Sampling Methods for Counting Temporal Motifs

Paul Liu
Stanford University
Stanford, California
paul.liu@stanford.edu

Austin R. Benson
Cornell University
Ithaca, New York
arb@cs.cornell.edu

Moses Charikar
Stanford University
Stanford, California
moses@cs.stanford.edu

ABSTRACT

Pattern counting in graphs is fundamental to several network science tasks, and there is an abundance of scalable methods for estimating counts of small patterns, often called motifs, in large graphs. However, modern graph datasets now contain richer structure, and incorporating temporal information in particular has become a key part of network analysis. Consequently, temporal motifs, which are generalizations of small subgraph patterns that incorporate temporal ordering on edges, are an emerging part of the network analysis toolbox. However, there are no algorithms for fast estimation of temporal motifs counts; moreover, we show that even counting simple temporal star motifs is NP-complete. Thus, there is a need for fast and approximate algorithms. Here, we present the first frequency estimation algorithms for counting temporal motifs. More specifically, we develop a sampling framework that sits as a layer on top of existing exact counting algorithms and enables fast and accurate memory-efficient estimates of temporal motif counts. Our results show that we can achieve one to two orders of magnitude speedups over existing algorithms with minimal and controllable loss in accuracy on a number of datasets.

ACM Reference Format:
Paul Liu, Austin R. Benson, and Moses Charikar. 2019. Sampling Methods for Counting Temporal Motifs. In *The Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19), February 11–15, 2019, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3289600.3290988

1 SCALABLE PATTERN COUNTING IN TEMPORAL NETWORK DATA

Pattern counting is one of the fundamental problems in data mining [8, 18]. A particularly important case is counting patterns in graph data, which is used within a variety of network analysis tasks such as anomaly detection [42, 57], role discovery [19, 50], and clustering [6, 49, 59]. These methods typically make use of features derived from the frequencies of small graph patterns—usually called motifs [41] or graphlets [47] (we adopt the "motif" terminology in this paper)—and are used across a range of disciplines, including social network analysis [32, 60], neuroscience [5, 22], and computational biology [37, 46]. Furthermore, the counts of motifs have also been used to automatically uncover fundamental design principles in complex systems [37, 40, 41].

The scale of graph datasets has led to a number of algorithms for estimating the frequency of motif counts [2, 7, 12, 24, 63]. For example, just the task of estimating the number of triangles in a graph has garnered a substantial amount of attention [4, 11, 35, 52, 56, 58]. Many of these algorithms are based on sampling procedures amenable to streaming models of graph data [14, 38]. At this point, there is a reasonably mature set of algorithmic and statistical tools available for approximately counting motifs in large graph datasets.

While graphs have become large enough to warrant frequency estimation algorithms, graph datasets have, at the same time, become richer in structure. A particularly important type of rich information is time [13, 15, 21, 27, 51]. Specifically, in this paper, we consider datasets where edges are accompanied by a timestamp, such as the time a transaction was made with a cryptocurrency, the time an email was sent between colleagues, or the time a packet was forwarded from one IP address to another by a router. Accordingly, motifs have been generalized to incorporate temporal information [29, 45, 66] and have already been used in a variety of applications [30, 31, 39, 53]. However, we do not yet have algorithmic tools for estimating frequencies of temporal motifs in these large temporal graphs. This is especially problematic since including timestamps increases the size of the stored data; for example, a traditional email graph would only record if one person *has ever* emailed another person, whereas the temporal version of the same network would record *every time* there is a communication.

To exacerbate the problem, counting temporal motifs turns out to be fundamentally more difficult in a computational complexity sense. In particular, we prove that counting basic temporal star motifs is NP-complete. This contrasts sharply with stars in traditional static graphs, which are generally considered trivial to count (the number of non-induced k-edge stars with center node u is simply $\binom{d}{k}$, where d is the degree of u). Thus, our result highlights how counting problems in temporal graphs involve fundamentally more challenging computations, thus further motivating the need for approximation algorithms.

Here we develop the first frequency estimation algorithms for counting temporal motifs. We focus on the definition of temporal motifs from Paranjape et al. [45], but our methodology is general and could be adapted for other definitions. Our approach is based on sampling that employs as a subroutine any algorithm (satisfying some mild conditions) that *exactly* counts the number of instances of temporal motifs. Thus, our methodology provides a way to accelerate existing algorithms [36, 45], as well as better exact counting algorithms that could be developed in the future.

At a basic level, our sampling framework partitions time into intervals, uses some algorithm to find exact motif counts in a subset of the intervals, and weights these counts to get an estimate of

WSDM '19, February 11–15, 2019, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5940-5/19/02…$15.00
https://doi.org/10.1145/3289600.3290988

the number of temporal motifs. A key challenge is that the time duration of a temporal motifs can cross interval boundaries, which makes it challenging to obtain an accurate frequency estimator since motifs of larger duration are more likely to be omitted. At its core, our sampling framework uses importance sampling [43] in two different ways. First, we use importance sampling as a way to design an unbiased estimator by appropriately scaling the exact counts appearing in some intervals. Second, we use importance sampling as a way to (probabilistically) choose which intervals to sample, which reduces the variance of our unbiased estimator.

In addition to the scalability advantages offered by sampling, our framework has two other important features. First, the sampling requires a smaller amount of memory. We show an example where this enables us to count a complex motif on a large temporal graph when existing exact counting algorithms run out memory. Second, the sampling procedure has built-in opportunity for parallel computation, which provides a path to faster computation with exact counting algorithms that do not have built-in parallelism.

