

MOSER: Scalable Network Motif Discovery using Serial Test

Mohammad Matin Najafi[†], Chenhao Ma[‡], Xiaodong Li[†], Reynold Cheng[†], Laks V.S. Lakshmanan[§]

[†]The University of Hong Kong, Hong Kong SAR, China

[‡]The Chinese University of Hong Kong, Shenzhen, China

[§]The University of British Columbia, Vancouver, Canada

{matin,xldi,ckcheng}@cs.hku.hk;machenhao@cuhk.edu.cn;laks@cs.ubc.ca

ABSTRACT

Given a graph G , a motif (e.g., 3-node clique) is a fundamental building block for G . Recently, motif-based graph analysis has attracted much attention due to its efficacy in tasks such as clustering, ranking, and link prediction. These tasks require Network Motif Discovery (NMD) at the early stage to identify the motifs of G . However, existing NMD solutions have two drawbacks: (1) Lack of theoretical guarantees on the quality of the samples generated using the existing methods, and (2) inefficient algorithms, which are not scalable for large graphs. These limitations hinder the exploration of motifs for analyzing large graphs. To address the above issues, we propose a novel solution named MOSER (MOTif Discovery using SERial Test). This novel NMD framework leverages a significance testing method known as the serial test, which differs from the existing solutions. We further propose two fast incremental subgraph counting algorithms, allowing MOSER to scale to larger graphs than ever possible before. Extensive experimental results show that using MOSER can improve the state-of-the-art up by to 5 orders of magnitude in efficiency and that the motifs found by MOSER facilitate downstream tasks such as link prediction.

PVLDB Reference Format:

Mohammad Matin Najafi[†], Chenhao Ma[‡], Xiaodong Li[†], Reynold Cheng[†], Laks V.S. Lakshmanan[§], MOSER: Scalable Network Motif Discovery using Serial Test. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Given a graph G , a motif m is a small graph of a few nodes, whose occurrence in G is significantly higher than expected [27, 34]. A motif is a fundamental building block of a graph, and it facilitates the understanding of the non-random, structural, or evolutionary design principles used to construct the graph [44]. For example, a Feed-Forward Loop (FFL) motif (e.g., Fig. 1c) is used to study the regulatory control mechanism in gene transcriptional networks [27]; an undirected triangle motif \triangle supports the study of relationships in social and epidemiological contact networks [44]. In recent years, researchers have effectively employed motifs in “high-order” graph

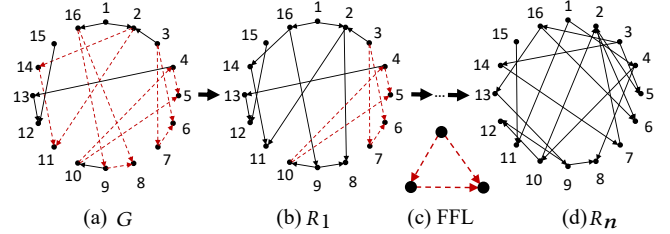


Figure 1: (a) A graph G ; (b) random graph R_1 after switching (2, 14) and (16, 8) on G ; (c) Feed-Forward-Loop (FFL); and (d) uniform graph R_n after n switching operations. The red dashed lines denote the edges that take part in the FFL motif, which is observed four times in the actual network.

analytics [22], including clustering [6, 21], link prediction [21], conductance [46], and community search [14].

Several different types of motifs such as FFL, stars, and triangles have been studied in the literature. Much of the prior work (e.g., [2, 13, 32, 36]) focuses on motif counting, i.e., given a graph, count the occurrences of *given* motif types. Not every motif type (i.e., graph pattern) occurs, or occurs sufficiently often in every graph. Thus, given a graph G , we first need to *discover* what are the underlying motifs that form a basis for constructing G , so that they can be leveraged for the “high-order” analytics tasks above. In the literature, Motif Discovery (MD) algorithms [6, 21, 22, 46]) have been developed for this purpose. A motif is a k -node subgraph g , whose frequency in G is higher than expected. In the literature, k is usually small (e.g., $k = 3, 4$, or 5). Fig. 1(a) shows a graph G . The frequency of the FFL motif in Fig. 1(c) in G is 4, the four instances being (9, 8, 16), (2, 11, 14), (3, 6, 7), and (4, 5, 10). Edges contributing to a FFL instance are colored red.

Existing MD algorithms are computationally expensive. Given a graph pattern g (e.g., Fig. 1(c)), a *statistical significance test* has to be performed on g . Specifically, g is considered to be a motif, if the frequency of g in G is higher than the expected frequency of g in some number of random graphs derived from G [27]. This test is very time consuming, because generating a uniform random sample is expensive. In current MD solutions, a random graph is often generated by applying *switching* operations repeatedly on the graph of interest. Fig. 1(b) shows a random graph R_1 , generated by performing a switching operation on G , where two edges are randomly selected from G (e.g., (2, 14) and (16, 8)). These are replaced by two new edges with the destination nodes of the selected edges swapped (i.e., (2, 8), (16, 14)). Notice that the random graph created in this way is *degree-equivalent* to G – i.e., the in- and out-degrees of the nodes of the random graph are identical

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. 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. 14, No. 1 ISSN 2150-8097.

doi:XX.XX/XXX.XX

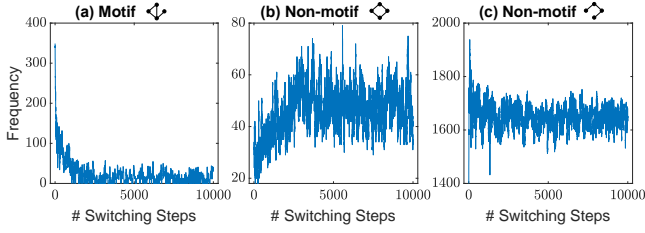


Figure 2: The frequency of motif and non-motifs over the no. of switching operations on *Social* dataset.

to those of G . In order for MD to be accurate, a huge number of switching operations have to be applied to G , so that an unbiased, or *uniform*, random sample can be generated (e.g., Fig. 1(d)). Previous approaches require the number of switching operations on G to be a multiple of the number of edges in G , making the number of switching operations prohibitive for large graphs with millions of edges. The problem is aggravated by the fact that a number (e.g., 1000 in the literature) of uniform samples have to be generated for carrying out the significance test.

• **Problem 2. Performing subgraph counting on a large graph is expensive.** In MD solutions, one of the main bottlenecks is to obtain the frequency of a given graph pattern g in graph G and its associated random samples. While several fast counting solutions (e.g., ESCAPE [32]) have been developed, they do not scale to large graphs. For example, it takes a few hours just to obtain the frequency of 5-node graph patterns in a million-node scale graph in our experiments.

To summarize, existing MD solutions cannot scale to large graphs, because generating a uniform random sample is expensive (Problem 1), and subgraph counting is expensive (Problem 2) for these graphs. As a reference, in our experiments on the *web-google* network with 1M nodes and 4.5M edges, 5 minutes are needed to draw a random sample uniformly. Using $n = 1000$ random graphs, the entire MD process for discovering 5-node motifs takes around 4 days with the use of the state-of-the-art subgraph counting algorithm ESCAPE [32].

Motif frequency and switching. To address the challenge of developing a *scalable* and *accurate* MD solution, we leverage the observation that when a switching operation is repeatedly performed on G , the frequency of a motif of G drops sharply unlike that of a non-motif. For example, the tailed-triangle ∇ is known to be a motif for the *Social* graph (details in Section 6, Table 2, row 2). In Fig. 2(a), we can see that its frequency on the random graph derived from *Social* drops very quickly with the number of switching operations. For a graph pattern g to qualify to be a motif of a given graph G , g has to be frequent in G . Hence, an edge e selected for switching has a high chance to participate in one or more instances of g . When e is removed by switching, its associated instances of g may also be “destroyed”. In Fig. 1(a), for example, G has 4 instances of FFL. After a switch is performed (Fig. 1(b)), the random graph R_1 has 2 instances; the number of FFL instances drop to zero after n switches are performed (Fig. 1(c)). As illustrated in Fig. 2(b)-(c), we do not observe this phenomenon for the two non-motifs.

MOSER: A novel MD framework. Based on the above intuition, we propose our algorithm MOSER (or MOTif discovery using SERIAL test), which facilitates scalable and accurate motif discovery. The main idea is to view a sequence of switching operations on graph G as a Monte Carlo Markov Chain (MCMC), and apply Serial Test, an advanced significance testing method, to quickly decide whether graph pattern g is a motif of G . We show that MOSER substantially reduces the sample complexity of the MD problem while achieving results equivalent to the state-of-the-art MD solution. An additional advantage of MOSER is that it supports *any* subgraph counting algorithm (for finding frequency of g in a graph). To further enhance the efficiency of MOSER, we develop two algorithms: (1) Track And Count (TAC) which computes the frequency of g on a random graph r' quickly from another random graph r , from which r' is derived by; and (2) Accelerated Track And Count (ATAC), a solution particularly optimized for simple graphs and for up to 5-node motifs.

We have performed extensive experiments on a variety of graphs, and found that MOSER performs much better (*up to 5 orders of magnitude*) over existing solutions. We have also performed a case study on a motif-based link prediction algorithm [1]. It shows that the motifs produced by MOSER are effective for downstream link prediction.

We make the following contributions:

- (1) We propose MOSER, a scalable framework for motif discovery from large graphs;
- (2) We prove that MOSER is equivalent to existing MD solutions, i.e., it discovers the same motifs;
- (3) We develop two fast and scalable incremental subgraph counting algorithms (TAC and ATAC); and
- (4) We perform extensive experiments, which show that MOSER is more efficient than existing solutions.

