC structures, when the uncertain graph is changed.

1. Early Stop Strategy. If the users are only interested in the mean or the variance, we could also estimate the relative error during runtime and, potentially, stop the program early before the sample size reaches N. Without loss of generality, we use the estimated variance as an example to illustrate the stopping rule following the strategy presented in [7]:

Given relative error tolerance ε and confidence level $1-\delta$, let $\mu_j = \frac{1}{j}\sum_{i=1}^{j} C_{M,G_i}$ denote the j-th sample mean and $\sigma_j^2 = \frac{1}{j-1}\sum_{i=1}^{j}(C_{M,G_i} - \mu_j)^2$ denote the j-th sample variance during the Monte-Carlo sampling. We define an auxiliary variable:

$$z_j = \begin{cases} z_{j-1}+1 & \text{if } L_{j-z_{j-1}} \leq \sigma_j^2 \leq U_{j-z_{j-1}}, \\ 1 & \text{otherwise} \end{cases} \quad (10)$$

where $L_{j-z_{j-1}} = (1-\frac{\varepsilon}{2})\sigma^2_{j-z_{j-1}}$ and $U_{j-z_{j-1}} = (1+\frac{\varepsilon}{2})\sigma^2_{j-z_{j-1}}$ denote the lower and upper limits of $\sigma^2_{j-z_{j-1}}$, respectively. According to Eq. 10, z_j denotes the number of adjacent sample variances including σ_j^2 which fall into the same upper and lower limit.

According to the *Central Limit Theorem*, σ_j^2 will converge to $\mathrm{Var}[C_{M,\mathcal{G}}]$ when j is large enough. This also means

$$\lim_{\zeta\to\infty} \mathbb{P}[\zeta] = \mathbb{P}\left[z_{j+1} = 1 | z_j = \zeta\right] = 0. \quad (11)$$

Thus, we need to find an optimal ζ^* satisfying $\mathbb{P}\left[z_{j+1} = 1 | z_j = \zeta^*\right] < \delta$ so that we can stop the algorithm when $z_j = \zeta^*$. Fortunately, $\mathbb{P}[\zeta]$ can be hypothesized well as a logarithmic series variate [16]:

$$\mathbb{P}[\zeta] = -\frac{c^\zeta}{\ln(1-c)\cdot\zeta}, \zeta = 1, 2, \ldots; 0 < c < 1, \quad (12)$$

according to [7]. Thus, ζ^* could be obtained by solving $\mathbb{P}[\zeta^*] < \delta$. It is observed in [7] that $c > 0.9$ is reasonable in Eq. 12 and we choose $c = 0.99$ for our implementation. To integrate the early stop strategy into LINC, we insert the "early stop check routine" after Line 21 in Algorithm 1.

2. Optimization for Specific Motifs. For some specific motifs, we can apply some formulas to further speed up their counting process. For example, the *2-star* motif ($M_{3,1}$) count in each sample graph G_i can be calculated by:

$$C_{M_{3,1},G_i} = \sum_{u\in V_{G_i}} \binom{d_u}{2} - 3\cdot C_{M_{3,2},G_i} \quad (13)$$

[6]An automorphism of graph G is an isomorphism from G to itself.

where d_u denotes the degree of node u in G_i. $\sum_{u\in V_{G_i}}\binom{d_u}{2}$ can be calculated in $O(|V|)$ time for each sample graph G_i. Using Eq. 13, we can avoid materializing the motif-instances of $M_{3,1}$ in the linking step, because we do not need to keep track of the motif-instances of $M_{3,1}$. We still need to link the instances of motifs $M' \in \mathcal{M}_{3,1} \setminus \{M_{3,1}\}$. By this method, we can further save some memory space and running time because of the decrease in the size of the L-tables.