As discussed above, our sampling framework employs an exact counting algorithm as a subroutine. The constraints on the algorithm are that it must provide the exact counts along with the so-called *duration* of the motif instance (the difference in the earliest and latest timestamp in the edges in the motif instance; for example, the duration in the top left motif instance in Figure 1C is 32 - 16 = 16). This constraint holds for some existing algorithms [36] but not for others [45]. An additional contribution of our work is a new exact counting algorithm for a class of star motifs that is compatible with our sampling framework. As an added bonus, this new exact counting algorithm actually out-performs existing algorithms.

We test our sampling procedure on several temporal graph datasets from a variety of domains, ranging in size from 60,000 to over 600 million temporal edges and find that our sampling framework can improve run time of existing algorithms by one to two orders of magnitude while maintaining a relative error tolerance of 5% in the counts. The variance analysis of our error bounds tends to be pessimistic, since we make no assumptions on the distribution of timestamps within our datasets. Thus, we also show empirically that our worst-case bounds are far from what we see in the data.

2 PRELIMINARIES ON TEMPORAL MOTIFS

We first review some basic notions of temporal motifs. There are a few types of temporal motifs, which we discuss in the context of related work in Section 6. Here we review the definitions used by Paranjape et al. [45], which is one of the more flexible definitions that also poses difficult computational challenges.

A *temporal edge* is a timestamped directed edge between an ordered pair of nodes. A collection of temporal edges is a *temporal graph* (see Figure 1A). Formally, a temporal graph T on a node set V is a collection of tuples (u_i, v_i, t_i), $i = 1, \ldots, m$, where each u_i and v_i are elements of V and each t_i is a timestamp in $\mathbb{R}$. There can be many temporal edges from u to v (one example is in email data, where one person sends an email to another many times). We assume that the timestamps t_i are unique so that the temporal edges in a graph can be ordered. This assumption makes the presentation

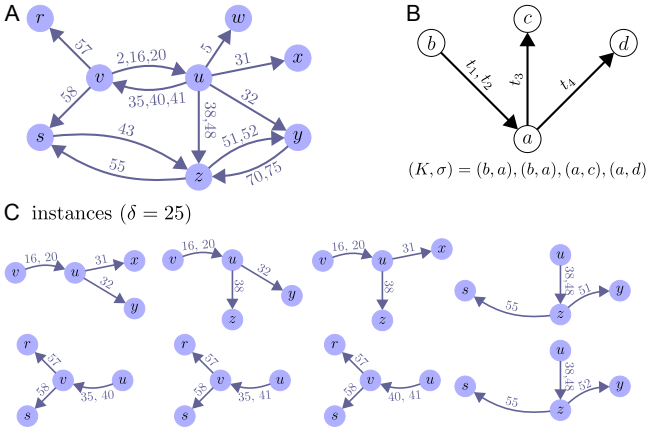

Figure 1: Temporal graph and temporal motifs. (A) Illustration of a temporal graph. The numbers along edges correspond to timestamps. There can be multiple timestamped edges between a given pair of nodes. (B) Illustration of a motif, which is formally a multigraph K with an ordering σ on its edges. (C) Eight δ-instances of the motif in the temporal graph with $\delta = 25$. The motifs match the multigraph, the edge ordering, and appear within the time span δ. The sequence of temporal edges $(u, v, 16)$, $(u, v, 20)$, $(u, y, 32)$, $(u, z, 48)$ is *not* a δ-instance of the motif because all edges do not fit within the time span δ. The duration of a motif instance M', denoted $\Delta(M')$, is the difference between the last and first timestamps; for example, the duration of the instance in the top left is $32 - 16 = 16$.

of the paper simpler, but our methods can handle temporal graphs with non-unique timestamps.

If we ignore time and duplicate edges, the temporal graph induces a standard (static) directed graph. Formally, the *static graph* of a temporal graph T on a node set V is a graph $G = (V, E)$, where $E = \{(u, v) \mid \exists t : (u, v, t) \in T\}$. Edges in G are called *static edges*.

Next, we formalize temporal motifs (illustrated in Figure 1B).

Definition 2.1 (Temporal motif [45]). A *k-node, l-edge* temporal motif $M = (K, \sigma)$ consists of a multigraph $K = (V, E)$ with k nodes and l edges and an ordering σ on the edges of E.

We often find it convenient to represent (K, σ) by an ordered sequence of edges $(u_1, v_1), (u_2, v_2), \ldots, (u_l, v_l)$. Definition 2.1 is a template for a temporal graph pattern, and we want to count how many times the pattern appears in a temporal network. Furthermore, we are interested in how often the motif occurs within some time span δ. A collection of edges in a temporal graph is a *δ-instance* of a temporal motif $M = (K, \sigma)$ if it matches the same edge pattern of the multigraph K, the temporal edges occur in the specified order σ, and all of the temporal edges occur within a δ time window (see Figure 1C). We now formalize this definition.

Definition 2.2 (Motif δ-instance [45]). A time-ordered sequence $S = (w_1, x_1, t_1), \ldots, (w_l, x_l, t_l)$ of l unique temporal edges from a temporal graph T is a *δ-instance* of the temporal motif $M = (u_1, v_1), \ldots, (u_l, v_l)$ if

(1) There exists a bijection f on the vertices in M such that $f(w_i) = u_i$ and $f(x_i) = v_i$, $i = 1, \ldots, l$; and
(2) The edges all occur within the δ time span, i.e., $t_l - t_1 \leq \delta$.

With this definition, motif instances are defined by just the existence of edges (a general subgraph) and not the non-existence of edges (an induced subgraph).

We are interested in counting how many motifs appear within a *maximum* time span of δ time units. Our sampling framework will also make use of the actual *duration* of motif instances, or the difference in the latest and earliest timestamp of a motif instance. We formalize this notion in the following definition.

