# MODESA: Discovering Motifs from Large Complex Networks


## ABSTRACT

Given a graph $G$ and a small integer $k$, the problem of *motif discovery* (MD) is to find all the $k$-node subgraphs (called motifs) that are statistically significant in $G$. Motifs, which are fundamental building blocks of $G$, allow the characteristics of $G$ to be better understood. In recent years, motif-based graph analysis has been proposed, which employs motifs to perform graph mining tasks such as link prediction and embedding effectively. Existing MD solutions, however, scale poorly with graph and motif sizes. Moreover, the accuracy of these solutions is not clear. These problems hinder the use of motifs for analyzing large graphs. To address these issues, we show that MD can be transformed to a graph random walk problem. Such a a transformation not only reduces the number of parameters required for performing MD, but also allows us to understand the amounts of sampling effort on $G$ required for achieving the accuracy required. Based on these results, we propose MODESA, which is a novel solution framework that enables MD to be performed efficiency and accurately. We further improve the performance of MODESA with the use of state-of-the-art motif-counting solutions. Experimental results on several large graph datasets show that (1) MODESA achieve up to 4-order-of-magnitude improvement over existing approaches, and (2) the motifs found from MODESA enables effective link prediction.



## KEYWORDS

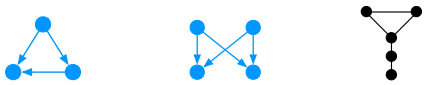
Motif Discovery, Graph Mining, Networks

**ACM Reference Format:**

. 2018. MODESA: Discovering Motifs from Large Complex Networks. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX


## 1 INTRODUCTION

A motif, also known as a *higher-order structure* or *graphlet*, is conceptually a fundamental "basic building block" of a large and complex graph [15, 19]. According to [17], a motif is a small subgraph of a given graph that occur more than statistically expected. Motifs facilitate the understanding of the non-random, structural, or evolutionary design principles used to construct the graph [24]. For example, a feed-forward loop motif (Fig. 1a) and a bi-fan motif (Fig. 1b), commonly used in systems biology, are used to study regulatory control mechanism in gene transcriptional networks [17]. A triangle motif (i.e., 3-node clique with undirected edges) is important to study connections of people in social networks and epidemiological contact networks [24]. Recently, motifs have been employed in various graph analytics tasks, including clustering [3, 12], link prediction [12], conductance [25], and community search [6]. Fig. 1 illustrates common motifs that have been studied in the literature.



**(a) feed-forward loop** **(b) bi-fan** **(c) tailed triangle**

**Figure 1: Examples of motifs.**

A common issue facing the above tasks is that they require the input of one or more motifs. In fact, *motif discovery* **(MD)**, or the finding of motifs from a graph, has been well studied in the literature. Given a graph $G$, a motif is a $k$-node graph $m$ that is *statistically significant* in $G$ [17]. Specifically, let $F_G(m)$ be the *frequency* of $m$ in $G$ (i.e., the number of subgraphs in $G$ (or *instances*) that are isomorphic to $m$). Then $m$ is a motif of $G$ if (1) $F_G(m)$ is larger than a threshold; and (2) $F_G(m)$ is probabilistically larger than $F_R(m)$, where $R$ is a random graph derived from $G$ whose node degree distribution is equal to that of $G$. While this definition has been extensively used in MD algorithms (e.g., [8, 13, 22]), its use is limited to small graphs (of hundreds to thousands of nodes). Particularly, existing MD algorithms often suffer from poor performance. For example, it took Kavosh [8], a state-of-the-art MD solution, more than 6 hours to find 3-node motifs in a $30k$-node-graph in our experiments. The problem exacerbates when the required size $k$ of $m$ increases. For example, when $k = 7$, there are more than 872 million types of candidate motifs with directed edges. However, in the literature, motifs of sizes up to 12 were also discovered. It is non-trivial to compute the $F_G$ and $F_R$ values for these "large" motifs. Yet, another problem of existing solutions is that there lacks a theoretical study of how many graph samples are needed to achieve the required accuracy, and the accuracy of the results is just shown empirically.

Our new solution can effectively return significant motifs while also reducing the number of parameters needed to be set compared with the definition in [17]. The main idea is to transform this problem definition, which involves comparing the frequency of candidate graph $m$ with those of random graphs, to a Monte Carlo Markov Chain problem. Based on this transformation, we derive the number of graph samples needed for the MD problem to achieve the required accuracy level. We further develop a new solution framework, known as **MO**tif **D**iscovery using **E**dge **S**witching **A**lgorithm (or *MODESA* in short)". An additional benefit of MODESA is that the number of input parameters is significantly reduced, allowing the motif discovery process to be more accessible to the MD user.

Based on this the MODESA framework, we design a highly-efficient algorithm called "**T**rack **A**nd **C**ount (TAC)". The main intuition of TAC is that the difference between the current and the next graph samples is often small because they only differ by a *node switching* operation [1] Therefore, instead of computing $F_R$ values for

[1]A node switching operation is a way of generating a new graph from the existing graph, based on swapping two randomly selected edges while preserving node degrees. We will discuss this in detail in Sec. 2.




*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*





ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

a random graph sample from scratch, their results can be derived by those obtained from the previous graph sample. [Rey: We further incorporate existing algorithms that can efficiently count 3-, 4-, and 5-node motifs for undirected graphs into the TAC framework, and devise "Fast TAC" (FTAC), which are up to 4-order-of-magnitude faster than state-of-the-art.]

To summarize, our contributions are:

(1) We transform the MD problem to a problem on a Markov chain called SSN, and derive the theoretical accuracy guarantees for the number of graph samples generated by node switching.
(2) Based on the problem transformation, We develop a novel MD framework known as MODESA.
(3) On top of MODESA, we develop two fast and scalable algorithms, known as TAC and FTAC. While TAC can discover motifs of any given sizes, FTAC is optimized for motifs whose motif-counting algorithms were developed.
(4) We perform extensive experiments and case studies, showing that motifs found by our solution are effective for link prediction and community detection.

The rest of the paper is organised as follows: In Sec. 2, we will formally introduce the notations & symbols used in the paper, then we will discuss switching algorithm and MD's original formulation. Next, in Sec. 4 we will formally define SSN and use it to reformulate the MD problem, then we will introduce our new definition of the Motif Discovery framework (MODESA) based on SSN. Sec. 3 shows how our new definition is used to design MODESA. Sec. 5 the running time and memory usage of MODESA is compared to the state-of-the-art methods in theory and empirically the rest of this section will discuss a case study on link prediction accuracy improvement using the motifs we discovered. In Sec. 6 we give a complete comparison over the available methods in the literature, and the last section is reserved for conclusions and future works.

## 2 PROBLEM ANALYSIS

In this section, we will discuss the problem of Network Motif Discovery in details, first we will start with the basic graph notation, then we will have an in-depth discussion on Network Motif Discovery framework and algorithm.

### 2.1 Graph Notation

A graph $G$ is a set $V(G)$ of nodes/vertices and a set $E(G)$ of binary relations between these nodes referred to as edges/connections. Mathematically, a graph $G$ can be defined as a pair $(V, E)$, where $V$ is a set of vertices, and $E$ is a set of edges between the vertices $E \subseteq \{(u, v) \mid u, v \in V\}$. In *directed* graphs, edge pairs $(u, v)$ are ordered pairs $(u \rightarrow v)$, while in *undirected* graphs there is no order $(u \leftrightarrow v)$. The number of the nodes in a graph is known as the *graph size* and it is written as $|V(G)|$ and the number of edges is written as $|E(G)|$. A graph is considered *simple* if there are no self-loops $(v, v)$ nor multiple edges (two or more edges connecting the same node pair). In this work, we refer to simple graphs unless mentioned.

*Definition 2.1 (Subgraph).* A subgraph $M_k$ of graph $G$ is a graph such that $|V(M_k)| = k$ and $V(M_k) \subseteq V(G)$ and $E(M_k) \subseteq E(G)$.

*Definition 2.2 (Induced Subgraph).* An induced subgraph is a subgraph that contains all the edges available in the original graph connecting its vertices. Formally, $\forall (u, v) \in E(M_k) \leftrightarrow (u, v) \in E(G)$.