Apart from *2-star* motif, similar equations can be used on some other motifs, e.g., *4-path* ($M_{4,1}$), *3-star* ($M_{4,2}$) and *4-tailedtriangle* ($M_{4,3}$). We provide the equations for those motifs in the Appendix of our full version [1]. Similar equations are also used to speed up the counting process in motif counting algorithms designed for deterministic graphs [5, 42].

3. Updating of LINC Structures. We next consider how the data structures of LINC should be changed to reflect the changes made to the uncertain graph. We consider three main kinds of graph updates.

•*Update of edge probability values*: the data structures of LINC do not need to be changed. This is because the probability change does not alter the backbone graph, and the data structure built based on it.

•*Edge deletion*: this is equivalent to updating the edge probability to zero, and LINC structures again do not need to be updated.

•*Edge insertion*: we check (1) which new motif-instances will be created due to the new edge, and (2) which existing instances need to be updated (we call them *affected motif-instances*). The following lemma limits the part of the graph $\mathcal{G}$ to be searched for cases (1) and (2).

LEMMA 1. *Let $\mathcal{G} = (G, P)$ be an uncertain graph, M be a k-node motif, and (u, v) be an edge just inserted to $\mathcal{G}$. Then the new or affected motif-instances only consist of nodes from $Q = \{w | \min(dis(w, u), dis(w, v)) \leq k-2, w \in V\}$, where $dis(w, u)$ means the shortest distance between w and u.*

PROOF. Assume there is a node w, which satisfies $\min(dis(w, u), dis(w, v)) > k-2$. The connected subgraph containing u, v, and w should have at least have $k+1$ nodes. Hence, the subgraph cannot be a k-node motif. □

Essentially, when a new edge (u, v) is inserted to $\mathcal{G}$, the new or affected motif-instances only contains set Q of nodes that are within a fixed network distance from nodes u and v. After Q has been derived, we then find the motif-instances on the graph induced by Q; those that do not contain u or v are not considered.

Note that vertex insertion or deletion can be treated as a sequence of edge updates, so their details are omitted here. Based on the cases above, the LINC structures are fixed. Then, the sampling process can be done again to obtain new motif instances.

8. EXPERIMENTS

In this section, we evaluate our algorithm on different aspects, i.e., effectiveness, efficiency, and the impact of parameters and optimization techniques.

8.1 Experiment Setup

We implement our algorithms in C++ with STL and Boost support and run the experiments on a 16GB memory machine with Intel(R) Core(TM) i7 CPU@2.3 GHz. Experiments are conducted on real-life datasets (Table 3):

PPI networks PPI networks capture the interaction relationships among proteins, where each node denotes a specific protein. Two proteins are linked if they are likely to interact with each other, with probabilities marked on edges denoting the interaction confidence: 1. **KRC**: the core dataset from [32] with

all probabilities larger than or equal to 0.27; 2. **KRE**: the extended interaction dataset from [32] which contains more edges with lower probabilities compared to the core dataset.

DBN A dataset of a Chinese online recommendation social network, douban[7] [33, 53]. The users are denoted by nodes and their friendships are captured by edges. The probabilities on the edges are injected by $(20, 10^{-3})$-*obfuscation* [10].

CAL A road network[8] of the State of California in USA [36, 33]. The nodes of the network are the intersections between roads and road endpoints, and the edges are road segments between intersections and road endpoints. Obfuscation technique is used on the road network to protect user privacy in location-based services [40]. The probabilities on the edges are also injected by $(20, 10^{-3})$-*obfuscation* [10].