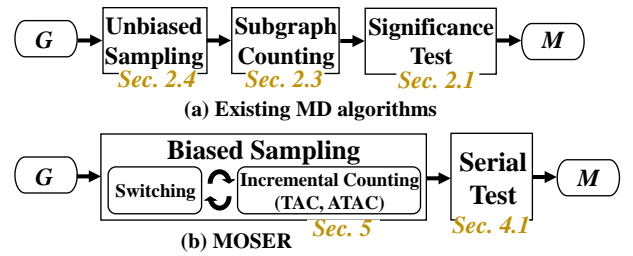


Figure 3: Roadmap of our paper (G : graph; M : motifs)

Paper Organization. The rest of the paper is organized as follows. Section 2 discusses the preliminaries. In Section 3 we study the relationship between the switching method and MCMC. In Section 4 we propose the MOSER framework. In Section 5 we present TAC and ATAC. Section 6 presents our experimental results. Section 7 covers the related work. Section 8 concludes. Fig. 3 shows the roadmap of the paper.

2 BACKGROUND

In this section, we discuss the problem of Motif Discovery in detail. We start with some basic notations and follow up with a description of the classic framework for motif discovery.

2.1 Graph Notation

Let $G = (V, E)$ be a graph with vertex set V and edge set E . When the edges are directed (resp. undirected), we refer to it as a directed (resp. undirected) graph. A graph is *simple* if there are no self-loops nor parallel edges. In this work, we only consider simple graphs.

Definition 2.1 (Subgraph and Induced Subgraph). A subgraph $g = (V_g, E_g)$ of a graph G is a graph such that $V_g \subseteq V$ and $E_g \subseteq E$. It is said to be an *induced* subgraph if $E = V_g \times V_g \cap E$, i.e., g contains all edges of G whose endpoints are in V_g . For brevity, we refer to induced subgraphs as *subgraphs*.

Definition 2.2 (Graph Isomorphism). Two graphs G and H are said to be isomorphic, denoted as $G \sim H$, if there exists a bijection between the vertices of both graphs such that two vertices of G share an edge if and only if their corresponding vertices in H also share an edge. The best known algorithm for testing if two graphs are isomorphic runs in pseudo-polynomial time [12].

2.2 Motif Discovery

As defined by Milo et al. [27], motifs are patterns of interconnections occurring in complex networks in numbers that are significantly higher than those in degree-equivalent randomized networks.

Definition 2.3 (Degree-equivalent Graphs). A graph $G' = (V', E')$ is degree-equivalent to $G = (V, E)$ if and only if $|V| = |V'|$ and $|E| = |E'|$ and there exists a bijection $\psi : V \rightarrow V'$, such that for any node $v \in V$, $d_{in}(v) = d_{in}(\psi(v))$ (in-degree), and $d_{out}(v) = d_{out}(\psi(v))$ (out-degree).

Let Λ be the set of all graphs degree-equivalent to G , and let g be a k -node subgraph pattern. We define $F_g(G)$ to be the frequency of all the subgraph patterns isomorphic to g in G . We define the *Motif Discovery* problem formally as:

Definition 2.4 (Motifs). Based on the definitions in the previous works [27, 48], we define a *motif* as an induced k -node subgraph g of G such that it satisfies the following conditions:

$$F_g(G) \geq u, \quad (1)$$

$$\text{Prob}(F_g(R) > F_g(G)) \leq p, \quad (2)$$

where u is a *minimum frequency* threshold that ensures g is frequent in G , R is a randomized graph degree-equivalent to G , and p is the *p-value*, that ensures that the $F_g(G)$ is statistically significant.

Let \mathcal{R} be a randomly selected subset of Λ with size n , the LHS of the Eq. 2 can be calculated as:

$$\text{Prob}(F_g(R) > F_g(G)) = \frac{1}{n} \sum_{R \in \mathcal{R}} \delta[F_g(R) > F_g(G)] \quad (3)$$

Here, $\delta[F_g(R) > F_g(G)]$ is an indicator function defined as:

$$\delta[F_g(R) > F_g(G)] = \begin{cases} 1 & F_g(R) > F_g(G) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The parameters $\{p, u, n\}$ are user-defined and Milo et al. [27] have suggested $\{p = 0.01, u = 4, n = 1000\}$. We refer to the above as the *BaseTest* for motif discovery. Later, in some works [17, 19, 23], another significance metric known as the *z-score*, was also used:

$$z = \frac{F_g(G) - \bar{F}_g(R)}{\sigma(F_g(R))}, \quad (5)$$

where $\bar{F}_g(R)$ refers to the mean frequency of g for every R in Λ , and $\sigma(F_g(R))$ refers to the standard deviation of the frequency of g in the random graphs. Note that these two criteria (resp. based on z and p) can be converted to each other if the underlying distribution of the frequencies is known. So, in this paper we focus on the *p-value* test (Eq. 2).

Let S_k be the set of all k -node non-isomorphic subgraph patterns that are present in G . We define the motif set of G as the set of subgraphs $\mathcal{M} = \{g \in S_k \mid g \text{ passes the BaseTest}\}$. To calculate \mathcal{M} , we need (1) to generate a set of n random graphs \mathcal{R} , each of which is degree-equivalent to G , and (2) to find the frequencies F_g of $g \in S_k$. We refer to (1) as *Random Graph Generation* and to (2) as *Subgraph Counting*. Algorithm 1 describes the baseline motif discovery procedure in detail.

Algorithm 1: Baseline Motif Discovery [17, 19, 24, 27, 37]

```

Input:  $G, k, n, p, u$ 
1  $\mathcal{M} \leftarrow \emptyset$  // Motif set
2  $C_0 \leftarrow \mathcal{F}_k(G)$ 
3  $\mathcal{R} \leftarrow R_1 \dots R_n$  // Random graphs
4 forall  $R_i \in \mathcal{R}$  do
5    $C_i \leftarrow \mathcal{F}_k(R_i)$ 
6 forall  $g \in S_k$  do // subgraphs of size  $k$  in  $G$ 
7   if  $\text{BaseTest}(g, C, p, u)$  then // assess significance of  $g$ 
8      $\mathcal{M} \leftarrow \mathcal{M} \cup \{g\}$ 
9 return  $\mathcal{M}$ 

```

Next, we discuss the *Subgraph Counting* and *Random Graph Generation* problems in depth.

2.3 Subgraph Counting

Given a graph G and a parameter k , the goal is to count the k -node subgraphs g of G . Subgraphs that are different in at least one node are considered distinct. Let S_k be the set of all k -node non-isomorphic subgraphs g that are present in G . Now we formally define the *subgraph counting* problem:

$$\mathcal{F}_k(G) = \{(g, F_g(G)) \mid g \in S_k\} \quad (6)$$

There are two main challenges in the subgraph counting problem. First, the number of instances of a subgraph g increases exponentially w.r.t. the size of G and k . Second, for every subgraph instance $g \subseteq G$, it is necessary to find the isomorphic class of g and increment the respective class frequency. As the number of subgraph instances grows exponentially with the size of G and with k , assigning every instance g to its class is the *main bottleneck* for the subgraph counting problem. For example, in the *web-google* dataset

there are $\sim 6 \times 10^{12}$ instances of undirected 4-node subgraphs and even assuming that visiting and classifying each instance takes a microsecond, the enumeration itself would take more than 40 days. This is the reason why most of the MD algorithms have based their solutions on reducing the number of isomorphism checks.

The next step of the MD is to generate an ensemble of random graph samples to apply significance testing. The details of the random graph generation problem are discussed next.

2.4 Random Graph Generation

The goal is to generate $R = (V, E')$, a random graph sample that is degree-equivalent to $G = (V, E)$, and is uniformly drawn from Λ . In other words, the probability of drawing a graph from Λ at random should be exactly $\frac{1}{|\Lambda|}$.

As suggested by Milo et al.[26], the most practical¹ random graph generation algorithm to maintain the degree-equivalence is the *Switching Method*. This is the most widely accepted and used method in the MD literature. To understand the switching method, we first define a *Single Switch*.

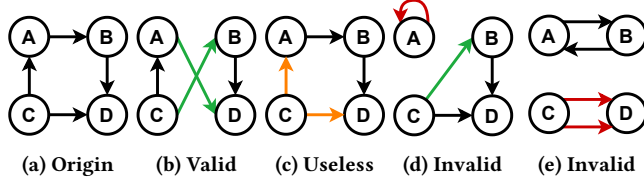


Figure 4: Single Switch from (a) an original graph to random graphs that (b) swap (a, b) and (c, d), (c) swap (c, a) and (c, d), (d) swap (c, a) and (a, b), and (e) swap (c, a) and (b, d).

Definition 2.5 (Single Switch). Let $\xi : \Lambda \rightarrow \Lambda$ be a function that given an input graph $G = (V, E)$ draws two random edges $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$ from E where $u_1 \neq v_2$, $u_2 \neq v_1$ (to avoid self-loops), and $e'_1 = (u_1, v_2) \notin E$, $e'_2 = (u_2, v_1) \notin E$ (to avoid parallel edges), and outputs $R = (V, E')$ where $E' = (E \setminus \{e_1, e_2\}) \cup \{e'_1, e'_2\}$. This ensures G and R are degree-equivalent. Fig. 4 illustrates different variations of a single switch.

Definition 2.6 (Switching Method). The procedure of starting with a graph G and performing τ single switches on G , resulting in a random graph sample R from Λ is called the *Switching Method*. Parameter τ must be chosen large enough to guarantee that R is a uniformly drawn sample.

Some questions arise governing the use of the switching method. (1) Is the switching method capable of generating uniform samples with a large enough τ ? (2) If so, how large should the τ be? In the MD literature [26], these questions have been only discussed *empirically*, and only on small graphs. It is empirically shown that the switching method is able to generate uniform samples when $\tau \geq 100 \times |E|$. In later works, due to the high computational complexity of the previous bound, a much smaller $\tau = 3 \times |V|$ is used [17, 19]. *Both of these bounds are ad hoc and do not come with any formal guarantees.*

¹By practical we mean, the algorithm strikes a balance between efficiency and accuracy.