### 2.2 Network Motif Discovery

As defined by Milo et al.[15], network motifs are patterns of interconnections occurring in complex networks in numbers that are significantly higher than those in similar randomized networks.

For the sake of simplicity we will use motifs and network motifs interchangeably.

Here we will define motifs formally:

*Definition 2.3 (Network Motifs).* An induced subgraph $g_k$ of $G$, is a $k$-node motif if it satisfies the following property:

$$Prob((\frac{1}{n}\sum_{i=1}^{n} F_{R_i}(g_k)) > F_G(g_k)) \leq p \quad (1)$$

The notation $F_G(g_k)$ refers to the frequency of the $k$-node subgraph $g_k$ in $G$, $R$ refers to a random graph similar to $G$, $n$ refers to the number of random graphs required for the comparison, and $p$ refers to the "p-value" threshold.

Now we will define set $S_k$ as the set of all $k$-node subgraphs $g_k$ that are available in $G$. In order to find the motif set "$M_k$" of all the motifs in $G$, we need to apply Eq. 1 on all $g_k$ in $S_k$. To apply the Eq. 1 we need to define an algorithm to generate $R$ similar $G$ and algorithms to calculate $F_G(g_k)$.

Based on the above definition, Milo et al. [15] also provided an algorithm to discover motifs from the network. The steps of the algorithm is summarized in the algorithm below:

**Algorithm 1** Network Motif Discovery

**Require:** $G(V, E)$, $k$
1: $F_G \leftarrow$ SubgraphCounting($G$, $k$)
2: $i \leftarrow 1$
3: **for** $i \leq m$ **do**
4: $\quad R \leftarrow$ RandomGraphGeneration($G$, $n$)
5: $\quad F_R[i] \leftarrow$ SubgraphCounting($R$, $k$)
6: $\quad i \leftarrow i + 1$
7: **end for**
8: $M \leftarrow$ SignificanceTest($F_G$, $F_R$, $p$)

Milo et al.[15] later used the above informal definition into practice and designed an algorithm to make use of this definition and discover network motifs. According to the definition we need to compare frequencies of the subgraphs in our original graph with their frequencies in some random graphs similar to our original graphs. So, a key concept that plays a crucial role to the problem is how we define our random graph generation method. We want to make sure that the motif appearance is unique to this particular network and is not determined by the intrinsic global and local features of the network. So the initial proposal was to preserve all the single-node structural properties, namely the in and out degrees of the nodes. Based on the work done in the Milo et al.'s[15] supplementary material, we can derive a subgraph is called a motif when three things occur. First and the most important metric that need to

occur is the "statistical significance". This can be measured by a metric called "p-value". Formally, we want to test the null hypothesis that the subgraphs in the original graph are not over-represented. If the "p-value" calculated for a subgraph is smaller than a predefined threshold (e.g. 0.01), we can reject the null hypothesis and make sure it is statistically significant and over-represented. There can be different ways to calculate this p-value, Milo et al.[15] proposed to calculate a metric called "z-score" and using the assumption that the subgraph frequencies follow a normal distribution within the random network space, we can use a pre-calculated table (e.g. t-table) to derive the p-values. Let $f_{org}(M_k)$ be the frequency of $M_k$ in the original graph and $f_{rnd}(M_k)$ be the frequency of $M_k$ in the random graph. We can use the formula below to calculate the "z-score".

$$z\text{-}score(M_k) = \frac{f_{org}(M_k) - \bar{f}_{rnd}(M_k)}{std(f_{rnd}(M_k))} \tag{2}$$

There are two other metrics that will help us to further support the significance. Those two metrics are called "minimum frequency" and "minimum deviation". We will formally define these metrics in the following definition:

*Definition 2.4 (Network Motifs - heuristic criteria).* In graph $G$, a network motif $M_k$ is an induced $k$-node subgraph constrained parameter $p$, which is the p-value threshold:

(1) $Prob(\bar{f}_{rnd}(M_k) > f_{org}(M_k)) \leq p$

There are two other bonus parameters that can further filter our motif candidates set and help to pick even more significant motifs:

(1) $f_{org}(M_k) \geq u$

(2) $f_{org}(M_k) - \bar{f}_{rnd}(M_k) > d \times \bar{f}_{rnd}(M_k)$

Milo et al. [15] set the parameters $\{p, u, d, n\}$ to {0.01, 4, 0.1, 1000}.($n$ denotes the number of random graphs required for comparison) However, the values of the parameters are set depending on the different motifs we want and the network we're working with.

According to the above algorithm, the first step to an efficient and accurate motif discovery is to find the subgraph frequencies efficiently and accurately on both the original graph and the large set of random graphs. Here, we will formally define the "Subgraph Counting" problem which is also sometimes referenced as the Subgraph Enumeration problem in the literature.

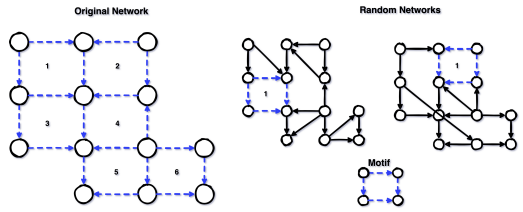



**Figure 2: Motif Discovery example.**

Now we can show the complexity of the MD problem as below:

$$O(\overbrace{n \times q}^{RG\ generation} + \underbrace{f_k(G) \times (n+1)}_{Counting\ subgraphs\ on\ RGs})$$ (3)

Where:

- $n$: is the number of random graphs required.
- $q$: is the number of switches required to generate a random graph.
- $f_k(G)$: is the subgraph counting algorithm used to count on the original and random graphs.

In what follows, we will show that $q$ and $n$ grow exponentially with the size of graph and there are no theoretical bounds provided for these parameters. Another important point to mention is that the problem of counting k-node subgraphs is very challenging itself and performing it for and unbounded $n+1$ times is not tractable.

Here we will explain the MD components in details:

*Definition 2.5 (Subgraph Counting).* Given a set $\Psi$ of non-isomorphic subgraphs and a graph $G$, find the frequency of all the induced instances of the subgraphs $G_s \in \Psi$ in $G$. Subgraph occurrences that have at least one edge or node that they do not share are considered different. Terms *subgraph census* and *subgraph enumeration* are also used for this problem. In other words, one wants to count all the occurrences of all the subgraphs of a given size. So the inputs are the graph $G$ and the subgraph size $k$ and the output is a set of every k-node subgraph frequency available in $G$[20]. This is shown as $f_k(G)$ in Equation 3.

There are two main challenges with the exact subgraph counting algorithms, the first is that the subgraph frequencies grow exponentially to the graph size (number of nodes and edges in $G$) and exponentially to $k$, this makes it inefficient to in order to get the subgraph counts, we check every instance of that subgraph one by one. The second challenge is the "Graph Isomorphism" problem which we will discuss below:

*Definition 2.6 (Graph Isomorphism).* A mapping of a graph is a bijection where each vertex is assigned a value. Two graphs $G$ and $H$ are said to be isomorphic, denoted as $G \sim H$, if there is a one-to-one mapping between the vertices of both graphs where two vertices of $G$ share an edge if and only if their corresponding vertices in $H$ also share an edge. The set of isomorphisms of a graph into itself is called the group of *automorphisms* and is denoted as $Aut(G)$. Two vertices are said to be equivalent when there exist some *automorphisms* that maps one vertex into the other. This equivalence relation partitions the vertices of a graph $G$ into equivalence classes.[21] This problem is known to be an NP-Complete problem.[2]

The main bottleneck of the exact subgraph counting algorithms is the graph isomorphism which is the process of classification of every subgraph instance to its isomorphic group. This is the reason why most of the state-of-the-art motif discovery algorithms has based their solutions on reducing this isomorphism checks.