Table 3: Statistics of Datasets

Dataset	$\|V\|$	$\|E\|$	min/max/avg $P(e)$	$\mathrm{E}[\frac{\|E_\Delta\|}{\|E\|}]$
real-world datasets				
KRC	2,708	7,123	0.27/0.99/0.68	30.7%
KRE	3,672	14,317	0.10/0.99/0.42	28.2%
DBN	154,908	288,416	0.00/1.00/0.90	8.5%
CAL	1,965,206	3,319,928	0.00/1.00/0.82	6.5%
synthetic datasets				
S1	200	396	0.28/0.98/0.68	30.6%
S2	2,000	3,996	0.28/0.98/0.68	30.8%
S3	20,000	39,996	0.28/0.98/0.68	31.0%
S4	200,000	399,996	0.28/0.98/0.68	31.0%
S5	2,000,000	3,999,996	0.28/0.98/0.68	31.0%
S6	20,000,000	39,999,996	0.28/0.98/0.68	31.0%

Besides, we use 5 synthetic datasets generated by *Barabási-Albert Model* [8] to test the scalability of LINC. The probabilities on edges are drawn from the histogram of edge probabilites of KRC, i.e., those datasets have similar edge probability distribution to KRC. Table 3 shows the dataset statistics. The expected relative edge change $\mathrm{E}[\frac{|E_\Delta|}{|E|}]$ (Theorem 2) is reported in the last column of Table 3.

We use the following **competitors** in our experiments to evaluate effectiveness and efficiency:

•**BM** Given an uncertain graph $\mathcal{G}$ and a motif M, BM [47] can output the accurate mean and variance of the motif count. But it's quite slow when the graph becomes larger. We use BM to provide the ground truth of mean and variance.

•**BMA** BMA [47] is the approximate (and faster) version of BM providing approximate variances. The mean computed by BMA are still accurate.

•**PGS** PGS is the basic sampling method discussed in Section 4. For querying motifs with 5 nodes or less, we use the state-of-the-art deterministic motif counting algorithm, Escape [42] as the kernel. When dealing with motifs with more than 5 nodes, we use VF3 [12], as Escape only supports up to 5 nodes. PGS has the same effectiveness as LINC; thus we only test its efficiency.

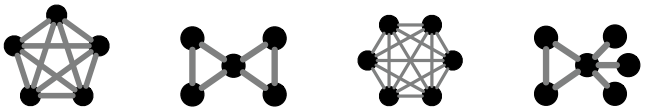
•**BS** BS has the same effectiveness as LINC; thus we only test its efficiency. We use BS as the comparison to show the effect of the techniques designed in LINC.

Among the above methods, PGS and BS support counting motifs based on both non-induced and induced subgraphs, but BM/BMA can only support counting based non-induced subgraphs [47].When comparing with BM/BMA, all methods follow the non-induced subgraph semantics. Unless stated otherwise, the experiments follow the induced subgraph semantics.

[7]http://konect.uni-koblenz.de/networks/douban

[8]http://konect.uni-koblenz.de/networks/roadNet-CA

We use all 3-node and 4-node motifs (listed in Table 1) and four selected 5-node and 6-node motifs (including dense and sparse ones, shown in Fig. 10) as representatives to conduct the experiments on effectiveness and efficiency.