2.5 Complexity

The MD algorithm consists of two main parts: (1) Generating n random graph samples R , (2) subgraph counting on the original graph G and on the n random graphs. The run time complexity of the classic MD algorithm (Algorithm 1) is

$$O \left(\overbrace{n \cdot (\tau \cdot T(\xi(G)))}^{\text{Sampling}} + \overbrace{T(\mathcal{F}_k(G)) + \sum_{R_i \in R} T(\mathcal{F}_k(R_i))}^{\text{Counting subgraphs on } G \text{ and } R} \right) \quad (7)$$

where:

- n : Number of random graph samples.
- $T(\xi(\cdot))$: Time complexity of performing a single switch on a given graph.
- τ : The minimum number of switches to generate a uniform random graph sample.
- $T(\mathcal{F}_k(\cdot))$: Time complexity of counting all the k -node subgraphs on a given graph.

As shown above, there are three main parameters that affect the time complexity of MD. We've discussed the effect of τ and \mathcal{F}_k before. The final essential parameter is n , the number of random graph samples. Milo et al. [26] suggested $n = 1000$ is sufficient in most cases. For further reference, the notations and symbols used in this paper are summarized in Table 1.

Table 1: The symbols and notations table.

Symb.	Definition	Symb.	Definition
$G(V, E)$	Graph (Vertices, Edges)	S	Set of all k -node subgraphs in G
\mathcal{M}	Set of all k -node motifs in G	τ	Number of switches for sampling
p	Significance level of motifs	u	Minimum frequency of a motif
$\mathcal{F}_k(G)$	Freq. of all k -node subgraphs in G	R_i	SSN's i -th random graph
n	Number of Random Graphs	$\xi(G)$	The <i>Single Switch</i> function
$d(i)$	Degree of node i	$W(G)$	Wedge count in G
$T(G)$	Triangle counts in graph G	$t(e)$	Triangle counts around edge e
$t(i)$	Triangle counts around node i	$DD(G)$	Directed Diamond

To summarize, the *key questions* governing the use of the switching method are:

- (1) Is the switching method capable of generating uniform samples given a large enough τ ?
- (2) If yes, what is the minimum number of switches for uniform sampling (bound on τ)?
- (3) Given τ , is it computationally feasible to generate samples using the switching method on *large* graphs?

The answers to the above questions are critical to understanding the accuracy of the MD algorithms, and to solving the scalability issue of MD algorithms. We address these questions in the next section. To the best of our knowledge, this is the first time this analysis is done in the MD literature.

3 THE SWITCHING METHOD REVISITED

In this section, we revisit the switching method as a *Markov chains* problem and answer the key questions raised in the previous section. An important observation is that the random graph sampling via the switching method has the Markov property. By definition, every

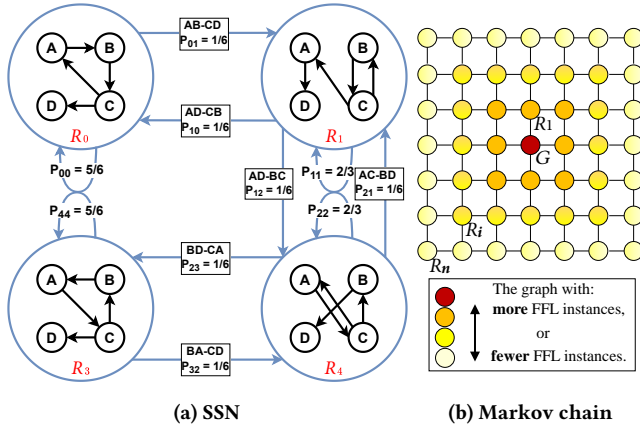


Figure 5: A toy example of (a) SSN and (b) Markov chain.

single switch only depends on the current state of the graph: given the current state, it is independent of the previous states.

Let M be a Markov chain with the stationary distribution $\vec{\pi}$ created using the switching method on the original graph G . Every state $R_i \in M$ is a random graph degree-equivalent to G . To theoretically show that this Markov chain is suitable for drawing uniform samples from Λ , we need to prove it has the following properties:

- (1) M converges to its stationary distribution $\vec{\pi}$ after a long enough random walk (Irreducibility and Aperiodicity [4]).
- (2) $\vec{\pi}$ is a uniform distribution on the state space Λ , i.e.,

$$\pi_i = \frac{1}{|\Lambda|}, \forall \pi_i \in \vec{\pi}$$

Proving the above conditions is equivalent to answering the *key question* (1). We refer to the Markov chain created using the switching method as **Switched State Network (SSN)**.

3.1 Switched State Network (SSN)

Let $P_{|\Lambda| \times |\Lambda|}$ be the transition matrix of M on Λ . Every state R_i of the chain, represents a random graph degree-equivalent to $R_0 := G$. Every transition between the states happens with a single switch operation, i.e., state R_i transits to R_j if and only if there exist edges e_1, e_2 in R_i such that switching them transforms R_i to R_j . Let Ω_i be the set of all the possible single switches on state $R_i = (V_i, E_i)$. Formally, we define Ω_i as:

$$\Omega_i = \{(e_1, e_2) \mid (e_1, e_2) \in E_i \times E_i \wedge e_1 \neq e_2\}$$

Note that $|\Omega_i| = \binom{|E_i|}{2}$, and the number of edges will not change after a switch, so the number of edges $|E_i| = |E|$ for every i . Now we define $\Omega_i^{val} \subset \Omega_i$ as the set of all the *valid switch pairs* and $\Omega_i^{inv} = \Omega_i - \Omega_i^{val}$ as the set of all *invalid switch pairs*. The *valid switch pairs* are the pairs of edges that if switched, do not result in parallel edges nor self-loops in R_i (cf. Fig. 4b). We include useless switches (cf. Fig. 4c) also in Ω_i^{inv} .

Using the above sets, we define the transition matrix P as:

$$P_{ij} = \begin{cases} \frac{1}{|\Omega_i|} & \text{if } i \neq j \text{ and a valid switch exists between states } i, j \\ 1 - \frac{|\Omega_i^{val}|}{|\Omega_i|} & \text{if } i=j \\ 0 & \text{if } i \neq j \text{ and no valid switch exists between states } i, j \end{cases}$$

We can operationalize the transition rules as follows:

- (1) From the current state $R_i = (V_i, E_i)$, calculate Ω_i and Ω_i^{val} sets.
- (2) From Ω_i , choose one pair (e_1, e_2) uniformly at random.
- (3) If $(e_1, e_2) \in \Omega_i^{val}$ apply switch on R_i and transit to R_j , stay in R_i otherwise.

Example 3.1 (Toy Example). Fig. 5a illustrates a toy SSN example. State R_0 represents the original graph. $\Omega_0^{val} = \{(e_1 = (A, B), e_2 = (C, D))\}$ contains only one *valid switch pair*. There are 4 edges in R_0 , so $|\Omega_0| = \binom{4}{2} = 6$. So, $P_{01} = \frac{1}{|\Omega_0|} = \frac{1}{6}$ and $P_{00} = 1 - \frac{1}{6} = \frac{5}{6}$. The entire transition matrix is illustrated in Fig. 5a. \square

Now that we have modeled the *Switching Method* using the SSN, we show that the SSN converges to its stationary distribution after a sufficiently long random walk. This requires showing that the SSN is irreducible and aperiodic[4].

PROPERTY 3.1. Irreducibility: *The SSN is irreducible, because the chain consists of one connected component by definition.*

PROPERTY 3.2. Aperiodicity: *Periodicity is a class property. Two states i and j are in the same communication class if and only if i is reachable from j and j is reachable from i . If state i of a Markov chain is periodic with period ω (resp. aperiodic), then all the states in the same communication class as i are periodic with period ω (resp. aperiodic). In SSN, all the transitions are bidirectional (i.e., $P_{ij} = P_{ji}$). So all the states of the SSN are in the same communication class. So, if any state i in SSN is aperiodic, the whole chain will be aperiodic. It is known that if a state has a self-loop (i.e., $P_{ii} > 0$), it is aperiodic[4]. So, to show SSN is aperiodic it is sufficient to show there exists a state in SSN that has a self-loop. According to the definition of P , probability of self-loops on the chain is $1 - \frac{|\Omega_i^{val}|}{|\Omega_i|}$. So, it is enough to show $|\Omega_i^{val}| < |\Omega_i|$ which is true if there exists a node u in G such that $\deg(u) \geq 2$, because there are always edges such as (a, u) and (u, b) (or (u, a) and (u, b)) in $|\Omega_i|$ that cannot exist in $|\Omega_i^{val}|$ because their switch will end in a self-loop (or a switch that does not change the state) in G_i and it's an invalid switch. The invalid switch will cause self-loops on the state i of the SSN.*

Given the properties above, it follows that the SSN will converge to a stationary distribution[4]. We denote the stationary distribution by $\vec{\pi}$.

As shown before, the transition probability between every two states R_i and R_j is either $\frac{1}{|\Omega_i|}$ if these two states are a *single valid switch* away, or 0 otherwise. We know that if state R_i is a *single valid switch* away from R_j , the reverse is also true due to the fact that every *single switch* is reversible. So $P_{ij} = P_{ji}$ holds for every R_i and R_j in Λ . We now prove that $\vec{\pi}$ is uniform, meaning that after convergence, it is equally likely to be in any state i of the chain.