The next step of the network motif discovery algorithms is to generate an ensemble of random graphs as a null model in order to test the hypothesis that a subgraph in the original graph is over-represented or not. The idea proposed by Milo et al.[15] was to check whether these higher-order structures (e.g. triangles) appear only because of the lower-order structures (e.g. node-level features) or there exist other factors such as the network domain and etc. that lead to these over-represented higher-order structures. To test this, we need to generate random graphs that maintain all single-node

properties, namely in and out degrees. We will discuss the details of the random graph generation process below:

## 2.3 Random Graph Generation

In the original definition of MD[15], random graphs that have the same node degree sequence as the original graph are used to be the null model. There are several algorithms to generate these random graphs, but as suggested by Milo et al.[16] the most practical algorithm is the "Switching Method". This method is the most widely used method in the Motif Discovery literature.

*Definition 2.7 (Degree Sequence).* In graph theory, the list of degrees for each vertex in a graph is called the degree sequence. The degrees are typically listed from highest degree to shortest degree in non-increasing order. The definition we use in this paper is an extension of this definition. In this paper, by **keeping the degree sequence**, we mean that the graphs adjacency matrix can be manipulated during random graph generation process iff the sums of the rows and columns in the adjacency matrix is kept.

*Definition 2.8 (Single Switch).* We define this function as $\xi_{e_1,e_2} : G(V,E) \rightarrow G'(V,E')$ in a way that it draws two random edges from $E(G)$ as $\{e_1 = (u_1,v_1), e_2 = (u_2,v_2)\}$ and outputs $\{e'_1 = (u_1,v_2), e'_2 = (u_2,v_1)\}$ iff $\{(u_1 \neq v_2), (u_2 \neq v_1)\}$ to avoid self-loops, and $\{(u_1,v_2) \notin E(G), (u_2,v_1) \notin E(G)\}$ to avoid parallel edges. This function ensures $V(G) = V(G')$ and the in-degree and out-degree of every vertex are the same in both $G$ and $G'$.Fig(3a) graphically describes how a single switch works, and Fig(3b) shows that the degree sequence is kept after every single switch.

*Definition 2.9 (Switching Method).* The method starts from the original graph ($G$) and performs a series of Monte Carlo swapping steps in which, a pair of edges $(A \rightarrow B, C \rightarrow D)$ is drawn randomly from $G$ and a switch is performed to get $(A \rightarrow D, C \rightarrow B)$. However, the exchange is only performed if it does not generate self-loops nor multiple edges; otherwise the switch is dismissed. The entire process is performed $Q \times |V(E)|$ where $Q$ is chosen large enough that the Markov Chain shows good mixing. Milo et al.[16] empirically found that $Q = 100$ is more than adequate. Fig 3 shows how one switch is done.

Now that we have a clear definition of Network Motif Discovery and its underlying algorithm, we can summarize the challenges that the state-of-the-art solutions are facing:

*Theoretical Challenges* — There are so many parameters used as the input of the NMD algorithms, but there is no theoretical discussions on how we can set these parameters. For example, Milo et al. [15] suggested that q=100 is more than adequate or n=1000 is enough for most cases, but these results were suggested empirically and they used very small networks to back their experiments.

*Scalability and Efficiency* — As discussed earlier, the current NMD framework highly relies on Subgraph Counting and even discovering motifs on a single graph requires thousands of times of Subgraph Counting, which is known to be an NP-Complete problem. Almost all of the state-of-the-art solutions focused on finding a better counting solution, but is there a way to increase the performance by skipping these subgraph counting calls?

| Symbol | Definition |
|---|---|
| $G(V,E)$ | Original Graph, $V$ refers to the vertex set and $E$ is the edge set. |
| $M_k$ | Subgraph of size $k$ |
| $Q$ | A coefficient chosen by domain experts that controls the number of switches needed to generate the random graphs |
| $p$ | the $p_{value}$ that controls the significance level of motifs |
| $u$ | the minimum frequency of a subgraph that becomes a motif candidate |
| $d$ | the minimum distance ratio between the subgraph counts in the original graph vs the average counts in the random graphs |
| $G'_i$ | The Random Graphs generated using the Switching algorithm, also denotes the SSN nodes. |
| $n$ | The number of Random Graphs needed for comparison (Sample Size). |
| $\xi_{e_1,e_2}(G)$ | The Single Switch function: Gets a graph as its input and performs a Single Switch on that graph. |
| $M(\Sigma,P)$ | Switched State Network: $M$ <br> $\Sigma$ refers to the state space. <br> $P$ is the transition matrix of the chain. |

**Table 1: The symbols and notations table.**

# 3 MODESA

Based on the results and conclusions of the previous section, we've shown that even with the fast and efficient subgraph counting algorithms, the motif discovery algorithms will have scalability and efficiency challenges because the motif discovery framework requires many random graphs to test the subgraph's significance.

In this section we are aiming to design a new framework to discover high quality motifs efficiently.

We will first define a Markov Chain which we will refer to as "**S**witched **S**tate **N**etwork"(SSN). Then we will use SSN to revisit the current MD framework and design our new framework.

## 3.1 Motif Discovery as significance testing

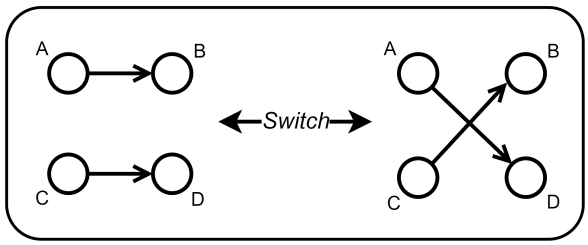
As mentioned earlier, MD's ultimate goal is to test the significance of a subgraph. In other words, we want to reject the null hypothesis that the subgraph's over-representation is random and can happen to every random graph with the same lower-order structural features (same degree sequence in our case). We need to calculate a $p$-value to complete the test. In order to do so, we need to sample from space $\Sigma$ of all the random graphs with the same degree sequence as the original graph $G$. For high quality significance testing, we need high quality samples, and by high quality samples we mean that the probability of drawing each sample from $\Sigma$ should be equal to $\frac{1}{|\Sigma|}$.

It has been shown in [16] that one of the most efficient and effective ways to sample from $\Sigma$ is to use Markov Chains and the Switching Method.

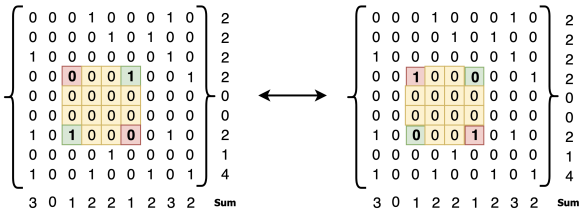
The Markov Chain that can be used for this sampling task, should have the following properties:

(1) Draw correct samples from $\Sigma$.
(2) Converge to a stationary distribution $\vec{\pi}$ after long enough mixing.
(3) $\vec{\pi}$ should be a uniform distribution with $\frac{1}{|\Sigma|}$ probability.

Here, we will first define a Markov Chain which we will refer to as "**S**witched **S**tate **N**etwork"(SSN). Then we will use SSN to revisit the current MD framework and design our new framework.




(a) The graphical view of switching.



(b) The Matrix view of switching. It shows how row and column sums are kept after a single switch is performed.

**Figure 3: This figure shows how we can generate a random graph based on an input graph, without changing the input's degree sequence.**

## 3.2 Switched State Network (SSN)

We want to build a Markov chain $M$ with transition probability matrix $P$ on the state space $\Sigma$, the space of all the possible random graphs that have the same degree sequence as $G$ (our original graph). To enumerate this state space, we can use the switching method. Note that this state space is enormously large even for graphs with few thousand nodes and edges.

For a state $G_i(V_i, E_i)$ in the chain, we define set $S$ as:

$$S = \{(e1, e2) \mid \forall (e1, e2) \in E_i \times E_i, e1 \neq e2\}$$

This will be the set of all possible switches between every two edges in the current state. The size of this set will be $\binom{|E_i|}{2}=\binom{|E|}{2}$. Now we will define $S_{val} \subseteq S$ as the set of all the "valid switches" and $S_{inv} = S - S_{val}$ as the set of all "'invalid switches".