Definition 2.3 (Motif duration). Let $S = (w_1, x_1, t_1), \ldots, (w_l, x_l, t_l)$ be an instance of a motif M as per Definition 2.2 with $t_1 < t_2 < \ldots t_l$. Then the *duration* of the instance, denoted $\Delta(S)$, is $t_l - t_1$.

3 COUNTING TEMPORAL STARS IS HARD

Star motifs as in Figure 1B are one of the fundamental small graph patterns and are used in, e.g., anomaly detection [3] and graph summarization [28]. In static graphs, counting non-induced instances of stars is simple. Given the degree d_u of node u, u is the center of $\binom{d_u}{k}$ $(k+1)$-node stars. Thus, there is a simple polynomial-time algorithm for computing the total number of stars.

In contrast, once we introduce temporal information, it turns out that stars become hard to compute. Specifically, we show in this section that counting temporal stars is NP-complete, and even determining the existence of a temporal star motif is NP-complete. This result serves two purposes. First, it highlights that the computational challenges with temporal graph data are fundamentally different from those in traditional static graph analysis. Second, the computational difficulty in such a simple type of temporal motif motivates the need for scalable approximation algorithms, which we develop in the next section. We begin with a formal definition of a temporal star motif.

Definition 3.1. A *k-temporal star* is a temporal motif where the multigraph is connected and has node set $\{0, 1, \ldots, k\}$ with edges (u_i, v_i), $i = 1, \ldots, m$, where either u_i or v_i is 0, $i = 1, \ldots, m$.

The restriction that either u_i or v_i is 0 means that each edge either originates from node 0 or enters node 0. The ordering σ of the edges in the multigraph needed by Definition 2.1 is arbitrary—we only need the star structure of the multigraph. We will show that determining the existence of an instance of a *k-temporal star* in a temporal graph is NP-complete and then generalize our result to an even more restricted class of star motifs. We begin with the formal decision problem.

Problem 1. *Given a temporal graph T, a k-temporal star S, and a time span δ, the k-Star-Motif problem asks if there exists at least one δ-instance of S in T.*

To establish NP-completeness, we reduce k-Clique to k-Star-Motif. A k-Clique problem instance is formalized as follows: given an undirected graph G and an integer k, the k-Clique problem asks if there exists at least one clique of size k in G.

Theorem 3.2. *k-Star-Motif is NP-complete.*

Proof. Our input is an instance (G, k) of k-Clique on a vertex set V. Assume that the nodes in V are numbered from 1 to $n = |V|$ (Figure 2A). We construct an instance (T, S, δ) of k-Star-Motif:

- **Construction of** T (Figure 2B). For each undirected edge (u, v) in G, add to T two edges $(0, u, (v-1)\cdot(n+2)+u+1)$ and $(0, v, (u-1)\cdot(n+2)+v+1)$. For each $u \in V$, we add two backward edges, $(u, 0, (u-1)\cdot(n+2)+1)$ and $(u, 0, u\cdot(n+2))$.
- **Construction of** S (Figure 2C). For each node $u \in V$, add two backward edges with timestamps $(u-1)\cdot(n+2)+1$ and $u\cdot(n+2)$, and $k-1$ forward edges with timestamps $\{(i-1)\cdot(n+2)+1+u \mid i \in [n] \setminus \{u\}\}$.
- Set $\delta = \infty$.

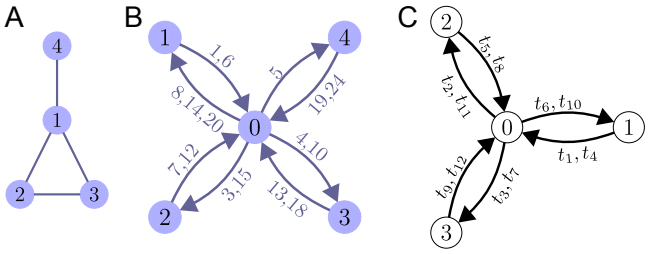

Figure 2: Structures used in proof of Theorem 3.2, which says that determining the existence of a temporal star is NP-complete. (A) A static graph G. (B) A temporal graph T. (C) A star motif S. With the reduction, there is a 3-clique in G if and only if there is a δ-instance of S in T with $\delta = \infty$.

The timestamps come from the set $\{1, \ldots, n^2 + 2n\}$, and we think of the timestamps as partitioned into n blocks, with $n+2$ timestamps in each block. If the timestamp of an edge lies in $\{(u-1)\cdot(n+2)+1, \ldots, u\cdot(n+2)\}$, then we say that the edge belongs to block u. Each block then corresponds to a node in G, with the first and last timestamp in each block reserved for the backward edges we add to T. For each node u in the original graph, we add the two backward edges in block u to node u in T, and for each neighbor v of u, we add a forward edge using the timestamp in the $(u+1)$-th position of block v. Figure 2 is a schematic of the construction. Observe that if there is a clique in G, then by construction the star motif S occurs in T.

Intuitively, the backward edges added to T and S serve as "bookends". If the two backward edges corresponding to a node u are found to be part of S, then each of the $k-1$ other nodes in the motif has to contribute a forward edge with timestamps between the two backward edges of u. By construction of S, an edge connected to v can only have a timestamp in block u if v is connected to u in G. This implies that u is connected to the $k-1$ other nodes selected in the motif S. Applying this argument to each node u in the motif S, there must be a clique in the original graph G. □

The result does not depend on having edges in two directions. We call a star motif S *unidirectional* if all of the edges in S either originate from or enter the center node (node 0, in our notation).