LEMMA 3.2. *SSN has a unique and uniform stationary distribution.*

PROOF. Let $|\Lambda|$ be the number of states in Λ . We first check that $\forall i : \pi_i = \frac{1}{|\Lambda|}$ satisfies $\vec{\pi} \cdot P = \vec{\pi}$. Write:

$$(\vec{\pi} \cdot P)_i = \sum_j \pi_j \cdot P_{ji} = \frac{1}{|\Lambda|} \sum_j P_{ji} = \frac{1}{|\Lambda|} \cdot 1 = \pi_i$$

This shows that the uniform distribution satisfies the stationary property on the SSN. Next, we show that $\vec{\pi}$ is the unique stationary distribution of the chain. According to the Perron-Frobenius theorem [31], an irreducible and aperiodic Markov chain has a unique stationary distribution. The Perron-Frobenius theorem states that if a square matrix has a positive dominant eigenvalue, then the matrix has a unique stationary distribution. A Markov chain with an irreducible and aperiodic transition matrix has a dominant eigenvalue, and therefore has a unique stationary distribution. \square

We have rigorously shown that the SSN converges to a uniform stationary distribution, which means after a long enough random walk, the SSN generates uniform samples from Λ . This answers key question (1). We address the key questions (2) and (3) next.

3.2 Mixing Time of the SSN

Taking τ switches to generate a random graph sample in the switching method is equivalent to taking a random walk of τ steps on the SSN and report the resulting state as our random graph sample. The minimum number of steps that a random walk should take before converging to the chain's stationary distribution is called the *mixing time* denoted by τ_{mix} . So, the switching method needs to apply $\tau \geq \tau_{mix}$ switches to G to draw a uniform sample from Λ . To our best knowledge, there are no tractable theoretical bounds in the literature on the τ_{mix} for the switching method. For example, the mixing time estimate in [16] take the form of upper bounds and of $O(|E|^6)$. It is clear that this bound is intractable even for a graph with more than a thousand edges.

Mixing Time Bounds: Milo et al. in [26] has empirically shown that $\tau = 100 \times |E|$ is sufficient in practice for small graphs (1K nodes). In later works [17, 19], $\tau = 3 \times |V|$ is commonly used, as using $\tau = 100 \times |E|$ is computationally expensive. In a later work, Ray et al. [35] showed that the mixing time is linear to the number of edges in the original graph, i.e. $\tau_{mix} \approx c \cdot O(|E|)$. They also show empirically that $5 \leq c \leq 30$ is sufficient in most cases.

Infeasibility of Uniform Sampling: As an example, when analyzing the *Web-Google* dataset, which features approximately 876K nodes and 4.3M edges, even if we use $\tau = 3 \times |E|$ (which is smaller than the bounds provided in [35]) as a valid means of ensuring mixing for large graphs, it remains computationally infeasible to execute. Our experiments indicate that generating one unbiased sample on the *com-youtube* dataset takes around 4 minutes, which means generating $n = 1000$ samples would require around 3 days.

In summary, we have demonstrated that the switching algorithm can draw uniform samples from Λ if and only if sufficient (τ_{mix}) switching steps (key questions (1) and (2)) are taken. We have established that generating uniform samples for large graphs is not computationally feasible, even when utilizing empirical bounds (key question (3)). In the next section, we efficiently solve the problem of MD using *non-uniform* (biased) sampling without sacrificing quality.

4 ASSESSING SIGNIFICANCE IN A MARKOV CHAIN WITHOUT MIXING

The significance testing method used by the baseline MD solutions, highly relies on unbiased samples to estimate the underlying distribution of Λ , in terms of subgraph frequencies. In this section, we show that significance testing is possible on *Reversible* Markov chains, without requiring the chain to mix. Our approach requires the same number of samples n as the traditional methods, but drawing each sample has much lower complexity.

Motifs by definition, are subgraph patterns that are statistically over-represented, meaning that if subgraph pattern g is a motif in G , then $F_g(G)$ is relatively large. Intuitively, if we uniformly pick two random edges e_1 and e_2 from G , there will be a high probability that either e_1 or e_2 appear in one or more subgraphs of type g . This means that if we switch e_1 and e_2 , we tend to break the structure of the instances of g and eventually transforming it to other subgraph patterns. So, we can say every single switch on G tends to decrease $F_g(G)$, if g is a motif of G . Also based on the fact that no reversible Markov chain can have too many local outliers, we will use a new statistical significance test on Markov chains without requiring the chain to mix named as the *Serial Test*. As an intuitive explanation, Fig. 5b shows a small part of a Markov chain with a simple structure. Each state in this region has four neighboring states to which it can transition with equal probability. The colors in this figure indicate frequencies of a motif g (e.g. FFL) assigned to each state, with light yellow being the smallest and red being the largest. It is not possible to determine from this local region alone whether the red state highlighted in the figure has an unusually large frequency w.r.t. g compared to the rest of the chain. However, this state is considered a local outlier because it stands out from its neighboring states to a significant degree.

4.1 Serial Test

Our goal is to test the frequency significance of the subgraph g in state R_0 against n other random graphs sampled from the SSN's stationary distribution. In other words, MD would like to show that $R_0 = G$ is an unusual state in terms of $F_g(R_0)$ for states drawn from the chain's uniform stationary distribution $\vec{\pi}$. Note that this is equivalent to the Def.2.4, due to the fact that, MD is aiming to assign a p -value to every k -node subgraph g in G by comparing the $F_g(G)$ against an ensemble of random graphs uniformly drawn from Λ .

Definition 4.1 (Motifs - SSN version). An induced subgraph g of G is a motif if state R_0 of the SSN is a significant state with respect to the labeling function F_g under the null hypothesis that R_0 was chosen from the SSN's stationary distribution.

Based on this, solving the MD is equivalent to solving the problem of significance testing on Markov chains. The baseline solutions require unbiased sampling i.e. Markov chain mixing, which we have shown infeasible for large graphs. Here we introduce a statistical test known as the *Serial Test* that was first introduced by Besag and Clifford in [7] and then further refined by Chikina et al. in [9]. The serial test can assess the significance in a Markov chain *without* mixing, given that the chain is reversible.

PROPERTY 4.1. **Reversibility** SSN is reversible because it supports the detailed balance property which is $\pi_i P_{ij} = \pi_j P_{ji}$.

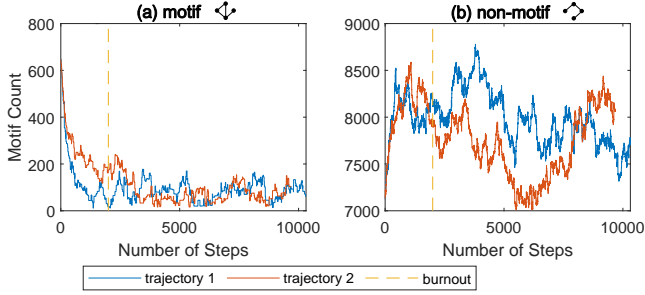


Figure 6: Frequency trends of subgraphs in E.coli dataset on iterative switching with double trajectory on (a) tailed-triangle motif and (b) 3-path non-motif.

THEOREM 4.2. (**Serial Test**) Fix any number t and assume that R_0 was chosen from the chain’s stationary distribution π , and that t' is chosen uniformly from $\{0, 1, \dots, t\}$. Consider two independent trajectories $Y_0, Y_1, \dots, Y_{t'}$ and $Z_0, Z_1, \dots, Z_{t-t'}$ in the reversible Markov chain M (whose states have real-valued labels) from $Y_0 = Z_0 = R_0$. If we choose R_0 from a stationary distribution π of M , then for any t we have that:

$$\text{Prob}(F_g(R_0) \text{ is an } \varepsilon\text{-outlier among } F_g(R_0), F_g(Y_1), \dots, F_g(Y_{t'}), F_g(Z_1), \dots, F_g(Z_{t-t'})) \geq \varepsilon) \quad (8)$$

To define the ε -outlier, we say that a real number α_0 is an “ ε -outlier” among the numbers $\alpha_0, \dots, \alpha_t$ if there are no more than $\varepsilon(t+1)$ indices for which $\alpha_i \leq \alpha_0$.

Theorem 4.2 establishes that when sampling from a reversible Markov chain to assess the significance of a state relative to the null hypothesis that it is drawn from the chain’s stationary distribution, it is not necessary to wait until the chain reaches its stationary distribution. Testing the initial state against its neighbors suffices, provided that there are enough neighbors. This theorem provides a means to determine whether a given state of a reversible Markov chain was selected from a stationary distribution. In our particular case, we seek to evaluate the significance of R_0 , which is the initial state of the SSN, with respect to the labeling function F_g . By applying the theorems above, instead of sampling from the SSN’s stationary distribution, we employ the serial test to identify motifs.

We design a novel MD framework based on the serial test named as MOSER. The details of MOSER are further discussed in Alg. 2.