Using the above sets we can now define the transition matrix $P_{|\Sigma|\times|\Sigma|}$ as below:

$$P_{|\Sigma|\times|\Sigma|} = \begin{cases} P_{ij} = 1 - \frac{|S_{val}|}{\binom{|E_i|}{2}} & i = j \\ P_{ij} = \frac{1}{\binom{|E_i|}{2}} & \textit{exists a valid switch between states } i \,\&\, j \\ P_{ij} = 0 & \textit{otherwise} \end{cases}$$

We can use the steps below to define transitions on this state space to define the Markov chain:

(1) From the current state $G^*(V^*, E^*)$, determine the set $S$ of all the edge pairs $(e1, e2) \in E^*$ that $e1 \neq e2$. Note that the size of this set is always $\binom{|E^*|}{2}$.
(2) From $S$, choose one pair $(e1_0, e2_0)$ uniformly at random.
(3) Apply the single switch on $G^*$ using $(e1_0, e2_0)$ if the single switch is valid and stay in the current state otherwise.

Note that in our case, the initial state is always our original graph $G(V, E)$. Fig(4) illustrates a toy SSN example. Defined as above, we can now discuss the properties of SSN.

Theorem 3.1. *Let*

Property 3.1. ***Aperiodicity*** *SSN is aperiodic because every state contains a self-loop.*

Property 3.2. ***Reducibility*** *SSN is irreducible because the chain is fully connected.*

The properties above prove that SSN will converge to a stationary distribution $\vec{\pi}$.

Moreover, the transition probability between any two states is either $\binom{|E|}{2}$ if these two states are a single switch away or 0 otherwise. So $P_{ij} = P_{ji}$ holds for every $i$ and $j$.

Property 3.3. ***Uniform Stationary*** *SSN has a uniform stationary distribution because the symmetry of the transition probability matrix (P) proves that* $\vec{\pi} = \{\frac{1}{|\Sigma|}, \frac{1}{|\Sigma|}, \ldots, \frac{1}{|\Sigma|}\}$.

This is a very important property that shows if SSN reach its stationary distribution, sampling from it is fair because all the states has the same probability of being drawn.

Property 3.4. ***Reversibility*** *SSN is reversible because it supports the detailed balance property which is* $\pi_i P_{ij} = \pi_j P_{ji}$.

These properties will help us study the current MD framework and design our MODESA framework.

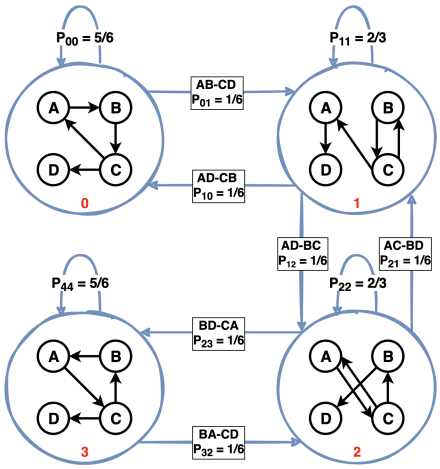
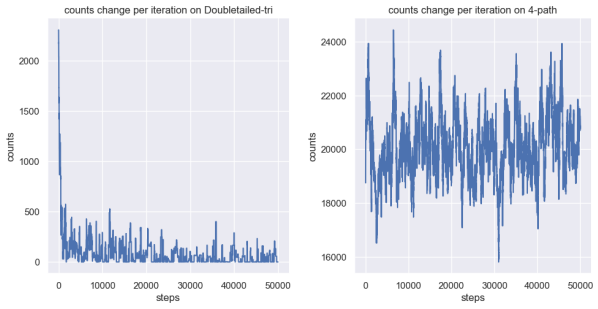



**Figure 4: A toy SSN example.**

## 3.3 Observations and Intuitions

According to the motif definition, intuitively, if a subgraph is a motif, it is very frequent on our original graph and subsequently the subgraphs that are structurally very close to our motif is usually under-represented. For example in 3-node subgraphs, if a triangles

are very frequent in a graph and are considered as motifs, the other 3-node motif which is a wedge will be less frequent, as we are considering the induced subgraphs. We've observed that if we start to switch edges in our original graph, we're very likely to break motif structures, because with a high probability, every random edge we pick participates in one or more motifs and switching those edges will reduce the motif counts. So, if we keep doing this switching process we might see a significant subgraph count drop on the motifs. Fig 5 shows the different trends that happened in our experiments.




**(a) Decreasing Trend** **(b) No Special Trend**

**Figure 5: We can see the above trends when tracking the counts over the incremental steps. Part (5a) shows a decreasing trend which means the counts stabilized with frequencies all lower than the original count, which can be interpreted as a motif. Part (5b) shows a wavy trend that although the starting frequency of the subgraph is very high, it does not show any significance which can't be interpreted as a motif.**

## 3.4 Assessing significance in a Markov chain without mixing

In this part, we will design a framework that can detect motifs based on SSN using random walk without mixing. We will use the $\sqrt{\varepsilon}$-test first introduced by Chikina et al [4]. This test was first applied to the problem of detecting bias in Congressional districting, to test whether the initial state of a Markov chain is an outlier or not.

Theorem 3.2. *Let $M = X_0, X_1, \ldots$ be a reversible Markov chain with a stationary distribution $\pi$, and suppose the states of $M$ have real-valued labels with a label function $\omega : \Sigma \to \mathbb{R}$. If $X_0 \sim \pi$, then for any fixed $t$, the probability that the label of $X_0$ is an $\varepsilon$-outlier from among the list of labels observed in the trajectory $X_0, X_1, X_2, \ldots, X_t$ is, at most, $\sqrt{2\varepsilon}$.*

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

This theorem has proven that if one is sampling from a reversible Markov chain to test the significance of a state against the null hypothesis that the state is drawn from the chain's stationary distribution, they don't need to wait until the mixing is achieved. It is sufficient to test the significance of the initial state against the neighbours if they have enough neighbours. They have provided a formula to rigorously calculate the desired p-value as follows:

$$\rho^t_{0,l} \leq \sqrt{\frac{2l+1}{t+1}} \tag{4}$$

We let $\pi$ denote any stationary distribution for $M$ and suppose that the initial state $X_0$ is distributed as $X_0 \sim \vec{\pi}$, so that in fact, $X_i \sim \vec{\pi}$ for all $i$. We say $\sigma_j$ is $l$-small among $\sigma_0, \ldots, \sigma_t$ if there are, at most, $l$ indices $i \neq j$ among $0, \ldots, t$, such that the label of $\sigma_i$ is, at most, the label of $\sigma_j$. So, For $0 \leq j \leq t$, we define:

$$\rho^t_{j,l} := Pr(X_j \text{ is } l\text{-small among } X_0, \ldots, X_t)$$

Theorem 3.2 provides a test to detect if a presented state of a reversible Markov chain was indeed selected from a stationary distribution.

Corollary 3.3. *We can now instead of the traditional MCMC sampling for the significance testing, use the theorem 3.2.*

*Motif Discovery framework using the $\sqrt{\varepsilon}$-test will be as follows:*

1. *Count subgraphs in the original graph.*
2. *Set the counter to 0.*
3. *Switch once.*
4. *Count the subgraphs in the new random graph.*
5. *Increase the counter by 1.*
6. *Go back to step (3) if counter $\leq t$.*
7. *Calculate the p-value for every subgraph using the formula [4].*
8. *Report the significant subgraphs as motifs.*

The above framework solves the problem of mixing on the chain and provides guarantees on the provided p-value based on how large we choose the parameter $t$, it also helps us reduce the input parameters for the MD problem.

## 4 ALGORITHM

In the previous section, we proposed a way to first, decrease the number of input parameters of the algorithm, which reduces the user effort to tune those parameters for results with better quality, and second, we can rigorously calculate $p$-value for every subgraph in the graph which was not possible due to the unknown mixing time (number of switches) for the chain and unknown number of samples(random graphs) required.

The question that arises here is that how these improvements affect our efficiency? In most cases, there is a trade-off between accuracy and efficiency and if one is increased, the other will decrease. But in our case, based on the new framework, we have designed algorithms that can efficiently find the motifs with up to 4-orders of magnitude faster than the state-of-the-art solutions.

There are few ways to optimize the step (4) of 3.3, the trivial way is to call the Motif Counting algorithm on the new random graph and count the subgraphs. But is it the most efficient way of extracting the subgraph frequencies?