Theorem 3.3. *k-Star-Motif is NP-complete even when restricted to unidirectional stars.*

Proof. (Sketch.) Instead of using two backward edges for bookkeeping, we can expand the size of each block to $3n$ and use the first n and last n timestamps within the block as the bookends. Thus, the graph T in the previous proof is modified by connecting the center node to each node u with $2n$ forward edges using the $2n$ timestamps reserved for bookkeeping in block u. The motif S is modified by requiring the same forward edges as before, plus an additional $2n$ forward edges, with timestamps in $3(u-1)\cdot n\cdot+i$ and

$3(u-1)\cdot n+n+i$ for $i=1,\ldots,n$. By using n forward edges for each bookend, we ensure that any occurrence S found in T must include at least one edge from each bookend of the chosen nodes. This allows us to again argue that $k-1$ forward edges must be between the bookends of block u, implying that there is a k-clique. □

These hardness results illustrate the computational difficulties in counting temporal graph patterns, which motivates scalable approximation algorithms for counting such patterns. We next present a general sampling framework for scalable estimation of the number of instances of temporal motifs.

4 ALGORITHMIC SAMPLING FRAMEWORK

Suppose we are given a motif M, a time span δ, and a temporal graph T. In this section, we develop a sampling framework to estimate the number of δ-instances of M in T, which we denote by C_M. Our sampling framework will employ some algorithm that can compute exactly the number of δ-instances of M on temporal subgraphs of T. The requirements on the algorithm are that, given a temporal graph T', a motif M, and a time span δ, the algorithm A outputs a sequence of the count-duration pairs $\{(\text{count}_i, \Delta_i)\}$, where count_i is the number of instances of the motif with duration Δ_i. We denote this output by $A(T', M, \delta)$. We work from these assumptions in this section, and Section 5 discusses compatible algorithms.

Intervals and the count vector Y_s. We begin with some definitions and technical lemmas that will later be used to develop our estimator. Let s be a random integer uniformly drawn from $\{-c\delta+1,\ldots,0\}$ for some input integer $c>0$ that controls the size of the sampling windows. We call s a *shift*, and we will eventually make use of multiple shifts within our sampling framework. We consider the set of intervals of width $c\delta$ with shift s:

$$\mathcal{I}_s = \{[s+(j-1)c\delta,\, s+j\cdot c\delta-1],\ j=1,2,\ldots\}. \tag{1}$$

For an instance M' of the motif M with duration $\Delta(M')$, it is easy to see that the probability (over a random choice of shift s) that M' is completely contained within an interval in $\mathcal{I}_s$ is

$$p_{M'} = 1-\frac{\Delta(M')}{c\delta}. \tag{2}$$

Next, for an interval $I\in\mathcal{I}_s$, let $X_{M'}(I)$ be an indicator random variable which equals 1 if M' is completely contained in I and 0 otherwise. For each interval $I\in\mathcal{I}_s$, we associate a weighted count $w(I)$ of the number of instances of motif M completely contained in the interval I:

$$w(I) = \textstyle\sum_{M'} \frac{1}{p_{M'}} X_{M'}(I). \tag{3}$$

Let Y_s be the vector of such counts:

$$Y_{s,j} = w(I_j),\ I_j = [s+(j-1)c\delta,\, s+j\cdot c\delta-1]\in\mathcal{I}_s \tag{4}$$

(here, $Y_{s,j}$ denotes the jth coordinate of Y_s). Next, let $X_{M'}$ be an indicator random variable that equals 1 if the motif instance M' is completely contained in an interval in $\mathcal{I}_s$ and 0 otherwise. Then $\|Y_s\|_1 = \sum_{M'} \frac{1}{p_{M'}} X_{M'}$. The following lemma says that $\|Y_s\|_1$ is an unbiased estimator for the motif count C_M for any value of s.

Lemma 4.1. $\mathbb{E}[\|Y_s\|_1] = C_M$.

Proof. Since $\mathbb{E}[X_{M'}] = p_{M'}$, $\mathbb{E}[\|Y_s\|_1] = \sum_{M'} \frac{1}{p_{M'}}\mathbb{E}[X_{M'}] = \sum_{M'} 1 = C_M$. □

The next lemma bounds the variance of $\|Y_s\|_1$.

Lemma 4.2. $\text{Var}[\|Y_s\|_1] \le \frac{1}{c-1} C_M^2$.

Proof. First, we have that

$$\mathbb{E}[\|Y_s\|_1^2] = \mathbb{E}\left[\textstyle\sum_{M'} \frac{1}{p_{M'}} X_{M'}\right] = \sum_{M_1}\sum_{M_2} \frac{1}{p_{M_1} p_{M_2}} \mathbb{E}[X_{M_1} X_{M_2}],$$

where M_1 and M_2 range over the instances of the motif M. Using the bounds $\frac{1}{p_{M_2}} \le \frac{c}{c-1}$ and $\mathbb{E}[X_{M_1}X_{M_2}] \le \mathbb{E}[X_{M_1}]$,

$$\mathbb{E}[\|Y_s\|_1^2] \le \tfrac{c}{c-1} \textstyle\sum_{M_1} \frac{1}{p_{M_1}} \mathbb{E}[X_{M_1}] \sum_{M_2} 1 = \frac{c}{c-1} C_M^2.$$

Putting everything together,

$$\begin{aligned}\text{Var}[\|Y_s\|_1] &= \mathbb{E}[\|Y_s\|_1^2] - (\mathbb{E}[\|Y_s\|_1])^2 \\ &\le \frac{c}{c-1} C_M^2 - C_M^2 = \frac{1}{c-1} C_M^2.\end{aligned}$$

□

Our sampling framework estimates $\|Y_s\|_1$ in order to estimate the number of motif instances C_M. The basic idea of our approach is to use importance sampling to speed up this estimation task, by picking a set of intervals in $\mathcal{I}_s$ and computing their weights. Here, computing the weight for an interval I uses an *exact* motif count restricted to the interval I. Equivalently, we (i) sample a subset of coordinates of Y_s, (ii) compute their values exactly, and (iii) combine them to estimate $\|Y_s\|_1$. We describe this procedure next.