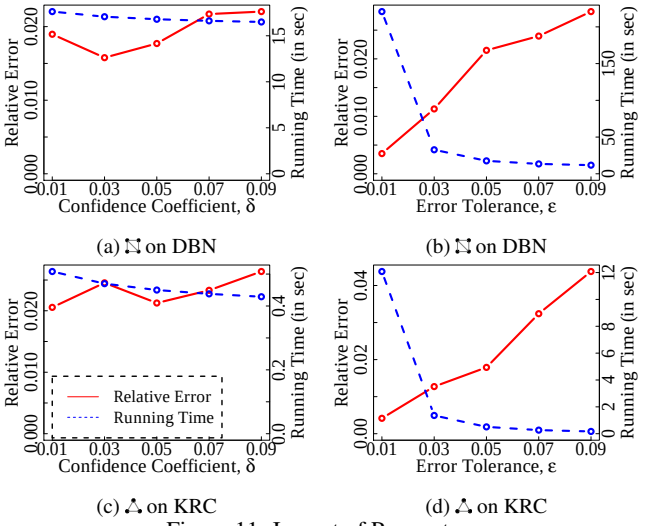


(a) $M_{5,1}$ (b) $M_{5,2}$ (c) $M_{6,1}$ (d) $M_{6,2}$

Figure 10: Motif Representatives

8.2 Impact of Parameters

Before comparing effectiveness and efficiency with competitors, we first investigate the influence of two user-defined parameters—error tolerance ε and confidence coefficient δ. We evaluate the parameters' impact on efficiency and effectiveness. We run the LINC-n algorithm 50 times for each parameter setting and compute the average relative errors of the estimated variances and the average running time to measure the effectiveness and efficiency, respectively. BM provides the ground-truth variances.



(a) ⋈ on DBN (b) ⋈ on DBN

(c) ⋔ on KRC (d) ⋔ on KRC

Figure 11: Impact of Parameters

First, we fix $\varepsilon = 0.05$ and increase δ by 0.02 each time starting from 0.01. The results are illustrated in Fig. 11a and Fig. 11c. The general trend is that the running time decreases and the relative error increases along with the growth of δ. What's more, the running time and the relative error do not change much as δ gets larger.

Then, we evaluate the impact of $\varepsilon \in \{0.01, 0.03, 0.05, 0.07, 0.09\}$, with fixing $\delta = 0.01$. Fig. 11b and Fig. 11d summarize the results on two datasets. Both figures show that the execution time decreases with increasing ε and the decreasing rate is less remarkable when ε becomes larger. Along with the increment of ε, the relative error increases almost linearly. Compared to the impact of δ, the influence of ε on efficiency and effectiveness is more noteworthy. This is reasonable, because the time complexity of the algorithm is proportional to $\frac{1}{\varepsilon^2}$ and $\ln\frac{1}{\delta}$ according to Section 4.1. From the experiments, we conclude that it is reasonable to choose $\varepsilon = 0.05$ and $\delta = 0.01$ as the default parameters to obtain relatively high-quality results without incurring excessive time in the following experiments.

8.3 Effectiveness

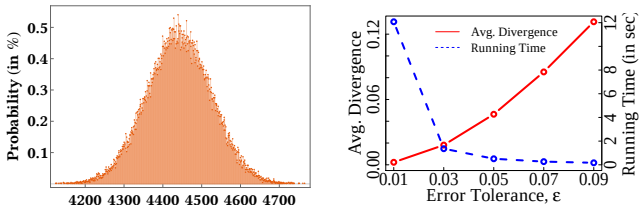
As we discussed in Section 3, it is infeasible to obtain the exact pmf on even moderately large graphs. Hence, to evaluate the approximation results, we have measured the average relative error of estimated means and variances based on our approximate pmfs results with respect to different motifs, where the accurate means and variances are calculated through BM [47]. The relative error is calculated via $|\frac{v - v_{\text{approx}}}{v}|$, where v is the ground truth value and v_{approx} is the corresponding approximation.

Table 4: Summary of Relative Error (in %) of Est. Variance

Algoritham	Mean	Standard Deviation
LINC-n	1.89	1.29
BMA	19.43	37.85

We omit the comparison between LINC-n and BMA regarding relative errors of the estimated mean of the motif counts here because all of them are within 1%.

Then, we evaluate the accuracy of estimated variances on different datasets for different motifs. Here, we compare LINC-n with BMA, using its default parameter in their code (the sampling ratio $q = 0.5$). The setting for LINC-n follows the default setting. We repeat the experiments ten times. Table 4 summarize the mean and the standard deviation of the average relative errors for these two algorithms. (We provide more details in our full version.) BM calculates the ground truth for this task. According to the table, we observe that LINC-n is more stable compared to BMA, whose relative errors fluctuate wildly. Among all these experiments, the relative errors of LINC-n are below 5%. By contrast, about 40% of the relative errors obtained by BMA are above 5%, and about 18% of those have relative error $\geq 50\%$.