There are two main intuitions behind our algorithm, first is that we know the frequencies of the subgraphs in the previous steps, can we use this information to our advantage to decrease the counting time? second, after every switch, only the motif instances within a small area around the neighbourhoods of the switched edges will be affected and this will not affect the subgraph counts far away from the action area.

Using the above intuitions, we have designed two algorithms, a version that can can be applied on both directed and undirected graphs with any motif size, which sacrifices efficiency for the generality of the inputs, and a customized version that supports up to 5-node undirected subgraphs, we call the former Track & Count and the later Fast-Track & Count.

As discussed before, Motif Discovery is closely associated with the Subgraph Counting algorithms as the subgraph frequencies are the main criterion for the discovery task. So, a good Motif Discovery algorithm requires a good counting algorithm.

## 4.1 Track & Count

The input of the Track & Count algorithm is a graph $G$, and the k-node subgraph frequencies of that Graph. This algorithm first picks two random edges from $G$, applies a single switch to $G$, updates the k-node subgraph frequencies based on the switch, and returns $G'$ and the new frequency set.

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

*4.1.1 Edge Selection.* At this step, the algorithm selects two edges at random from the edges set ($E$), to perform switching. We need to keep in mind that some edge selections will result into self-loops or parallel edges that may be illegal in most domains, so we will skip those selections and re-select the edges. Figure(6) illustrates how invalid switches can affect the graph.

After the edges are selected for switching, the next step is to apply the switch! We can break it into two hidden steps: First, remove the old edges, and second, add the new edges. In every single switch $\{(a,b),(c,d)\} \rightarrow \{(a,d),(c,b)\}$, we have four atomic actions, two edge deletions(delete $\{(a,b),(c,d)\}$) and two edge additions(add $\{(a,d),(c,b)\}$). On every atomic action, one edge is involved.

*4.1.2 Track Deletion.* The basic trivial observation here is that if an edge is removed, only the frequency of the subgraphs that contain that edge will be affected and the frequency of the other subgraphs will not change. Using the above assumption, we need to first find the k-node subgraphs that contain the edge which is going to be removed. Every edge has 2 nodes associated with it, so to count the k-node subgraphs around an edge, we need to perform a search (e.g., BFS, DFS) on the (k-2)-node neighbourhood of that edge.

The inputs for this algorithm are the graph $G(V,E)$, edge ($e \in E$) (the edge that is going to be removed), $k$ which is the subgraph size, and $F_G$ the frequencies of the k-node subgraphs in $G$.

The algorithm returns $G'(V, E' = E - \{e\})$ and $F_{G'}$ the updated k-node subgraph frequencies after the edge removal.

Algorithm 2 shows the steps of Edge Deletion Tracking.

The SubgraphSearch function, performs a Breadth-First Search(BFS) around the edge, with the (k-2) depth and finds all the subgraphs. The IsomorohismCheck relabels the subgraph to its unique isomorphic labelling. Then for every subgraph, we know that its count will decrease as the edge is being removed. We need to check after this edge removal, the subgraph turns into another subgraph or not, if the subgraph stays connected after the removal of the edge, it means it changed to another subgraph so we have to increase the frequency of the new subgraph, if not, we don't need to update any other frequencies.

**Algorithm 2** Track Deletion

**Require:** $G(V,E), e \in E, F_G, k$
**Ensure:** $G'(V, E-\{e\}), F_{G'}$
1: $D \leftarrow SubgraphSearch(G, k-2, e)$
2: **for** $d \in D$ **do**
3: $d' \leftarrow IsomorphismCheck(d)$
4: $F_G(d') \leftarrow F_G(d') - 1$
5: $DeleteEdge(G, e)$
6: **if** Connected($d'$) **then**
7: $F_G(d') \leftarrow F_G(d') + 1$
8: **end if**
9: $AddEdge(G, e)$
10: **end for**
11: $DeleteEdge(G, e)$

*4.1.3 Track Addition.* To track changes after edge addition (assume we want to add $e = (u, v)$), we need to first add $e$ to $G$ then we will find the set of subgraphs that contain $e$ we call this set $D$ , then for every subgraph $d$ in $D$ we increase the frequency of $d$, then we remove $e$ from $d$. If $d$ stays connected after edge removal, we decrease its frequency in $T$. In the end, we should add $e$ to $G$ again.

## 4.2 Fast-Track & Count

The previous algorithm can support both directed and undirected graphs and it has no limits on $k$ (subgraph size). The bottleneck of the previous algorithm is that the search space of every step exponentially grows with $k$. In this section we've designed an algorithm for undirected graphs, that can track the frequencies of up to 5-node subgraphs around the edge such that we don't need to enumerate every motif instance one by one and we can use formulas to track these changes instead. This also helps with the graph isomorphism problem since we don't need to classify every subgraph instance and we can derive counts for every class using pre-calculated formulas based on the smaller subgraphs.

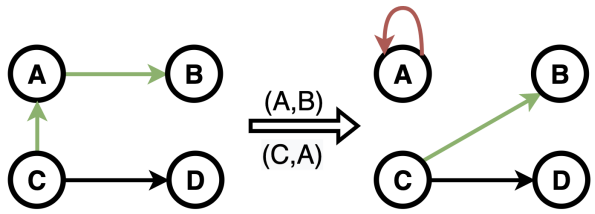
The main idea here is that every large subgraph consists of smaller building blocks which are the smaller subgraphs, so if we have the counts of the smaller subgraphs, we can use some formula to calculate the frequencies of larger subgraphs. Note that these formulas are now limited for up to 5-nodes and with the advancement of these counting formulas, our algorithm can be extended to support larger subgraphs as well.

In this section, we will first review the formulas for calculating the frequencies of 4-node subgraphs, then we will extend them to support the dynamic nature of tracking.

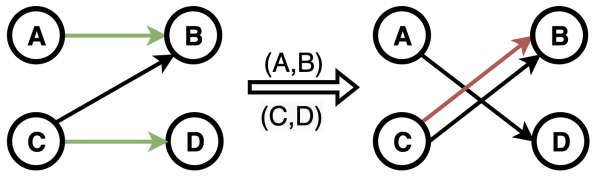
*4.2.1 Frequency Calculation for 4-node Subgraphs.* The formulas in this section are proven in [1, 7, 18].

Below you can find a list of 4-node motifs with their calculation formula:

(1) # 3-Star = $\sum_i^V \binom{d(i)}{3}$
(2) # Diamonds = $\sum_e^E \binom{t(e)}{2}$
(3) # 3-Paths = $\sum_{(i,j)}^E (d(i)-1)(d(j)-1) - 3T(G)$
(4) # Tailed-Triangles = $\sum_i^V t(i)(d(i)-2)$
(5) # 4-Cycles = $\sum_{i>j}^E \binom{W_{++}(i,j)+W_{+-}(i,j)}{2}$




**(a) This edge selection and switching creates a self-loop and considered invalid.**




**(b) This edge selection and switching creates parallel edges and considered invalid.**

**Figure 6: This figure illustrates the cases that invalid switches may happen.**

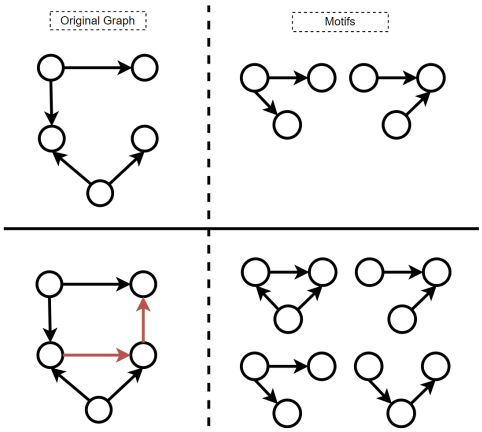



**Figure 7: How the motifs react to the edge addition and deletion.**

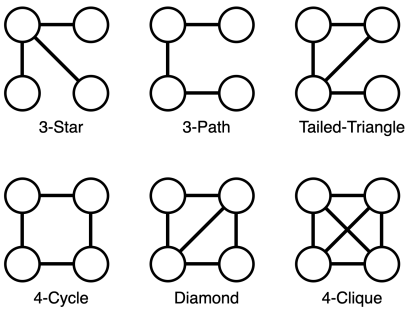



**Figure 8: List of all 4-node undirected subgraphs that we refer to in this paper.**

Now we can state the derived formula for tracking the count differences after edge(a,b) 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) $\Delta$ 3-Star = $-\frac{1}{2}((d(a)-1)(d(a)-2)+(d(b)-1)(d(b)-2))$