Importance sampling for an estimator. Let $Y_{s,j}$ denote the jth coordinate of Y_s, corresponding to interval I_j. Our estimator Z is a random variable defined as follows. First, we sample interval $I_j\in\mathcal{I}_s$ (independently) with some probability q_j. These q_j values will be based on simple statistics of the intervals; we will specify choices for q_j later but note for now that they do not necessarily sum to 1. Second, let Q_j be an indicator random variable corresponding to interval I_j, where Q_j equals 1 if j is picked and 0 otherwise. Finally, our estimator is

$$Z \triangleq \sum_j Q_j \frac{Y_{s,j}}{q_j}. \tag{5}$$

Our first result is that Z is an unbiased estimator for C_M, the number of instances of the motif M.

Theorem 4.3. *The random variable Z in Eq. (5) is an unbiased estimator for the number of motif instances, i.e., $\mathbb{E}[Z]=C_M$.*

Proof. First, note that $\mathbb{E}[Q_j] = q_j$. For any s, $\mathbb{E}[Z \mid s] = \sum_j Y_{s,j} = \|Y_s\|_1$. Hence, $\mathbb{E}[Z] = \mathbb{E}[\|Y_s\|_1] = C_M$ by Lemma 4.1. □

Next, we work to bound the variance of our estimator Z. To this end, it will be useful to define a scaled version $\hat{Y}_s$ of Y_s:

$$\hat{Y}_{s,j} \triangleq Y_{s,j}/\sqrt{q_j}. \tag{6}$$

The following lemma provides a useful equality on the variance of our estimator in terms of $\hat{Y}_s$ and Y_s, conditioned on the shift s.

Lemma 4.4. $\text{Var}[Z \mid s] = \|\hat{Y}_s\|_2^2 - \|Y_s\|_2^2$.

Proof. By independence of the Q_j,

$$\text{Var}[Z\mid s] = \textstyle\sum_j \text{Var}\left[Q_j \frac{Y_{s,j}}{q_j}\right] = \sum_j \frac{Y_{s,j}^2}{q_j^2} q_j(1-q_j).$$

Therefore, $\text{Var}[Z\mid s] = \sum_j Y_{s,j}^2/q_j - Y_{s,j}^2 = \|\hat{Y}_s\|_2^2 - \|Y_s\|_2^2$. □

Algorithm 1: Sampling framework for estimating the number of instances of a temporal motif in a temporal network. Without loss of generality, the timestamps in the temporal network are normalized to start at 0.

Input: Temporal graph T, motif M, time span δ, sampling probabilities q, number of shifts b, window size parameter c, exact motif counting algorithm A.
Output: Estimate of the number of instances of M.

$Z \leftarrow 0, \quad t_{\max} \leftarrow \max\{t \mid (u, v, t) \in T\}$
for $k = 1, \ldots, b$ **do**
  $s \leftarrow$ random integer from $\{-c\delta + 1, \ldots, 0\}$
  for $j = 1, \ldots, 1 + \lceil \frac{t_{\max}}{c\delta} \rceil$ *(in parallel)* **do**
    if *Uniform*$(0, 1) \le q_j$ **then**
      $T_j \leftarrow \{(u, v, t) \in T \mid t \in [s+(j-1)c\delta, s+j\cdot c\delta-1]\}$
      for $(\text{count}_i, \Delta_i) \in A(T_j, M, \delta)$ **do**
        $Z_k \leftarrow Z_k + \text{count}_i/((1 - \Delta_i/(c\delta)) \cdot q_j)$
return $\frac{1}{b} \sum_{k=1}^{b} Z_k$

We are now ready to bound the variance of Z.

Theorem 4.5. $\text{Var}[Z] \le \mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2] + \frac{1}{c-1} C_M^2$.

Proof. For this bound, we first condition on s and then take the expectation over random choice of s.

$$\begin{aligned}
\text{Var}[Z] &= \mathbb{E}[(Z - C_M)^2] \\
&= \mathbb{E}_s\left[((Z - \|Y_s\|_1) + (\|Y_s\|_1 - C_M))^2\right] \\
&= \mathbb{E}_s\left[\text{Var}[Z \mid s] + (\|Y_s\|_1 - C_M)^2\right] \\
&= \mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2] + \text{Var}[\|Y_s\|_1] \qquad \text{(by Lemma 4.4)} \\
&\le \mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2] + \frac{1}{c-1} C_M^2 \qquad \text{(by Lemma 4.2)}
\end{aligned}$$

□

Our analysis thus far has been for a single shift s. If we repeat the above computations for b randomly chosen shifts and report the mean of the estimates, then the variance is reduced by a factor of b. Algorithm 1 outlines the the overall sampling procedure, assuming that the sampling probabilities q_j are given along with the exact counting algorithm A. In the algorithm, we use T_j to denote the subgraph restricted to interval I_j and $A(T_j, M, \delta)$ to denote the output of the exact counting algorithm on the interval, which is a sequence of the count-duration pairs $\{(\text{count}_i, \Delta_i)\}$ of motif instances contained in the interval. The algorithm also explicitly states that the parallelism that can be performed over the samples.