Based on the single trajectory trend, according to a test introduced in [10], known as the $\sqrt{\varepsilon}$ -test, the best bound on possible on a single trajectory is a constant factor of $\sqrt{\varepsilon}$. In order to get the best possible bound (ε) which is equivalent to the bound with perfect sampling, we will use the serial test introduced earlier. The serial test requires two trajectories to start from the same point (R_0) and take two independent random walks. Figure 6 depicts an experiment performed on the well-known E. coli dataset, comparing the frequency trends of iterative single switching between motif and non-motif subgraphs. On the left, the frequency trend for a motif

Algorithm 2: MOSER

Input: G, k, t, p, u

- 1 $\mathcal{M} \leftarrow \emptyset$ // Motif set
- 2 $C_0 = C'_0 \leftarrow \mathcal{F}_k(G)$
- 3 $R_0 = R'_0 \leftarrow G$ // initial state of SSN
- 4 $t' \leftarrow \text{RandomInt}(1, t)$
- 5 **for** $1 \leq i \leq t'$ **do**
- 6 $R_i \leftarrow \text{SingleSwitch}(R_{i-1})$
- 7 $C_i \leftarrow \mathcal{F}_k(R_i)$ // e.g. TAC, ATAC, cf. Sec. 5
- 8 **for** $1 \leq i \leq (t - t')$ **do**
- 9 $R'_i \leftarrow \text{SingleSwitch}(R'_{i-1})$
- 10 $C'_i \leftarrow \mathcal{F}_k(R'_i)$
- 11 **forall** $g \in S_k$ **do** // subgraphs of size k in G
- 12 **if** $\text{SerialTest}(g, C, C', p, u)$ **then**
- 13 $\mathcal{M} \leftarrow \mathcal{M} \cup \{g\}$
- 14 **return** \mathcal{M}

identified by traditional MD solutions is shown. The frequency of the motif drops significantly after the first few thousand switches and stabilizes around a value lower than the starting point ($F_g(G)$). On the right, the frequency trend for a non-motif subgraph (3-path) is displayed. The trend oscillates from the beginning, and the frequency does not follow a specific trend, as expected. Notably, if only the initial counts of the subgraphs were considered to identify motifs, the 3-path would have been mistakenly chosen instead of the tailed triangle since the initial count of the 3-path ($\sim 7.5K$) is considerably higher than that of the tailed triangle (~ 700). The burnout line will be discussed in Sec.6.

4.2 MOSER Complexity

We show the complexity of MOSER:

$$O \left(\overbrace{n \cdot T(\xi(G))}^{\text{Sampling}} + \overbrace{T(\mathcal{F}_k(G)) + \sum_{R_i \in R} T(\mathcal{F}_k(R_i))}^{\text{Counting subgraphs on } G \text{ and } R} \right) \quad (9)$$

As shown above, the parameter τ discussed earlier, which can become very large on large graphs, is removed from the complexity compared to the baseline MD complexity discussed in Eq. 7. This improvement removes the first bottleneck of baseline MD by eliminating the need for expensive uniform random graph sampling.

5 INCREMENTAL SUBGRAPH COUNTING ALGORITHM

In the previous section, we formalized the sampling part of the Motif Discovery, and we proposed a framework to rigorously calculate the p -value using biased samples. In this section we are aiming to describe how to leverage the biased sampling in the subgraph counting phase based on the below observations:

- (1) At step i of the switching, the subgraph frequencies of step $i - 1$ are known.

- (2) A single switch entails exactly two edge deletions and additions. Every deletion/addition only affects the k -node subgraphs of $(k-2)$ -hop neighbourhood around the affected edge since the edge already covers 2 nodes.

In what follows, we introduce the *Track And Count*(TAC) algorithm that leverages the above properties.

5.1 Track And Count

The input of the TAC is a graph G_{i-1} , and $\mathcal{F}_k(G_{i-1})$. This algorithm first picks two random edges from G_{i-1} , applies a single switch to G_{i-1} , updates the k -node subgraph frequencies based on the switch, and returns G_i and $\mathcal{F}_k(G_i)$.

To understand how the algorithm works, we are going to break TAC into the following steps and discuss each in detail:

Edge Selection: The algorithm proceeds by selecting two edges at random from the set of all edges, denoted by E . It should be noted that some edge selections may result in invalid switches, as illustrated in Figure 4. In such cases, the edges are re-selected. Once the edges have been selected for switching, the next step is to perform the switch. This can be broken down into two distinct steps: First, the old edges are removed, and second, the new edges are added. For each switch, denoted as $(a, b), (c, d) \rightarrow (a, d), (c, b)$, there are four atomic actions. Specifically, two edge deletions are performed to remove the edges $(a, b), (c, d)$, and two edge additions are performed to add the edges $(a, d), (c, b)$. In each atomic action, exactly one edge is involved.

Tracking: A fundamental observation is that removing/adding an edge affects only the frequency of subgraphs containing that edge, leaving the frequency of other subgraphs unchanged. Building on this insight, the first step of the Tracking algorithm is to identify the k -node subgraphs that include the edge slated for removal/addition. Since every edge connects two nodes, determining the number of k -node subgraphs around an edge requires a search of the $(k-2)$ -node neighborhood using a method such as breadth-first search.

The TAC algorithm (Alg. 3) takes as inputs a graph $G = (V, E)$, edges e_1, e_2 that are switched, k (the size of the subgraph of interest), and $\mathcal{F}_k(G)$ (the frequencies of k -node subgraphs in G). The algorithm outputs $G' = (V, E')$ and the updated subgraph frequencies $\mathcal{F}_k(G')$ after the single switch. The *GetNeighbourhood* function, performs a breadth-first search around the edge, with the $(k-2)$ depth and finds returns the $D \subset G$.

Algorithm 3: Track and Count

Input : $G = (V, E)$, $e_1, e_2, k, \mathcal{F}_k(G)$.

Output: $G' = (V, E')$, $\mathcal{F}_k(G')$

- 1 $\mathcal{N} \leftarrow \text{GetNeighbourhood}(G, k-2, e_1, e_2)$ // e.g. via BFS
 - 2 $\mathcal{N}' \leftarrow \text{SingleSwitch}(\mathcal{N}, e_1, e_2)$
 - 3 $\Delta \leftarrow \mathcal{F}_k(\mathcal{N}') - \mathcal{F}_k(\mathcal{N})$
 - 4 $G' \leftarrow \text{SingleSwitch}(G, e_1, e_2)$
 - 5 $\mathcal{F}_k(G') \leftarrow \mathcal{F}_k(G) + \Delta$
 - 6 **return** $G'(V, E')$ and $\mathcal{F}_k(G')$
-

The TAC algorithm exploits the two previously mentioned key observations. Specifically, TAC selectively focuses on the local

neighborhood of the impacted edges and avoids unnecessary computations in unaffected areas. TAC is versatile and supports all motif sizes, and works seamlessly with both directed and undirected graphs. Moreover, TAC facilitates easy integration with existing subgraph counting solutions.

5.2 Accelerated Track And Count

Although TAC is a general algorithm that supports different problem settings, it can be further optimized for specific setups that are commonly studied in the literature. In particular, small undirected subgraphs (up to 5-node) have been proven to be the most useful in practice [32]. For this purpose, we introduce *Accelerated Track And Count* (ATAC), an optimized algorithm based on ESCAPE [32], that is more efficient than TAC.

The key concept is that subgraphs can be decomposed into smaller patterns, and the frequency of the subgraph can be computed efficiently based on the frequencies of these patterns using formulas which we will refer to as the **ESCAPE Formulas** (ESF). Notably, this approach avoids the need to enumerate all subgraph instances and bypasses the isomorphism checks. The ESF returns the non-induced subgraph counts, from which the induced counts can be easily derived using a simple (invertible) linear transformation [33]. We now take a closer look at ESF, then we will extend them to support the incremental nature of tracking.

Frequency Calculation for 4-node Subgraphs: The formulas in this section are proven in [2, 15, 32]. Below you can find a list of 4-node motifs with their calculation formula:

- (1) # Wedge = $\sum_{i \in V} \binom{d(i)}{2}$
- (2) # Triangle = $\frac{1}{3} \sum_{(i,j) \in E} |N(i) \cap N(j)|$
- (3) # 3-Star = $\sum_{i \in V} \binom{d(i)}{3}$
- (4) # 3-Paths = $\sum_{(i,j) \in E} (d(i)-1)(d(j)-1) - 3T(G)$
- (5) # Tailed-Triangles = $\sum_{i \in V} t(i)(d(i)-2)$
- (6) # Diamonds = $\sum_{e \in E} \binom{t(e)}{2}$

Now we can state the derived formula for tracking the count differences after edge $e = (i, j)$ is removed from the graph. Below, you can find the formulas we have derived, the mathematical steps and proofs are added in the appendix.

- (1) $\Delta \text{ Wedge} = -(d(i) + d(j) - 2)$
- (2) $\Delta \text{ Triangle} = -|N(i) \cap N(j)|$
- (3) $\Delta \text{ 3-Star} = -\frac{1}{2}((d(i)-1)(d(i)-2) + (d(j)-1)(d(j)-2))$
- (4) $\Delta \text{ 3-Paths} = \sum_{u \in N(i) \cup N(j)} 1 - d(u) - (d(i)-1)(d(j)-1) - 3 \cdot \Delta T$
- (5) $\Delta \text{ Tailed-Triangles} = \Delta t \cdot (d(i)-2) - t_2(i) + \Delta t \cdot (d(j)-2) - t_2(j) + \sum_{u \in V^*} 2 - d(u)$
- (6) $\Delta \text{ Diamonds} = \sum_{e^* \in E^*} 1 - t(e^*) + \binom{\Delta T}{2}$

In the above formulation, V^* and E^* refer to the affected edges and nodes, namely the edges and nodes that form a triangle with e .

We can track the number of 4-cycles that are removed by first identifying the neighboring sets of nodes i and j , denoted as $N(i)$ and $N(j)$, respectively. We then check how many edges in $N(i)$ are connected to nodes in $N(j)$ to find 4-cycles. To track 4-cliques, we look for two additional edges between the neighbors of i and j , and vice versa.

The above formulation supports subgraph frequency update the general case of dynamic graphs with deletion/addition of edges. For

the special case of switching, the node degrees always remain unchanged, so the non-induced frequencies of star-shaped subgraphs (wedges, 3-stars, etc) also remain unchanged as they only rely on the node degrees.

As shown in Sec. 6, ATAC improves the baseline solution implemented using the ESCAPE counting algorithm by up to five orders of magnitude.

5.3 Time and Space Complexity Analysis

We have shown in Eq. 7, that the complexity of the state-of-the-art solutions is determined by the sampling complexity and the subgraph counting complexity. As shown in Eq. 9, we have reduced the *sampling* complexity. In this part we highlight the impact of TAC and ATAC on reducing the subgraph counting complexity.