(2) $\Delta$ Diamonds = $\sum_{e*} 1 - t(e) + \binom{\Delta T}{2}$

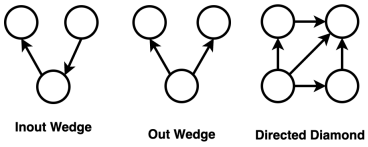



**Figure 9: Different types of directed subgraphs used in the formulas.**

| Symbol | Definition |
|---|---|
| $d(i)$ | Degree of node $i$. |
| $W(G)$ | Wedge. |
| $W_{++}(G), W_{+-}(G)$ | out-wedge, inout-wedge |
| $W_{++}(i,j)$ | out-wedge between $i$, $j$ |
| $W_{+-}(i,j)$ | Wedge from $i$ to $j$ |
| $T(G)$ | Triangle counts in graph $G$ |
| $t(e)$ | Triangle counts around edge $e$ |
| $t(i)$ | Triangle counts around node $i$ |
| $DD(G)$ | Directed Diamond. |

**Table 2: The symbols and notations table for subgraph counting.**

(3) $\Delta$ 3-Paths = $\sum_{i\in(N(a)\cup N(b))} 1-d(i)-(d(a)-1)(d(b)-1)-3\Delta T$

(4) $\Delta$ Tailed-Triangles = $\Delta t(d(a)-2)-t_2(a)+\Delta t(d(b)-2)-t_2(b)+\sum_i^{V*} 2-d(i)$

For the 4-Cycles and 4-Cliques tracking after deletion of an edge, we've designed a framework to do the tracking and the framework is as follows: To find how many 4-cycles were removed after an edge removal, we need to find the neighbours of the each node of that edge (N(a), N(b)) and check how many of the edges in set N(a) are connected to set N(b). For all the 4-cycles found, if we check the existence of two more edges between neighbour of node "a" and node "b", we can also keep track of the 4-cliques.

## 4.3 Time & Space analysis

*4.3.1 Track & Count.* As mentioned before, our algorithm is "plug & play" which means that for every valid subgraph counting algorithm, we can use our framework to discover motifs in the graph. So, to compare our algorithm with the existing Motif Discovery solutions, we will assume that both algorithms are using the same subgraph counting algorithm. Our algorithm reduces the size of the random network that we need to count over drastically compared to the current motif discovery solutions. For comparison simplicity, and because there are no theoretical bounds on the number of random graphs needed for motif discovery, we assume that the number of random networks needed is denoted as $n$ (which should be highly related to the graph size). We will denote the k-subgraph counting time on graph $G_{(V,E)}$ as $f_k(G_{(V,E)})$ and $m$ is the number of samples needed.

$$MODESA : O(f_k(G_{(V,E)}) + m \times f_k(G_{(d_{max})^{k-2},(\frac{d_{max}}{2})^{k-1}})) \quad (5)$$

$$SOTA : O(f_k(G_{(V,E)}) + n \times f_k(G_{(V,E)})) \quad (6)$$

As the real world graphs ratio between the edges and nodes are small, our average degree of each node will be small, so the (k-2)-neighbourhood around every edge will be so much smaller than the full graph. Our algorithm only requires counting on the (k-2)-neighbourhood and because the counting time increases exponentially, our algorithm outperforms the state-of-the-art solutions by orders of magnitude.

$$m \times f_k(G_{(d_{max})^{k-2},(\frac{d_{max}}{2})^{k-1}}) \ll n \times f_k(G_{(V,E)})) \quad (7)$$

Finally the improvement ratio will be:

$$\frac{n}{m} \times \frac{f_k(G_{(V,E)})}{f_k(G_{(d_{max})^{k-2},(\frac{d_{max}}{2})^{k-1}})} \quad (8)$$

The memory(space) required for both Motif Discovery and our approach is the space required to store the original graph size in memory and it is identical between our method and the state-of-the-art.

*4.3.2 Fast Track & Count.* As FTAC is a customized and optimized version of TAC for undirected graphs and we eliminate the local counting phase for tracking, it is worth mentioning that how this will affect the time complexity of the algorithm. As you can see in the FTAC formulas, for example the 3-Star counting can be done in $O(1)$ because access to the node degree is possible in constant time.

For tracking the Diamond counts, we need to go over all the triangles the candidate edge "$e$" for removal/addition involves, and for each "$e^*$" edge on those triangles we need to calculate "$t(e^*)$", these calculations can be done in $O(|t(e)|)$, if for every edge, we store a list of nodes that form a triangle with that edge.

For the 3-Paths we only need the degrees of the one-hop neighbours of the edge, the global triangle count changes, and we can apply our formula in constant time. The global triangle counts are calculated in the previous step (3-node counting) and has been calculated in constant time. One-hop degree gathering would be $O(d_{avg})$ on average and would be $O(d_{max})$ for the worst case scenario.

To track the changes on Tailed-Triangles we need to access to the degree of every node that forms a triangle with our candidate edge. Those nodes are accessible in constant time via the index built over the edges that keeps track of the nodes that form a triangle with that edge and for each of those nodes we need to get the degree which is again accessible in a constant time. So the Tailed-Triangles tracking would also take $O(|t(e)|)$.

For the 4-Cliques, we have to 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 $O(|t(e)|^2)$.

For the 4-Cycles we need to check whether the nodes that are neighbouring our candidate edge and they do not form a triangle with are connected or not. So it will take $O(|N(a) - (N(a) \cap T(e))| * |N(b) - (N(b) \cap T(e))|)$

We can conclude that at every step of deletion/addition the most time consuming 4-node subgraphs(bottlenecks) to track are the 3-paths and the 4-cycles/cliques which will take $O(|t(e)_{max}|^2 + d_{max})$ in the worst-case scenario and $O(|t(e)_{avg}|^2 + d_{avg})$ on average.

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 will be discussed in a separate technical report.

*4.3.3 Space Analysis.* The space used by most of the SOTA MD algorithms can be divided into two parts, first the original graph size and second, the index built to reduce isomorphism checks. These two are the memory used by the motif counting algorithm and adding the discovery part would not add more overhead to these algorithms. As Track and Count is a plug and play algorithm and uses the counting algorithms, its memory usage is also affected by the counting solution. Our algorithm and framework also does not add extra memory usage to the counting algorithm.

# 5 EXPERIMENTS

## 5.1 Experiments Setup

We implemented our algorithms in C++ & Python and ran our experiments on a computer equipped with a 2.3 GHz 8-Core Intel Core i9, and 16GB memory.(can do much better!) We ran MODESA on a large collection of graphs from the Network Repository[23], SNAP[11]. In the first half of the cases, directionality is used and in the other half it is ignored, and duplicate edges and self loops are omitted. Tab. 3 has the properties of all these graphs. We have

| Dataset | $\|V\|$ | $\|E\|$ |
|---|---|---|
| Social | 67 | 182 |
| Electronic | 97 | 189 |
| S. cereviciae | 688 | 1079 |
| E. coli | 672 | 1276 |
| soc-birghtkite | 56.7K | 426K |
| flickr | 244K | 3.64M |
| web-google-dir | 876K | 8.64M |
| orkut | 3.08M | 234M |

**Table 3: These are the datasets used in our experiments.**

chosen these datasets for a couple of reasons. The first four datasets

are the datasets used in the bioinformatics community to test and run experiments. We are using these datasets for two reasons, first is to compare our result's correctness with the baseline solutions, and then to show our advantage in efficiency and scalability compared to the state-of-the-art solutions. The second half of the datasets are the ones mainly used by the data mining community and they are so much larger than the former datasets. We test our algorithm against these datasets to show how efficiently it can scale to the larger graphs. We've compared our solution with the state-of-the-art solutions in the motif discovery domain [8, 10, 14, 21].