Choosing the sampling probabilities. In order to get average squared error $(\epsilon C_M)^2$, we need to set the parameters as follows:

$$\mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2] + \frac{1}{c-1} C_M^2 \le b(\epsilon C_M)^2 \tag{7}$$

$$\iff \frac{\mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2]}{C_M^2} + \frac{1}{c-1} \le b\epsilon^2. \tag{8}$$

The first term in the left-hand side of Eq. (8) combines (i) a natural measure of sparsity of the distribution of motifs with (ii) the extent of correlation between the sampling probabilities q_j and the (weighted) motif counts for intervals $Y_{s,j}$. In order to understand this, let ℓ denote the dimension of Y_s and consider the simple uniform setting of $q_j = 1/\ell$ (so one interval is sampled in expectation). In this case, the term becomes

$$\frac{(\ell - 1)\mathbb{E}[\|Y_s\|_2^2]}{\mathbb{E}[\|Y_s\|_1^2]}. \tag{9}$$

Equation (9) is a natural measure of sparsity of the vector Y_s. In the extreme case where Y_s is a vector with only one non-zero coordinate, the value is $\ell - 1$, and in the other extreme where Y_s is a uniform vector, the value is bounded above by 1. In the sparse case, we need to increase the sampling probabilities—thus sampling more intervals—to compensate for the large variance. In the worst case, this would require looking at all the intervals, i.e., we get no running time savings from sampling (however, we will see in our experiments that the data is far from the worst case in practice). Nonetheless, the ability of the algorithm to pick sampling probabilities q_j gives flexibility to mitigate the dependence on the sparsity of Y_s. To illustrate this point, in the extremely favorable case when q_j is proportional to $Y_{s,j}$, i.e., $q_j = Y_{s,j}/\sum_j Y_{s,j}$ (so one interval is sampled in expectation), the first term on the left-hand side of Eq. (8) is less than 1. This analysis suggests that a good choice of sampling probabilities roughly balances the two terms:

$$\frac{\mathbb{E}[\|\hat{Y}_s\|_2^2] - \mathbb{E}[\|Y_s\|_2^2]}{C_M^2} \approx \frac{1}{c-1}.$$

A priori, we do not know Y_s or $\hat{Y}_s$. What we can do is choose the sampling probabilities by some easily measured statistic that we think is correlated with Y_s, such as the number of temporal edges or number of static edges. For this paper, we simply choose q_j to be proportional to the number of temporal edges in the interval, i.e.,

$$q_j = r \cdot \frac{|\{(u, v, t) \in T \mid t \in I_j\}|}{|T|}, \tag{10}$$

where r is a small constant (in practice on the order of 10–100). This leads to substantial speedups, as we will see in the next section. There are certainly more sophisticated approaches one could take to choose the q_j, and we leave this as an avenue for future research.

Streaming from sampling. When memory is at a premium, the sampling framework above can be made memory efficient. By considering the windows $\mathcal{I}_s$ in chronological order, the edges of past windows do not need to be stored. By running several estimators in parallel, we can achieve any accuracy we want while only needing to store edges in an interval of at most $c\delta$ at a time. As we will see in our experiments, the memory savings allows us to processes larger temporal graphs than we could with an exact algorithm.

5 COMPUTATIONAL EXPERIMENTS

In this section, we use our sampling framework from Section 4 and various exact temporal motif counting algorithms to count temporal motifs on real-world datasets. By exploiting sampling and the ability sample in parallel, we obtain substantial speedups with modest computational resources and only a small error in the estimation. Datasets and implementations of our algorithm are available at https://gitlab.com/paul.liu.ubc/sampling-temporal-motifs.

Data. We gathered 10 datasets for our experiments. Paranjape et al. analyzed seven of them [45], and we collected three larger datasets

Table 1: Summary statistics of temporal networks.

dataset	# nodes	# static edges	# temporal edges	time span
CollegeMsg	1.9K	20.3K	59.8K	194 days
email-Eu-core	986	24.9K	332K	2.20 years
MathOverflow	24.8K	228K	390K	6.44 years
AskUbuntu	157K	545K	727K	7.16 years
SuperUser	192K	854K	1.11M	7.60 years
WikiTalk	1.09M	3.13M	6.10M	6.24 years
StackOverflow	2.58M	34.9M	47.9M	7.60 years
Bitcoin	48.1M	86.8M	113M	7.08 years
EquinixChicago	12.9M	17.0M	345M	62.0 mins
RedditComments	8.40M	517M	636M	10.1 years

to better analyze the performance of our methodology. Table 1 lists summary statistics of the datasets, and we briefly describe them below. Each dataset is a collection of timestamped directed edges. The time resolution of each dataset is 1 second, except for the EquinixChicago dataset, where the time resolution is 1 microsecond.

CollegeMsg [44]. A network of private messages sent on an online social network at the University of California, Irvine.

email-Eu-core [45]. A collection of internal email records from a European research institution.

MathOverflow, AskUbuntu, SuperUser, and StackOverflow [45]. These datasets are derived from user interactions on Stack Exchange question and answer forums. A temporal edge represents a user replying to a question, replying to a comment, or commenting on a question.

WikiTalk [33, 45]. A network of Wikipedia users making edits on each others' "talk pages."

Bitcoin [26]. A network representing timestamped transactions on Bitcoin. The addresses were partially aggregated by a de-identification heuristic [48] implemented by Kondor et al., using all transactions up to February 9, 2016 [26]. Timestamps are the creation time of the block on the blockchain containing the transaction. We will release this dataset with the paper.

EquinixChicago [1]. This dataset was constructed from passive internet traffic traces from CAIDA's monitor in Chicago on February 17, 2011. Each edge represents a packet sent from one anonymized IP address to another. Data was collected from the "A direction" of the monitor.

RedditComments [20]. This dataset was constructed from a large collection of comments made by users on https://www.reddit.com, a popular social media platform. A comment from user u to user v at time t induces a temporal edge in our dataset.