TAC: For a fair comparison, we use the same counting solution for both TAC and baseline, since TAC is compatible with any subgraph counting solution. We refer to the subgraph counting complexity as $T(\mathcal{F}_k(\cdot))$. Since the counting solution is identical, the advantage lies in only counting on a substantially smaller portion of the input graph, in contrast to baseline methods that perform counting on the entire graph. We can update the counting part of the Eq.9 to fit the TAC as below:

$$O \left(\underbrace{n \cdot T(\xi(G))}_{\text{Sampling}} + \underbrace{T(\mathcal{F}_k(G)) + 2 \cdot \sum_{D_i \in \mathcal{D}} T(\mathcal{F}_k(D_i))}_{\text{Counting subgraphs on } G \text{ and } D} \right) \quad (10)$$

Where $D_i \in \mathcal{D}$ refers to the $(k-2)$ -hop neighbourhood around the edges added/removed during the i -th single switch operation. Every single switch requires two counting operations, once before and once after applying the switch. The size of subgraph $D_i = (V_i, E_i)$ is determined as below:

$$\begin{aligned} |V_i| &\leq \min \left(4 \cdot (2 \cdot d_{\max}^{(k-2)}), |V| \right) \\ |E_i| &\leq \min \left(1/2 \cdot |V_i| \cdot d_{\max}, |E| \right) \\ &= \min \left(4 \cdot d_{\max}^{(k-1)}, |E| \right) \end{aligned}$$

The upper bound for $|V_i|$ is calculated by considering that for every edge containing nodes a and b , both a and b need to be expanded for $k-2$ hops, as a and b are already present in the subgraph. This expansion is based on a pessimistic assumption that a, b , and their neighbors have d_{\max} neighbors. The factor of 4 in the calculation is due to the involvement of four edges in each switch operation. Similarly, the upper bound for $|E_i|$ is determined using a pessimistic assumption that every node in $|V_i|$ has the maximal degree.

ATAC: In this part, we break down the complexity of ATAC analyzing every single formula separately.

(1) *Wedge* and (3) *3-Star*: The non-induced frequency update can be done in $O(1)$ because it only depends on the node degrees in the affected edges which can be accessed in constant time. (2) *Triangle*: This requires the calculation of $N(i) \cap N(j)$, which can be done in $O(d_{\max})$ given that $N(i)$ and $N(j)$ are sorted. (4) *3-Path*: This only requires the degrees of the one-hop neighbours of e , and the updated global triangle count (calculated on the previous

Table 2: MD Datasets from bioinformatics community (top) and the data mining community (bottom).

Bioinformatics Graphs	$ V $	$ E $	d_{avg}	d_{max}	CC_{avg}
Dolphins (DO)	62	159	5.12	12	0.129
Social (SO)	67	182	4.23	11	0.233
Electronic (EL)	252	399	3.16	14	0.028
E. coli (EC)	672	1276	2.57	23	0.047
Yeast (YE)	688	1079	3.13	71	0.023
Data Mining Graphs	$ V $	$ E $	d_{avg}	d_{max}	CC_{avg}
ca-AstroPh (AS)	18.8K	198K	21.10	504	0.31
flickr (FL)	105K	2.32M	43.74	5425	0.044
soc-google-plus (SG)	211K	1.50M	10.82	1790	0.111
web-google (WG)	876K	4.32M	9.87	6332	0.257
com-youtube (YT)	1.34M	3M	5.26	28754	0.040

step). One-hop degree gathering would be $O(d_{\max})$ for the worst case scenario. (5) *Tailed-Triangles*: We access the degree ($O(1)$) of every node that forms a triangle with e ($O(d_{\max})$). (6) *Diamond*: It is necessary to iterate over all the *triangles* that e is involved in ($O(d_{\max})$). For each edge e^* in those triangles we calculate $t(e^*)$, these calculations can be done in $O(1)$, as we store the $t(e)$ for all $e \in E$ in the original triangle counting phase. (7) *4-Cycles*: We check whether the nodes in $N(i)$ are connected to the nodes in $N(j)$ or not, which takes $O(d_{\max}^2)$. (8) *4-Cliques*: We go over the nodes that form a triangle with our candidate edge and check if they are connected to each other or not, so the algorithm would take $O(|t(e)|^2)$.

We conclude that the most time-consuming 4-node subgraphs to track during deletion or addition are the 3-paths and 4-cycles/cliques. These bottlenecks will take $O(|t(e)_{\max}|^2 + d_{\max}^2)$ time, which can be reduced to $O(d_{\max}^2)$ since $|t(e)_{\max}| \leq d_{\max}$ in the worst-case scenario. On average, the time complexity can be reduced to $O(d_{avg}^2)$.

As mentioned before, this algorithm is extendable to support 5-node subgraphs, the details of the this extension and the formulas are beyond the scope of this paper.

5.3.1 Space Analysis. The space used by most of the state-of-the-art MD algorithms consists of two main components: the original graph size and the index built to reduce isomorphism checks. These two components make up the memory used by the motif counting algorithm. As Track and Count is a versatile algorithm that utilizes the counting algorithms, its memory usage is also affected by the counting solution. However, our proposed algorithm and framework do not add extra memory usage to the counting algorithm.













6 EXPERIMENTS

We used C++ and Python to implement our algorithms, and executed our experiments on a machine that features an 8-Core Intel i9 processor clocked at 2.3 GHz and equipped with 16GB of memory. We conducted a series of experiments to test our algorithms using both popular MD datasets from the bioinformatics community (Table 2 top) [17], and the popular data mining community datasets (Table 2 bottom) [20, 39]. We tested our algorithms extensively to ensure their efficacy and accuracy. For further information on the datasets used in our experimentation, please refer to Table 2.

Our dataset selection was influenced by multiple factors. The first half of the datasets were sourced from the bioinformatics community [17] and were chosen for two primary reasons. Firstly we utilized them to compare the accuracy of our results with baseline solutions as baseline solutions cannot scale well to the larger datasets, and secondly to showcase our superior efficiency and scalability compared to state-of-the-art solutions. The latter half of the datasets were acquired from the data mining community [20, 39] and were considerably larger than the former datasets and any other dataset used in previous methods. We employed them to test our algorithm’s scalability on larger graphs, and to demonstrate its efficiency in handling big data. In evaluating our algorithm’s performance, we compared it to state-of-the-art solutions in the motif discovery domain, namely Kavosh[17], QuateXelero (QX) [19], G-tries[37], and ACC-Motif[24]. We refer to these as $\text{BASE}^{\text{initial}}$, for example we use BASE^{K} to refer to Kavosh. Similarly, we use $\text{MOSER}^{\text{initial}}$ to refer to the implementation of MOSER without any optimizations on the counting method (namely TAC and ATAC). MOSER^+ uses TAC as the counting optimization technique, while MOSER^{++} leverages ATAC. As suggested in [10], we use the first 10% of the samples as the burnout stage².

6.1 Method

We begin by presenting the parameter settings for both MOSER and the BASE algorithms. Following this, we conduct a correctness test on the datasets that are supported by the BASE algorithms, by discovering motifs using MOSER and BASE, and comparing these two sets with the Intersection over Union (IoU) similarity measure. Subsequently, we analyze the efficiency improvement of our method, and finally, we perform a case study to demonstrate the effectiveness of our approach on motif-based downstream tasks.

Parameter Settings: Our experiments were conducted on the datasets mentioned above, with motif sizes of $k = 3, 4, 5$. Here we list a few motifs that will be mentioned in the following sections:  Wedge,  Triangle,  3-star,  3-path,  4-cycle,  Tailed Triangle (T.Triangle for short),  Diamond,  4-clique,  4-star,  4-path,  Tailed 4-cycle (T.4-cycle for short),  Long Tailed Triangle (L.T.Triangle for short). The BASE solutions were executed with $3 \times |E|$ switches to produce one sample, and $n = 10K$ samples, and a significance threshold of $p = 0.01$. To ensure fair comparison, MOSER was also executed with $t = 10K$ samples, and $p = 0.01$. All the experiments have been performed with this setup, unless mentioned otherwise.

6.2 Correctness

To compare the effectiveness of our proposed solution with the BASE framework, we conducted experiments using the MOSER framework on the first half datasets with the parameter settings mentioned previously. We refer to the motif set generated using the MOSER as $\mathcal{M}_{\text{MOSER}}$ and the motif set generated using the BASE as $\mathcal{M}_{\text{BASE}}$. To compare the two sets of motifs, we used the Intersection over Union (IoU) metric, also known as the Jaccard distance, defined

as follows:

$$\text{IoU} = \frac{|\mathcal{M}_{\text{MOSER}} \cap \mathcal{M}_{\text{BASE}}|}{|\mathcal{M}_{\text{MOSER}} \cup \mathcal{M}_{\text{BASE}}|}$$

The IoU metric ranges between 0 and 1, where a score of 1 indicates that the motif sets generated using both frameworks are identical, while a lower score indicates more differences between the resulting sets. In our experiments, the IoU score for all the datasets and motif sizes was exactly 1, indicating that the MOSER framework generates identical results to the BASE framework in a more computationally efficient manner.

6.3 Efficiency & Scalability

Comparison with the BASE We begin by comparing the efficiency of our fastest³ solution to the BASE solutions, including Kavosh, G-Tries, QX, and ACC-Motif, as shown in Table 3. The selected parameters used in these experiments are $k = 4$ and $n = 10K$. For the AS graph, we ran the algorithm with $n = 100$, and scaled the results to match the previous setup since the running time grows almost linearly with the number of random graphs.