## 5.2 Correctness

As we are proposing a new framework with guarantees on the significance results, it is hard to compare our results with the state-of-the-art solutions as the p-values calculated using their approach maybe different from our approach. In this section we are reporting the motifs found using our algorithm, very similar to the existing solutions. The parameters used in the MD solutions are also different from the parameters used in our algorithm, Although it is hard to find a fair setup for comparing the accuracy results, we've compared the motif outputs of the different Motif Discovery algorithms with each other and reported here. The setup for motif discovery algorithms is: dataset: "E.coli", number of random graphs: "10K", switches per random graph: "$3 * |E|$". We've calculated the z-scores based on the mentioned setup and picked top 5 motifs. For MODESA we run the algorithm for 50K steps and we used 0.05 and 0.01 p-values as reference points to test whether a subgraph is a motif or not.

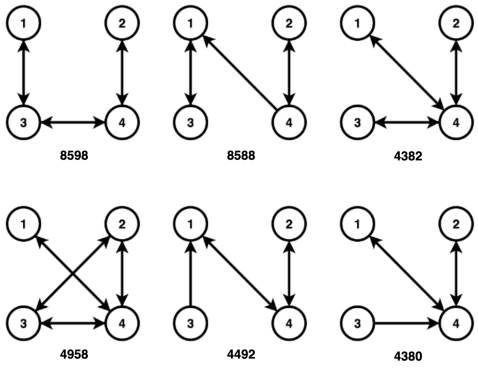
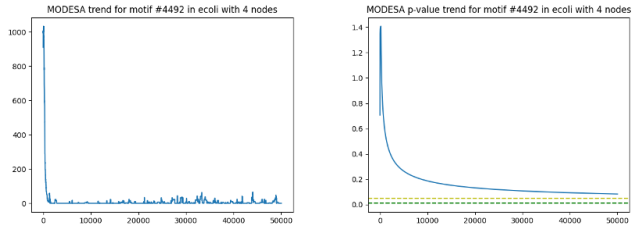



**Figure 10: Different types of motifs found using different algorithms.**

| Subgraph | Kavosh | QX | G-Tries | MODESA |
|---|---|---|---|---|
| 8598 | ✓ | ✓ | ✓ | ✓ |
| 8588 | ✓ | ✓ | ✗ | ✗* |
| 4382 | ✓ | ✓ | ✓ | ✓ |
| 4958 | ✓ | ✓ | ✓ | ✓ |
| 4492 | ✓ | ✓ | ✗ | ✗* |

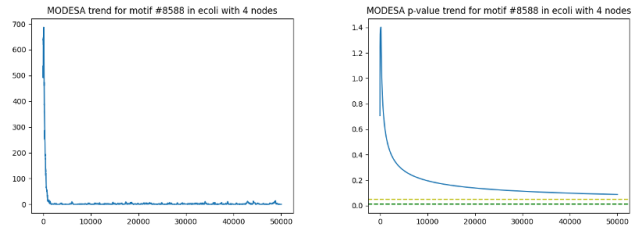
**Table 4: Different motifs found using different algorithms.**

There are two motifs (8588 and 4492) that has the ✗* in Table 6. As it is shown in the below figures the frequency of this subgraph increases first and then starts to decrease. This leads our algorithm to form doubts about this motif at first and this makes our algorithm to need to run longer to verify the significance of these subgraphs.




**(a) Count Trend** **(b) P-value Trend**

**Figure 11: Count and p-value trends for 4492.**




**(a) Count Trend** **(b) P-value Trend**

**Figure 12: Count and p-value trends for 8588.**

## 5.3 Efficiency & Scalability

In this section, for the efficiency part we are going to compare our results with the state-of-the-art Motif Discovery solutions such as Kavosh, G-Tries, QuateXelero, and ACC-Motif and we are going to show we can achieve results so much faster.

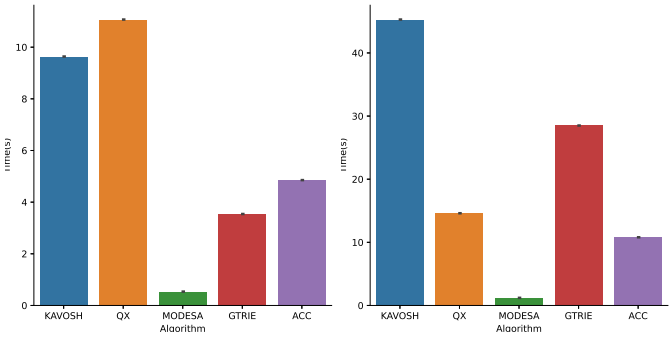
We have performed our solution to 3,4,5 node motifs and we have compared our results our solutions with state-of-the-art solutions on small datasets because those solutions can't scale to the large datasets.

Here, is a table summarising the experiments results on small datasets.

| Dataset | FTAC | QX | Kavosh | G-Tries | ACC-Motif | Ratio |
|---|---|---|---|---|---|---|
| Social | **3.19** | **47.68** | 56.59 | 57.58 | 84.15 | 14.94X |
| Electronic | **2.87** | 97.66 | 68.22 | 51.23 | **51.19** | 17.83X |
| S. cereviciae | **3.99** | 409.12 | 7897.31 | 359.96 | **160.66** | 40.26X |
| E. coli | **2.85** | 283.76 | 390.19 | 319.08 | **101.04** | 35.57X |
| YeastPPI | **5.58** | 5191.85 | 6919.22 | 4796.93 | **1852.23** | 332X |

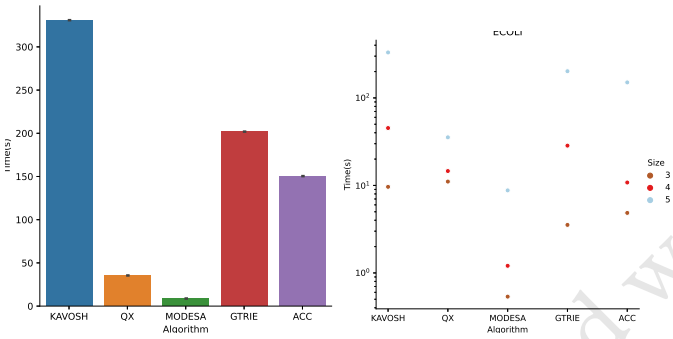
**Table 5: Comparison between MODESA and SOTA**

For the scalability, we are taking a step further and run our solutions on Million-Scale datasets to show that not only our algorithm




(a) 3-node (b) 4-node

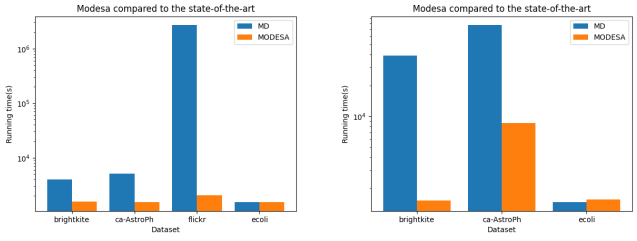
**Figure 13: The efficiency comparison on Ecoli dataset on 3 and 4 node subgraphs.**




(a) 5-node (b) Scalability comparison

**Figure 14: 5-node and scalability comparison on Ecoli dataset**

is efficient, but it also can scale well when it comes to real world large graphs.




(a) 4-node (b) 5-node

**Figure 15: Scalability test against large datasets (charts are in log scale).**

## 5.4 Case Studies

We believe that motifs can be helpful in many downstream tasks such as link prediction or community detection. In this section, we've taken one step further and tested our motif findings in the downstream tasks to show how high quality motifs can improve the downstream tasks.