(a) The pmf

(b) Impact of ε with $\delta = 0.01$

Figure 12: The pmf example of *triangle* count on KRC

Fig. 12a gives an example pmf of the *triangle* count on KRC. The pmf of *triangle* count on KRC (Fig. 12a) looks like the normal distribution. Among our experiments, we find that most pmfs are similar to the normal distribution and their shapes look like Fig. 12a. Here we set the parameters to error tolerance $\varepsilon = 0.01$ and confidence coefficient $\delta = 0.01$ to provide a high-quality approximate pmf. Fig. 12b shows the impact of ε on the approximate pmf quality with fixing $\delta = 0.01$. Here, we run the LINC algorithm 50 times over each parameter setting, calculate the average divergence between every two approximate pmfs and use the average divergence to measure the quality of the output, as it is #P-hard to obtain the accurate pmf (more details in our full version [1]). From Fig. 12b, we can observe that the parameter setting with $\varepsilon = 0.01, \delta = 0.01$ provides quite good results.

8.4 Efficiency

For assessing efficiency, we compare LINC-n to PGS-n, BS-n and BMA. Here all parameters are the same as in the effectiveness experiments. Again, we repeat the experiments 10 times.

Fig. 13 reports **the average running time of** LINC-n**,** PGS-n**,** BS-n **and** BMA. We omit the running time of BM here, which is

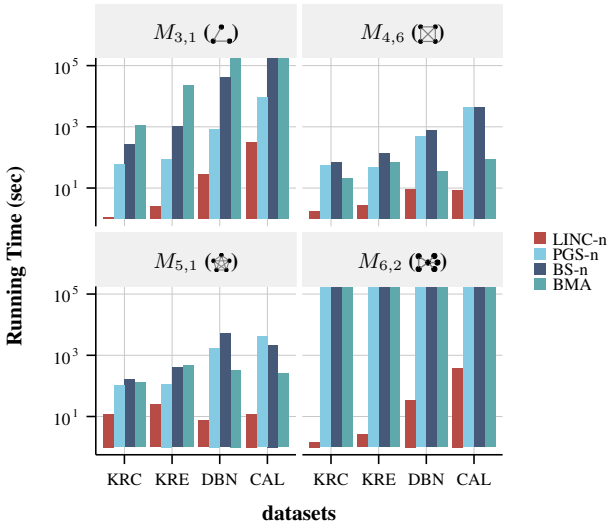


Figure 13: Query Efficiency Comparison

about 4× longer than BMA. Note that PGS/BS/BMA cannot finish in 30 hours on the some cases. As depicted in Fig. 13, we observe that LINC-n consistently outperforms PGS-n, BS and BMA over different queries and datasets, especially when querying $\triangle$, even around 1000× faster than BMA. Compared to PGS-n and BS-n, we observe that LINC-n provides more than 10× speedup in most cases. We provides more details on other eight motifs in the Appendix of our full version [1].

We present **the average memory usage of** LINC-n**,** PGS-n**,** BS-n **and** BMA in Fig. 14. For some processes which did not finish in 30 hours, we use the maximum memory already used. From Fig. 14, we can observe that LINC-n uses slightly more memory than PGS-n. They are better than BS-n and BMA. Combined with the results from the effectiveness experiment, we conclude that LINC-n strongly dominates BMA on both efficiency and effectiveness (w.r.t. variance estimation).

We have performed an experimental evaluation of the incremental update of LINC structures. Upon inserting an edge, the incremental update is up to 100 times faster than recomputing the data structures of LINC from scratch. (More details in our full version [1].) Hence, the incremental update can handle graph update operations effectively.

To have a better understanding of the memory overhead of LINC, we examine the number of bytes used to store the structures in LINC, divided by the number of bytes for storing the uncertain graph $\mathcal{G}$. We report this ratio averaged over the four datasets, in Table 5. We can see that it varies from 0.2 to 2.2, within a reasonable range.