5.1 Exact counting algorithms

Our sampling framework is flexible since it can use any algorithm that exactly counts temporal motifs as a subroutine, provided that this algorithm can be transformed to output the count-duration pairs $\{(\text{count}_i, \Delta_i)\}$, where count_i is the number of instances of the motif with duration Δ_i. A recently proposed "backtracking" algorithm satisfies this constraint [36]. The fast algorithms for 2-node, 3-edge star motifs introduced by Paranjape et al. do not satisfy these requirements, since the algorithm uses an inclusion-exclusion rule that cannot output the durations. However, we still use this algorithm as a baseline in our experiments. We also create a new exact counting algorithm that is compatible with our sampling framework, which we describe below.

Algorithm 2: EX23: A simple exact algorithm to count the two-node motif in Figure 3A. This algorithm can easily be modified to count any 2-node, 3-edge temporal motif.

Input: Two nodes u and v, and a sequence of temporal edges $(e_1, t_1), \ldots, (e_L, t_L)$ with $t_1 < \ldots < t_L$, time span δ, and $e_i = (u, v)$ or (v, u).
Output: List $\{(\text{count}_i, \Delta_i)\}$ of counts of instances of the motif in Figure 3A between nodes u and v with durations Δ_i.

```
C ← empty counter dictionary with default value 0
for i = 1 ... L do
    if e_i ≠ (u, v) then continue
    N_b ← 0
    for j = i + 2 ... L do
        Δ ← t_j − t_i
        if t_j − t_i > δ then break
        if e_j = (v, u) then N_b ← N_b + 1
        else C[Δ] ← C[Δ] + N_b
return [(C[Δ], Δ) for key Δ in C]
```

Backtracking algorithm (BT, [36]). The backtracking algorithm examines the edges of the input graph in chronlogical order and matches one edge of the motif at a time. The software was not released publicly, so we have re-implemented it with some optimizations. The algorithm is compatible with our sampling framework. The algorithm is inherently sequential, so our parallel sampling framework is especially useful with this method.

Fast 2-node, 3-edge algorithm (F23, [45]). Paranjape et al. introduced a collection of algorithms for counting motifs with at most 3 edges. Here we use their specialized algorithm for motifs with two nodes and three edges, which produces exact counts in time linear in the number of edges. This algorithm is incompatible with our sampling framework, as it does not report the durations (Definition 2.3) of the counting motifs. However, we still use it for comparative purposes.

A new exact algorithm for 2-node 3-edge motifs (EX23). We devised a new algorithm for 2-node, 3-edge motifs that is compatible with our sampling framework. In this case, each pair of nodes in the input graph forms an independent counting problem. For each pair of nodes in the input that are neighbours in the static graph, we gather all temporal edges between the two nodes. Then we fix the first and last edge of the 3-edge motif by iterating over all pairs of gathered temporal edges. By maintaining an additional counter of the number of edges between the two fixed edges, we can count the number of temporal motifs that begins on the first fixed edge and ends on the second fixed edge (the procedure is outlined in Algorithm 2). Overall, this procedure takes $O(\sum_{u,v} k_{u,v}^2)$ time, where the sum iterates over all pairs of nodes in the graph, and $k_{u,v}$ is the number of temporal edges between nodes u and v. With additional code complexity from special tree structures, the running time can be improved to $O(\sum_{u,v} k_{u,v} \log k_{u,v})$ and still be compatible with our sampling framework. However, this optimization is not crucial for the main focus of our paper, which is the acceleration of counting algorithms with sampling. Thus, we

Table 2: Running time in seconds of algorithms with and without our sampling framework and with and without parallelism. The modifiers "+S" and "+PS" stands for sampling and parallelized sampling respectively. We compare the backtracking algorithm (BT, [36]), our specialized algorithm for counting 2-node, 3-edge motifs (EX23, Algorithm 2), and the specialized fast algorithm from Paranjape et al. for counting 2-node, 3-edge motifs (F23, [45]). *In all datasets, our EX23 algorithm within our parallel sampling framework has the fastest running time.*

dataset	BT	BT+S	BT+PS	EX23	EX23+S	EX23+PS	F23	F23+P	error (%)
CollegeMsg	0.076	0.072	0.038	0.017	0.016	0.009	0.056	0.054	0.33
Email-Eu	0.339	0.305	0.191	0.073	0.078	0.027	0.217	0.165	1.44
MathOverflow	0.545	0.361	0.143	0.233	0.148	0.097	0.998	0.878	1.74
AskUbuntu	1.414	1.305	0.500	0.592	0.311	0.176	2.534	2.371	1.84
SuperUser	2.590	1.446	0.483	1.097	0.194	0.104	4.595	4.129	1.69
WikiTalk	15.92	14.88	5.463	4.737	3.645	0.876	20.46	18.23	0.89
StackOverflow	198.9	160.8	79.58	108.1	69.50	17.81	299.2	230.1	1.95
Bitcoin	514.0	520.7	102.3	494.4	233.5	88.66	10348	10135	3.59
EquinixChicago	480.4	180.3	37.64	382.7	56.33	24.64	477.3	383.8	0.00
RedditComments	7301	7433	2910	1563	3154	367.4	6602	5036	4.83

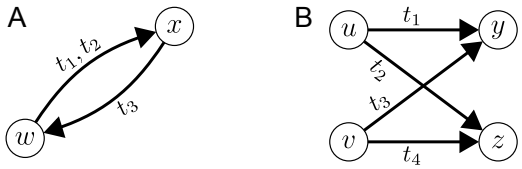

Figure 3: Motifs used in counting experiments. (A) The 2-node 3-edge motif for which results are reported in Table 2. (B) The bi-fan motif for which results are reported in Table 3.

use the simpler un-optimized algorithm, which we will see actually out-performs the other exact counting algorithms.