Table 3: Runtime (s) Comparison between MOSER and BASE

Dataset	Fastest ²	BASE ^K	BASE ^Q	BASE ^G	BASE ^A	Speedup
SO	0.15	8.09	6.05	8.22	5.33	35X
DO	0.38	56.62	35.8	41.32	11.76	30X
EL	0.40	8.05	13.91	8.04	5.87	14X
YE	0.54	1065.99	58.98	50.01	19.79	36X
EC	0.48	56.62	35.8	41.32	11.76	24X
AS	8.79	5.6 M	357 K	576 K	100 K	12,486X

We can see in Table. 3 that the BASE^{A} outperforms other BASE solutions, it is because it leverages multi-core processing. The downside of the BASE^{A} is that it suffers from high variability of running times. For this reason, we use the second fastest BASE algorithm (the BASE^{Q}) with low variability, in the rest of the experiments.

We have also conducted experiments to compare the performance of our approach w.r.t. to the motif size k , illustrated in Fig. 7.

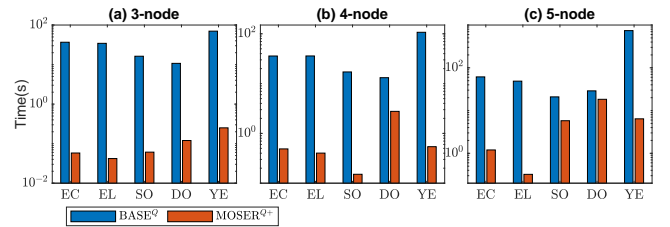


Figure 7: Efficiency comparison between the $\text{MOSER}^{\text{Q}+}$ and BASE^{Q} for Motif Discovery.

Comparison on larger graphs It is important to note that non of the available BASE methods are able to scale to the larger graphs used in our experiments. For example, the code provided by the Kavosh[17] and QX[19] papers crashes while loading graphs

²We do not use the first 10% of the steps made in the serial test.

³By fastest we mean, we used MOSER^{++} on the undirected networks, and $\text{MOSER}^{\text{Q}+}$ on the directed ones.

with one million nodes or greater. To enable comparisons on larger graphs, we implemented baseline motif discovery methods (BASE) using the state-of-the-art subgraph counting method, ESCAPE [32], which is faster than previous solutions and capable of scaling to graphs of one million nodes or more. The second part of our efficiency experiments compares our implementation of BASE using ESCAPE (BASE^E) with MOSER^{E+} and MOSER⁺⁺.

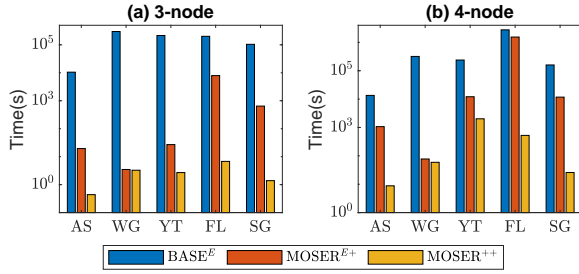


Figure 8: Comparison of Motif Discovery based on the MOSER^{E+} and MOSER⁺⁺ with BASE^E.

Fig. 8 shows that MOSER⁺⁺ is up to 5 orders of magnitude faster than BASE^E. Our proposed framework and algorithms improve upon the base framework in two dimensions: random graph sampling and subgraph counting. We conduct experiments to demonstrate the improvements in these two dimensions. Firstly, we compare the sampling time improvements of MOSER compared to BASE. Next, we evaluate the impact of BASE⁺ and BASE⁺⁺ on subgraph counting enhancements.

Sampling Effect Previous research has predominantly focused on optimizing subgraph counting, given its significant role in motif discovery. As far as we are aware, our study represents the first exploration of sampling complexity in this context. Prior works generally utilize small input graphs where the effect of sampling is negligible. However, as we scale towards graphs in the millions, we will demonstrate that sampling becomes a noticeable factor. Fig. 9 shows how sampling becomes a bottleneck on large graphs, and how MOSER is able to overcome this bottleneck.

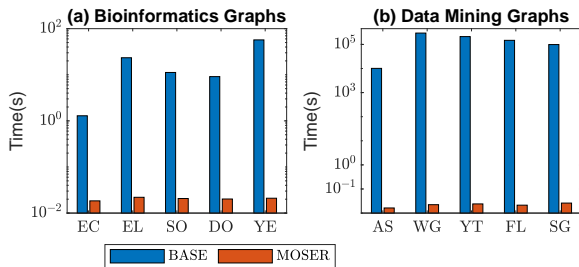


Figure 9: Generating Time of 10K unbiased samples using the Switching Method on (a) small graphs from the bioinformatics community and (b) large graphs from the data mining community.

Counting Effect In this part, we will focus on the efficiency gains on how using TAC incremental counting algorithm can improve over the BASE counting solution. Figs. 10 and 11 show the effectiveness of TAC.

To have the full

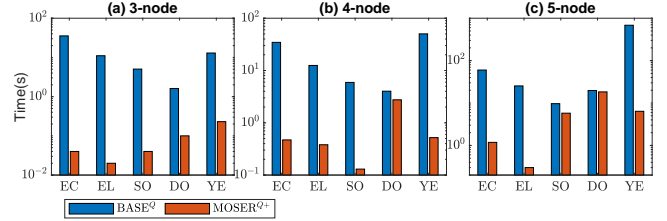


Figure 10: Counting efficiency comparison between the MOSER^{Q+} and BASE^Q on small dataset.

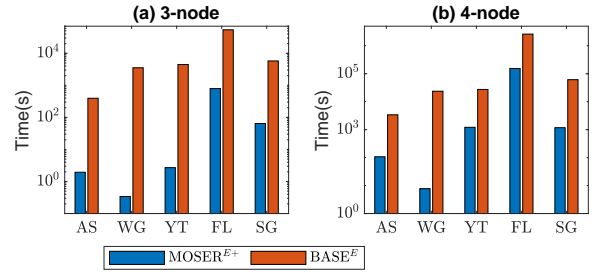


Figure 11: Counting efficiency of MOSER^{E+} and BASE^E on large datasets.

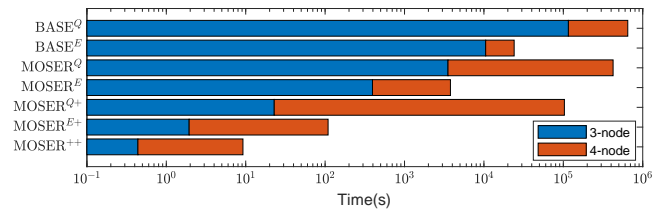


Figure 12: Motif Discovery efficiency comparison between the different BASE (BASE^Q, BASE^E) and MOSER variations (MOSER^Q, MOSER^E, MOSER^{Q+}, MOSER^{E+}, and MOSER⁺⁺), on the AS dataset.

Fig. 12 fully compares the efficiency of different methods with all the settings. We start by the BASE^Q as the state-of-the-art baseline solution, then we show that how using ESCAPE can improve the baseline shown in BASE^E. Next, we compare the effectiveness of using MOSER without the counting optimizations. Then we add the TAC optimization to the setup, and finally we show how MOSER⁺⁺ can improve the efficiency of MD.

6.4 Case Study

We select the top-5 motifs by both MD (i.e., the top-5 most frequent k -node subgraphs that pass the *significance* test with $p = 0.01$) and Subgraph Counting (SC) approaches (i.e., the top-5 most frequent k -node subgraphs) in a subset of the *Gavin* dataset, which is a standard dataset used in link prediction tasks [22], where each node denotes a yeast cell protein and each edge denotes an interaction between the corresponding proteins. Then these motifs are applied to the Motif-aware Link Prediction task developed by the authors in [22, 40, 41], and the effectiveness of each motifs are reported in Table 4 with the Area Under the Curve (AUC) score. $F_g(G)$ and $F_g(R)$ are also reported to help understand the difference between SC results and MD results. As show in the table, the top-5 g_{MD} obtain 28% higher AUC scores on average than the top-5 g_{SC} . It is also observed that many of the g_{SC} cannot pass the *significance* test; hence, we call such frequent k -node subgraphs as non-motifs.

Table 4: Link prediction effectiveness of top-5 motifs that selected by MD (left side) and MC (right side) respectively.

Top-5 g_{MD}	$F_g(G)$	$F_g(R)$	AUC	Top-5 g_{MC}	$F_g(G)$	$F_g(R)$	AUC
L.T.Triangle	2.7K	120	0.65	4-path	19.0K	20.0K	0.52
T.4-cycle	870	300	0.58	4-star	14.0K	16.4K	0.62
T.Triangle	650	55	0.68	3-path	7.1K	7.3K	0.49
Triangle	58	3	0.83	3-star	5.1K	5.7K	0.52
4-cycle	57	15	0.87	L.T.Triangle	2.7K	120	0.65
Average	867	99	0.72	Average	9.6K	9.9K	0.56

7 RELATED WORKS

The idea of MD that we use in this work, was first used by Milo et al. [25]. There has been some recent surveys studying state-of-the-art network MD solutions. [30, 48, 49] We categorize the different MD solutions based on the different Subgraph Souting (SC) and random graph generation method they use.

Subgraph Counting Based on subgraph counting, the MD algorithms can be divided into two main categories of *Exact Counting*, and *Approximate Counting*. The exact counting methods [8, 11, 17–19, 24, 29, 37, 42, 47] use the SC methods that return the exact subgraph frequencies. Both [47] and [18] have the support for the Approximate approach that samples subgraphs from the input graph G and the random graphs in R , and then identify motifs from those samples. Noga Alon et al. [5] used an approximate approach to find undirected non-induced network motifs. Approximate algorithms tend to offer superior efficiency in comparison to exact methods. However, they do not provide any assurances regarding the quality of the motifs found. For this reason, we focus on the exact counting methods in this work.