Table 5: Overhead of LINC structures over $\mathcal{G}$.

motif	$M_{3,1}$	$M_{3,2}$	$M_{4,1}$	$M_{4,2}$	$M_{4,3}$	$M_{4,4}$
overhead	0.2	0.2	2.2	1.6	1.6	2.0
motif	$M_{4,5}$	$M_{4,6}$	$M_{5,1}$	$M_{5,2}$	$M_{6,1}$	$M_{6,2}$
overhead	1.3	0.4	0.9	1.6	1.8	0.2

8.5 Effects of the Optimization Techniques

Effects of the Early Stop Strategy. Here, we evaluate the effect of the early stop strategy discussed in Section 7. With the help

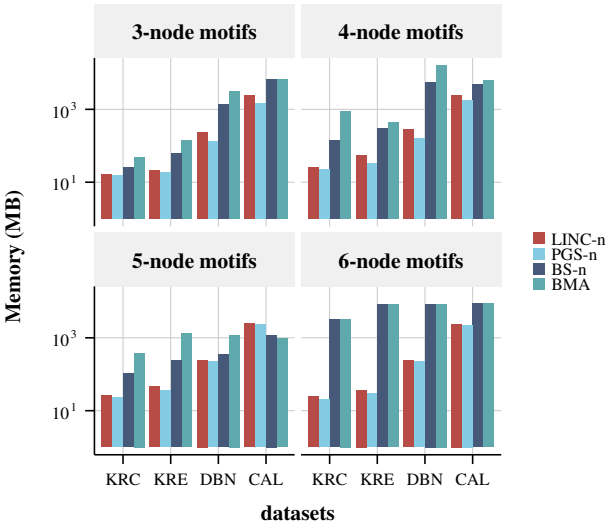


Figure 14: Memory Usage Comparison

Table 6: Early Stop Examples

Motifs	Dataset	$\bar{n}$	N	$\frac{\bar{n}}{N}$	Saved Time
triangle	KRC	1175	2829	42%	0.58s (48%)
5-clique	KRE	955	3101	31%	1.76s (7%)
4-cycle	DBN	1119	3673	30%	30s (48%)
3-star	CAL	1216	4966	24%	454s (74%)

of the early stop strategy, LINC-n could save about 68% samples on average while still keeping the relative error of the estimated variance less than 5%. We report several early stop examples in Table 6. N denotes the sample size determined by Eq. 7 and $\bar{n}$ denotes the average number of the actual samples used when applying the early stop strategy.

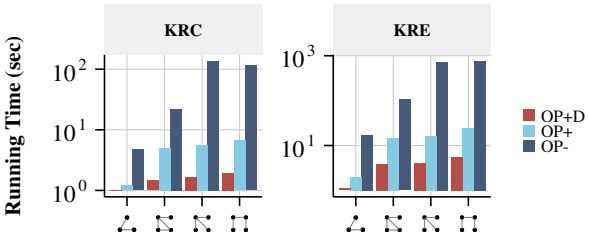


Figure 15: Effects of Optimization for Specific Motifs

Effects of the Optimization for Specific Motifs. (Section 7) Fig. 15 shows the optimization effects of querying *2-star*, *3-star*, *4-path* and *4-tailedtriangle* on KRC and KRE (The effects are similar on the other datasets). OP- denotes not using the optimization technique, OP+ denotes enabling the optimization technique (it is enabled in other experiments), and OP+D denotes enabling the optimization technique and early stop strategy. From Fig. 15, we observe that the optimization technique provides substantial improvements for those specific motifs.

8.6 Scalability

To evaluate the scalability of LINC, we query the motifs 10 times using the default parameters ($\varepsilon = 0.05$ and $\delta = 0.01$) over the synthetic datasets (Table 3), whose sizes are increased by $10\times$ each one compared to the previous one. We plot the average running time of each motif, respectively, over different datasets in Fig. 16. The figure shows that the running time of LINC grows approximately linearly with the increase of the graph size, which indicates good scalability of LINC.