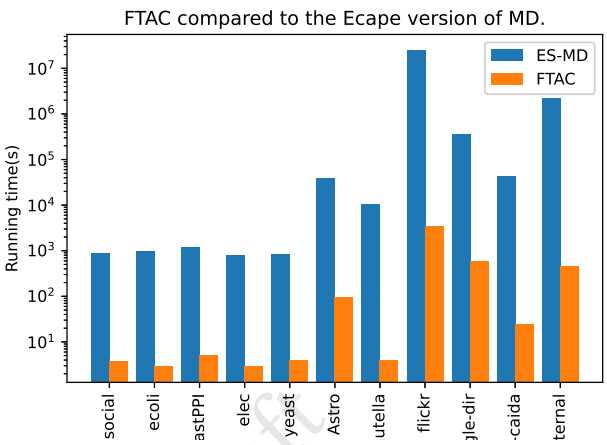



**Figure 16: Comparison of the FTAC with the implementation of motif discovery using ESCAPE.**

| Subgraph | Original Count | Random Count | Motif | AUC | Ratio | Size |
|---|---|---|---|---|---|---|
| M2 | 2400 | 2700 | ✗ | 0.5195 | 0.125 | 3 |
| M3 | 7100 | 7250 | ✗ | 0.492 | 0.02113 | 4 |
| M4 | 5100 | 5700 | ✗ | 0.5235 | 0.11765 | 4 |
| M5 | 58 | 3 | ✓ | **0.8255** | 0.94828 | 3 |
| M6 | 14000 | 16400 | ✗ | 0.6235 | 0.17143 | 5 |
| M7 | 19000 | 20000 | ✗ | 0.52 | 0.05263 | 5 |
| M9 | 650 | 55 | ✓ | 0.6785 | 0.91538 | 4 |
| M10 | 57 | 15 | ✓ | **0.8725** | 0.73684 | 4 |
| M15 | 870 | 300 | ✓ | 0.5755 | 0.65517 | 5 |
| M14 | 2700 | 120 | ✓ | 0.65 | 0.95556 | 5 |

**Table 6: Motif Discovery Effectiveness on Link Prediction**

## 6 RELATED WORKS

The idea of Network Motif Discovery was first introduced by Holland and Leinhardt[5] in a work that introduced the concept of triangle(triad) census in the networks and they also introduced the idea of counting different types of subgraphs and testing their significance against the random networks. The idea was then generalized and popularized by Milo et al. [15] in 2002. They have found motifs in the gene regulation (transcription) network of the bacteria E. coli and many other natural networks. Since then, a vast amount of studies have been conducted on this subject. There are two main lines of research in this area, some of the papers mainly focus on the biological applications of the network motifs, while others focus on the computational theory of network motifs.

As mentioned earlier, the main bottleneck to the motif counting problem is the graph isomorphism problem and almost all of the algorithms provided in the literature try to provide a more efficient counting algorithm that enumerates the subgraphs faster than the others. We will introduce the most famous and most used algorithms in the literature. The first algorithm in the literature is from Kashtan et al. and it is called mfinder [9]. This algorithm is based

on edge sampling and estimates concentrations of induced subgraphs in both directed and undirected networks. This algorithm does not provide any analysis on the classification time of sampled subgraphs that requires solving the graph isomorphism problem for each subgraph sample and the algorithm may sample same subgraphs over and over wasting time, it also only supports up to 6 node motifs and it only reports the most significant motif(only one motif for every network). Wernicke later solved the biased sampling problem of mfinder and also designed a way to sample every subgraph only once and packed his result in an algorithm called FANMOD[26](RAND-ESU).

## ACKNOWLEDGMENTS

This work was supported by the [...] Research Fund of [...] (Number [...]). Additional funding was provided by [...] and [...]. We also thank [...] for contributing [...].

## REFERENCES

[1] Nesreen Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick G. Duffield. 2015. Efficient Graphlet Counting for Large Networks. *2015 IEEE International Conference on Data Mining* (2015), 1–10.

[2] Vikraman Arvind and Johannes Köbler. 2000. Graph Isomorphism Is Low for ZPP(NP) and Other Lowness Results. In *STACS 2000*, Horst Reichel and Sophie Tison (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 431–442.

[3] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[4] Maria Chikina, Alan Frieze, and Wesley Pegden. 2017. Assessing significance in a Markov chain without mixing. *Proceedings of the National Academy of Sciences* 114, 11 (2017), 2860–2864. https://doi.org/10.1073/pnas.1617540114 arXiv:https://www.pnas.org/content/114/11/2860.full.pdf

[5] Paul W. Holland and Samuel Leinhardt. 1976. Local Structure in Social Networks. *Sociological Methodology* 7 (1976), 1–45. http://www.jstor.org/stable/270703

[6] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. ACM, 1311–1322.

[7] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 495–505. https://doi.org/10.1145/2736277.2741101

[8] Zahra Razaghi Moghadam Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. 2009. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics* 10, 1 (2009), 1–12.

[9] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. 2004. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20, 11 (03 2004), 1746–1758. https://doi.org/10.1093/bioinformatics/bth163 arXiv:https://academic.oup.com/bioinformatics/article-pdf/20/11/1746/851183/bth163.pdf

[10] Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter, Ina Koch, and Ali Masoudi-Nejad. 2013. QuateXelero: An Accelerated Exact Network Motif Detection Algorithm. *PLOS ONE* 8, 7 (07 2013), 1–15. https://doi.org/10.1371/journal.pone.0068073

[11] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

[12] Xiaodong Li, Tsz Nam Chan, Reynold Cheng, Kevin Chang, Caihua Shan, and Chenhao Ma. 2019. 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). https://www.cs.hku.hk/data/techreps/document/TR-2019-04.pdf

[13] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiao-Li Li. 2015. Network motif discovery: A GPU approach. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE. https://doi.org/10.1109/icde.2015.7113337

[14] Luis A. A. Meira, Vinícius R. Máximo, Álvaro L. Fazenda, and Arlindo F. da Conceição. 2014. acc-Motif: Accelerated Network Motif Detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11, 5 (2014), 853–862. https://doi.org/10.1109/TCBB.2014.2321150

[15] R. Milo. 2002. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (Oct. 2002), 824–827. https://doi.org/10.1126/science.298.5594.824

[16] R. Milo, N. Kashtan, S. Itzkovitz, M. Newman, and U. Alon. 2003. On the uniform generation of random graphs with prescribed degree sequences. *arXiv: Statistical Mechanics* (2003).

[17] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[18] Ali Pinar, C. Seshadhri, and V. Vishal. 2016. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. (10 2016).

[19] Nataša Pržulj and Noël Malod-Dognin. 2016. Network analytics in the age of big data. *Science* 353, 6295 (2016), 123–124.

[20] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. 2021. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Comput. Surv.* 54, 2, Article 28 (mar 2021), 36 pages. https://doi.org/10.1145/3433652

[21] Pedro Ribeiro and Fernando Silva. 2010. G-tries: An efficient data structure for discovering network motifs. *Proceedings of the ACM Symposium on Applied Computing*, 1559–1566. https://doi.org/10.1145/1774088.1774422

[22] Pedro Manuel Pinto Ribeiro and Fernando M. A. Silva. 2014. G-Tries: a data structure for storing and finding subgraphs. *Data Min. Knowl. Discov.* 28, 2 (2014), 337–377. https://doi.org/10.1007/s10618-013-0303-4

[23] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[24] Lewi Stone, Daniel Simberloff, and Yael Artzy-Randrup. 2019. Network motifs and their origins. *PLOS Computational Biology* 15 (04 2019).

[25] Charalampos E Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. 2017. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*. 1451–1460.

[26] Sebastian Wernicke. 2006. Efficient Detection of Network Motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3, 4 (2006), 347–359. https://doi.org/10.1109/TCBB.2006.51



# A RESEARCH METHODS

## A.1 Part One

Formal definition of the algorithm can be found in Algorithm 3.

**Algorithm 3** Track Addition

**Require:** $G(V, E), e \in E, F_G, k$
**Ensure:** $G'(V, E \cup \{e\}), F_{G'}$
  $AddEdge(G, e)$
  $D \leftarrow IsomorphismCheck(G, k-2, e)$
  **for** $d \in D$ **do**
    $d' \leftarrow IsomorphismCheck(d)$
    $T[d'] \leftarrow T[d'] + 1$
    $DeleteEdge(G, e)$
    **if** Connected($d'$) **then**
      $T[d'] \leftarrow T[d'] - 1$
    **end if**
    $AddEdge(G, e)$
  **end for**

## A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

# B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit

congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009