We can further categorize the exact subgraph counting methods into *Subgraph-Centric* [8, 11, 29, 42] and *Network-Centric* [17–19, 24, 37, 47]. The former counts all the instances of a single given subgraph g in a given graph G . The latter counts all the k -node subgraph patterns that are present in G . The subgraph-centric approach is efficient for *testing* if a single candidate subgraph g is a motif or not, however for finding all motifs one requires to test all the possible k -node candidate subgraphs. This renders the approach infeasible to discover all k -node motifs. In contrast, network-centric

methods only count all the *present* k -node subgraphs in G with a single pass over G , while discovering all the k -node motifs from G . Note that TAC supports both counting approaches, although our experiments are centered around the network-centric methods. There have been methods to alleviate the bottleneck of counting by parallelizing the counting [38] or utilizing GPU to accelerate the performance [23]. Almost all of the mentioned solutions focused on discovering both directed and undirected motifs except [5, 8] which can only support undirected discovery. Depending on the subgraph counting used, TAC can support both directed and undirected discovery.

There has been recent advancements in the subgraph counting research, according to a survey by Ribeiro et al. [36], the ESCAPE [32, 33] is among the fastest and most recent subgraph counting solutions which has not been used in the network motif discovery prior to this work.

Random Graph generation The null model employed to test the statistical significance of subgraphs is a random graph that is degree-equivalent to G . Several techniques have been developed to generate such random graphs. In their study, Milo et al. [26] conducted an empirical analysis comparing three primary algorithms for generating degree-preserving random graphs: (1) The Switching Method [35, 43, 45], (2) The Matching Algorithm [28], and (3) Go With The Winners [3].

The Matching Algorithm starts with an empty graph consisting of $|V|$ nodes, where each node is assigned a set of *in-stubs* and *out-stubs* based on its in-degree and out-degree in G , respectively. In each iteration, an in-stub and an out-stub are randomly selected and connected by an edge until all in-stubs and out-stubs are used. If the process generates a self-loop or multiple edges, the graph is discarded and the procedure is restarted. However, this algorithm has the drawback of rarely sampling from Λ [26].

The Go With The Winners algorithm is an improved version of the Matching Algorithm that aims to converge faster by utilizing a colony of random graphs at each step and sampling from that colony in the end. This method produces high-quality uniform samples, but it is not very efficient [26].

On the other hand, the Switching Algorithm strikes a balance between efficiency and quality, which is why it is the preferred method for generating degree-preserving random graphs by most state-of-the-art network motif discovery tools.

8 CONCLUSIONS

While most prior work using motifs for high-order analytics over large graphs focus on counting given motifs, motif discovery is a more fundamental task which is just as important. Known MD solutions fail to scale to large graphs and are based on approaches that do not come with any guarantees. To address these limitations, we introduced a novel MD framework called MOSER based on the serial test, and developed new algorithms (TAC and ATAC) on top of MOSER to achieve up to five orders of magnitude improvement in efficiency. Extensive experimental evaluation demonstrated the efficacy of our approach, and we presented case studies to demonstrate the practical utility of our methods in downstream tasks.

REFERENCES

- [1] Ghadeer Abuoda, Gianmarco De Francisci Morales, and Ashraf Aboulnaga. Link prediction via higher-order motif features, 2019.
- [2] Nesreen Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick G. Duffield. Efficient graphlet counting for large networks. *2015 IEEE International Conference on Data Mining*, pages 1–10, 2015.
- [3] D. Aldous and U. Vazirani. "go with the winners" algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press.
- [4] David Aldous. Markov chains and mixing times (second edition) by david a. levin and yuval peres. *The Mathematical Intelligencer*, 41(1):90–91, November 2018.
- [5] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, July 2008.
- [6] Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.
- [7] JULIAN BESAG and PETER CLIFFORD. Generalized monte carlo significance tests. *Biometrika*, 76(4):633–642, 1989.
- [8] Jin Chen, Wynne Hsu, Mong Lee, and See-Kiong Ng. Nemofinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. volume 2006, pages 106–115, 01 2006.
- [9] Maria Chikina, Alan Frieze, Jonathan C. Mattingly, and Wesley Pegden. Separating effect from significance in markov chain tests. *Statistics and Public Policy*, 7(1):101–114, 2020.
- [10] Maria Chikina, Alan Frieze, and Wesley Pegden. Assessing significance in a markov chain without mixing. *Proceedings of the National Academy of Sciences*, 114(11):2860–2864, 2017.
- [11] Joshua A Grochow and Manolis 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.
- [12] Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona. Graph isomorphisms in quasi-polynomial time, 2017.
- [13] Himanshu and Sarika Jain. Impact of memory space optimization technique on fast network motif search algorithm. In *Advances in Computer and Computational Sciences*, pages 559–567. Springer Singapore, 2017.
- [14] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322. ACM, 2014.
- [15] Madhav Jha, C. Seshadhri, and Ali Pinar. 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, page 495–505, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.
- [16] R Kannan, S Vempala, and P Tetali. Simple markov-chain algorithms for generating bipartite graphs and tournaments. 6 1997.
- [17] Zahra Razaghi Moghadam Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics*, 10(1):1–12, 2009.
- [18] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 03 2004.
- [19] Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, Ina Koch, and Ali Masoudi-Nejad. Quatexelero: An accelerated exact network motif detection algorithm. *PLOS ONE*, 8(7):1–15, 07 2013.
- [20] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.
- [21] Xiaodong Li, Tsz Nam Chan, Reynold Cheng, Kevin Chang, Caihua Shan, and Chenhao Ma. Motif paths: A new approach for analyzing higher-order semantics between graph nodes. *HKU Technical Report (https://www.cs.hku.hk/data/techreps/document/TR-2019-04.pdf)*, 2019.
- [22] Xiaodong Li, Reynold Cheng, Kevin Chang, Caihua Shan, Chenhao Ma, and Hongtai Cao. On analyzing graphs with motif-paths. *PVLDB*, 2021.
- [23] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiao-Li Li. Network motif discovery: A gpu approach. *IEEE TKDE*, 29(3):513–528, March 2017.
- [24] Luis A. A. Meira, Vinicius R. Máximo, Álvaro L. Fazenda, and Arlindo F. da Conceição. acc-motif: Accelerated network motif detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(5):853–862, 2014.
- [25] R. Milo. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.
- [26] R. Milo, N. Kashtan, S. Itzkovitz, M. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv: Statistical Mechanics*, 2003.
- [27] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
- [28] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6(2-3):161–180, March 1995.
- [29] Saeed Omid, Falk Schreiber, and Ali Masoudi-Nejad. Moda: an efficient algorithm for network motif discovery in biological networks. *Genes & genetic systems*, 84(5):385–395, 2009.
- [30] Sabyasachi Patra and Anjali Mohapatra. Review of tools and algorithms for network motif discovery in biological networks. *IET Systems Biology*, 14(4):171–189, August 2020.
- [31] Oskar Perron. Zur theorie der matrices. *Mathematische Annalen*, 64(2):248–263, June 1907.
- [32] Ali Pinar, C. Seshadhri, and V. Vishal. Escape: Efficiently counting all 5-vertex subgraphs. 10 2016.
- [33] Ali Pinar, C. Seshadhri, and V. Vishal. Escape: Efficiently counting all 5-vertex subgraphs, technical report. 10 2016.
- [34] Nataša Pržulj and Noël Malod-Dognin. Network analytics in the age of big data. *Science*, 353(6295):123–124, 2016.
- [35] J. Ray, A. Pinar, and C. Seshadhri. A stopping criterion for markov chains when generating independent random graphs. *Journal of Complex Networks*, 3(2):204–220, December 2014.
- [36] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets. *ACM Comput. Surv.*, 54(2), mar 2021.
- [37] Pedro Ribeiro and Fernando Silva. G-tries: An efficient data structure for discovering network motifs. pages 1559–1566, 01 2010.
- [38] Pedro Ribeiro, Fernando Silva, and Luis Lopes. Parallel discovery of network motifs. *Journal of Parallel and Distributed Computing*, 72(2):144–154, February 2012.
- [39] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.
- [40] Ryan A Rossi, Anup Rao, Sungchul Kim, Eunye Koh, and Nesreen Ahmed. From closing triangles to closing higher-order motifs. In *Companion Proceedings of the Web Conference 2020*, pages 42–43, 2020.
- [41] Ryan A Rossi, Anup Rao, Sungchul Kim, Eunye Koh, Nesreen K Ahmed, and Gang Wu. From closing triangles to higher-order motif closures for better unsupervised online link prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4085–4093, 2021.
- [42] Falk Schreiber and Henning Schwöbbermeyer. Mavisto: a tool for the exploration of network motifs. *Bioinformatics*, 21(17):3572–3574, 2005.
- [43] Alexandre Stauffer and Valmir Barbosa. A study of the edge-switching markov-chain method for the generation of random graphs. *Computing Research Repository - CORR*, 12 2005.
- [44] Lewi Stone, Daniel Simberloff, and Yael Artzy-Randrup. Network motifs and their origins. *PLOS Computational Biology*, 15, 04 2019.
- [45] R. Taylor. Contrained switchings in graphs. In *Lecture Notes in Mathematics*, pages 314–336. Springer Berlin Heidelberg, 1981.
- [46] Charalampos E Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1451–1460, 2017.
- [47] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359, 2006.
- [48] Feng Xia, Haoran Wei, Shuo Yu, Da Zhang, and Bo Xu. A survey of measures for network motifs. *IEEE Access*, 7:106576–106587, 2019.
- [49] Shuo Yu, Yufan Feng, Da Zhang, Hayat Dino Bedru, Bo Xu, and Feng Xia. Motif discovery in networks: A survey. *Computer Science Review*, 37:100267, August 2020.