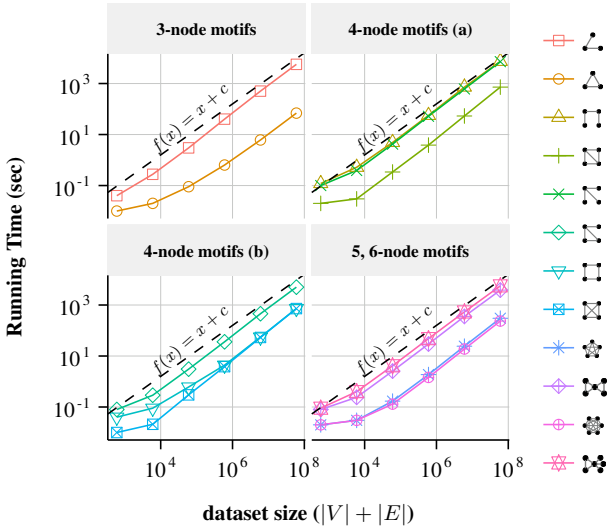


Figure 16: Scalability Evaluation

9. CONCLUSION

In this paper, we investigate the motif counting problem on uncertain graphs. Based on the Monte-Carlo sampling, we propose two approximate algorithms with probabilistic accuracy guarantees: PGS, which makes use of the current motif counting algorithms designed for deterministic graphs and LINC, which achieves further speedup by exploiting the common structures among different sample graphs. Finally, we evaluate the algorithms on both real and synthetic datasets of different sizes. The experimental results demonstrate that LINC is effective, efficient, and scalable for the motif counting queries on uncertain graphs. For future work, we plan to study how to extend our solution to support more complex uncertain graph models (e.g., Bayesian networks).

Acknowledgements

Reynold Cheng, Chenhao Ma, Tobias Grubenmann, and Xiaodong Li are supported by the Research Grants Council of Hong Kong (RGC GRF Projects HKU 17229116, 106150091, and 17205115), the University of Hong Kong (Projects 104004572, 102009508, and 104004129), the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18), and the RGC Germany / Hong Kong Joint Research Scheme (G-HKU706/18). Lakshmanans research was supported in part by a discovery grant and a discovery accelerator supplement grant from NSERC (Canada).

10. REFERENCES

[1] Linc: A motif counting algorithm for uncertain graphs [full version]. https://i.cs.hku.hk/~chma2/linc.pdf.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD*, pages 34–48, New York, NY, USA, 1987. ACM.

[3] E. Adar and C. Re. Managing uncertainty in social networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.

[4] K. Aggarwal, J. Gupta, and K. Misra. A simple method for reliability evaluation of a communication system. *IEEE Transactions on Communications*, 23(5):563–566, 1975.

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

[6] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 14(6):1170–1175, 2004.

[7] M. Y. Ata. Determining the optimal sample size in the monte carlo experiments. *Selcuk Journal of Applied Mathematics*, 7(2):103–111, 2006.

[8] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[9] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[10] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting uncertainty in graphs for identity obfuscation. *PVLDB*, 5(11):1376–1387, 2012.

[11] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Counting graphlets: Space vs time. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 557–566. ACM, 2017.

[12] V. Carletti, P. Foggia, A. Saggese, and M. Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

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

[14] Y. Chen, X. Zhao, X. Lin, and Y. Wang. Towards frequent subgraph mining on single large uncertain graphs. In *ICDM*, pages 41–50. IEEE, 2015.

[15] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4):523–544, 2007.

[16] M. Evans, N. Hastings, and B. Peacock. Statistical distributions. 2000.

[17] O. Frank. Estimating a graph from triad counts. *Journal of Statistical Computation and Simulation*, 9(1):31–46, 1979.

[18] O. Frank. Triad count statistics. In *Annals of Discrete Mathematics*, volume 38, pages 141–149. Elsevier, 1988.

[19] O. Frank and D. Strauss. Markov graphs. *Journal of the American Statistical Association*, 81(395):832–842, 1986.

[20] M. Fréchet. Généralisation du théoreme des probabilités totales. *Fundamenta Mathematicae*, 1(25):379–387, 1935.

[21] J. Ghosh, H. Q. Ngo, S. Yoon, and C. Qiao. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM*, pages 1721–1729. IEEE, 2007.

[22] M. Granovetter. The strength of weak ties: A network theory revisited. *Sociological Theory*, pages 201–233, 1983.

[23] J. A. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Annual International Conference on Research in Computational Molecular Biology*, pages 92–106. Springer, 2007.

[24] S. Gurukar, S. Ranu, and B. Ravindran. Commit: A scalable approach to mining communication motifs from dynamic networks. In *SIGMOD*, pages 475–489. ACM, 2015.

[25] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[26] J. Hu, R. Cheng, K. C. Chang, A. Sankar, Y. Fang, and B. Y. Lam. Discovering maximal motif cliques in large heterogeneous information networks. In *ICDE*. IEEE, Forthcoming 2019.

[27] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *SIGMOD*, pages 325–336. ACM, 2013.

[28] J. Huang, L. Antova, C. Koch, and D. Olteanu. Maybms: a probabilistic database management system. In *SIGMOD*, volume 9, pages 1071–1074, 2009.

[29] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322. ACM, 2014.

[30] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, pages 495–505. International World Wide Web Conferences Steering Committee, 2015.

[31] A. Kara, H. Q. Ngo, M. Nikolic, D. Olteanu, and H. Zhang. Counting triangles under updates in worst-case optimal time. In *ICDT*, pages 4:1–4:18, 2019.

[32] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637, 2006.

[33] J. Kunegis. Konect: the koblenz network collection. In *WWW*, pages 1343–1350. ACM, 2013.

[34] L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang. Scalable distributed subgraph enumeration. *PVLDB*, 10(3):217–228, 2016.

[35] L. V. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 22(3):419–469, 1997.

[36] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704. ACM, 2008.

[37] D. Marcus and Y. Shavitt. Rage–a rapid graphlet enumerator for large networks. *Computer Networks*, 56(2):810–819, 2012.

[38] T. Milenković, W. L. Ng, W. Hayes, and N. Pržulj. Optimal network alignment with graphlet degree vectors. *Cancer Informatics*, 9:121, 2010.

[39] 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.

[40] K. Mouratidis and M. L. Yiu. Anonymous query processing in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(1):2–15, 2010.

[41] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: new developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2014.

[42] A. Pinar, C. Seshadhri, and V. Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *WWW*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017.

[43] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest

neighbors in uncertain graphs. *PVLDB*, 3(1-2):997–1008, 2010.

[44] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[45] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[46] D. Szklarczyk, J. H. Morris, H. Cook, M. Kuhn, S. Wyder, M. Simonovic, A. Santos, N. T. Doncheva, A. Roth, P. Bork, et al. The string database in 2017: quality-controlled protein–protein association networks, made broadly accessible. *Nucleic Acids Research*, 45(D1):D362–D368, 2017.

[47] A. Todor, A. Dobra, and T. Kahveci. Counting motifs in probabilistic biological networks. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 116–125. ACM, 2015.

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

[49] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

[50] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*, volume 8. Cambridge University Press, 1994.

[51] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017.

[52] S.-H. Yook, Z. N. Oltvai, and A.-L. Barabási. Functional and topological characterization of protein interaction networks. *Proteomics*, 4(4):928–942, 2004.

[53] R. Zafarani and H. Liu. Social computing data repository at asu, 2009.