5.2 Performance results

We now evaluate the performance of several algorithms: (i) the three baseline exact counting algorithms described in the previous section (BT, F23, EX23); (ii) the F23 baseline with parallelism enabled (F23+P); (iii) our sampling framework on top of backtracking and our new exact counting algorithm (BT+S, EX23+S); and (iv) our parallelized sampling framework on top of backtracking and our new exact counting algorithm (BT+PS, EX23+PS). As explained above, the F23 algorithm is incompatible with our sampling framework; we include the algorithm and its parallelized version as baselines for fast exact counting.

All algorithms were implemented in C++, and all experiments were executed on a 16-core 2.20 GHz Intel Xeon CPU with 128 GB of RAM. The algorithms ran on a single thread unless explicitly stated to be parallel. The parallel algorithms used 16 threads. In the case of the sampling algorithm, parameters are set so that the approximations are within 5% relative error of the true value.

Experiments on a 2-node, 3-edge motif. Table 2 reports the performance of all algorithms on the 2-node, 3-edge temporal motif in Figure 3A (we chose this motif to allow us to compare against one of the fast algorithms of Paranjape et al. [45]). The time span δ was set to 86400 seconds = 1 day in all datasets except EquinixChicago, where δ was 86400 microseconds (these are the same parameters used in exploratory data analysis in prior work [36]).

We highlight three important findings. First, our new EX23 algorithm with parallel sampling is the fastest algorithm on every dataset. Comparing our algorithm against the previous state of the art, we see speedups up to 120 times faster than the slowest exact algorithm (see the results for Bitcoin). This is in part due to the fact that our EX23 algorithm is actually faster the than the backtracking algorithm (BT) and the fast algorithm of Paranjape et al. (F23). In other words, our proposed exact algorithm already out-performs the current state of the art.

Second, in all cases, parallel sampling provides a substantial speedup over the exact baseline algorithm. Speedups are typically on the order of 2–6x improvements in running time. We used 16 threads but did not optimize our parallel algorithms; there is ample room to improve these results with additional software effort.

Third, the running time of the backtracking algorithm with sampling (BT+S) is often comparable to simple backtracking (BT). In these cases, we hypothesize that the backtracking algorithm has enough overhead and is pruning enough edges to make simple sampling not worthwhile under our parameter settings. However, parallel sampling with the backtracking algorithm (BT+PS) can yield substantial speedups (see, e.g., Bitcoin, EquinixChicago, and RedditComments). This illuminates an important feature of our sampling framework, namely, we get parallelism for free. The backtracking algorithm is inherently sequential, but parallel sampling can achieve substantial speedups. Thus, future research in the design of fast exact counting algorithms can largely leave parallelism to be handled by our sampling framework. Finally, although not reported, the sampling framework requires a smaller amount of memory than the exact algorithms; thus, if no parallelism is available, we can at least gain in terms of memory, if not in speed.

We used the heuristic in Eq. (10) to determine the sampling probabilities q in these experiments. To understand why this heuristic worked for these datasets (i.e., the relative errors are small), we measured the correlation of q and the coordinates of the vector Y_s used in the sampling framework (Figure 4). In datasets such as CollegeMsg and WikiTalk, the correlation between is large, and consequently, the relative error in the estimates is small (Table 2).

Experiments on a 4-node, 4-edge bi-fan motif. Next, we show the results of the backtracking algorithm on a so-called "bi-fan" motif (Figure 3B). This motif has four nodes and four temporal

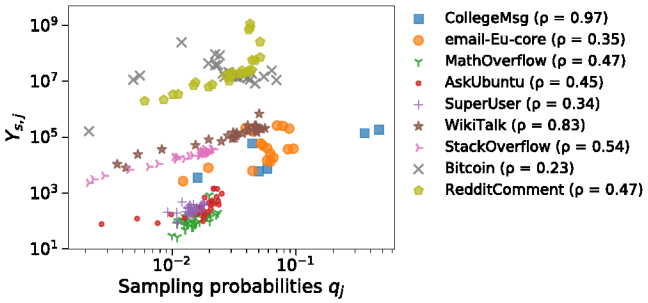

Figure 4: Sampling probabilities, which are based on the number of edges (Eq. (10)) and the values of Y_s for one sample s used in estimating the counts of the 2-node, 3-edge temporal motif in Figure 3A. For our sampling procedure to be effective, these values should be positively correlated (see the discussion in Section 4). Positive correlation leads to less variance in the estimation. The correlation ρ is listed in the legend. The CollegeMsg and WikiTalk datasets both have a strong positive correlation and a small relative error in the motif count estimate (see Table 2).

Table 3: Running times in seconds of the backtracking algorithm (BT, [36]) with our sampling framework ("+S" denotes serial sampling and "+PS" denotes parallel sampling). The factor of speedup over the baseline BT algorithm is also shown. The symbol "✗" indicates that the algorithm failed due to the machine running out of memory. Our sampling framework uses less memory than the exact counting algorithm, so it is always able to estimate the motif count on these datasets.

dataset	BT	BT+S	BT+PS	error (%)
CollegeMsg	0.081	0.076/1.1x	0.069/1.2x	1.02
Email-Eu	0.353	0.307/1.2x	0.120/2.9x	0.85
MathOverflow	0.528	0.362/1.5x	0.041/12.9x	3.60
AskUbuntu	1.408	0.909/1.5x	0.078/18.1x	4.52
SuperUser	2.486	1.269/1.96x	0.164/15.2x	2.33
WikiTalk	51.85	35.35/1.5x	11.99/4.3x	2.01
StackOverflow	221.7	93.10/2.4x	5.208/42.6x	4.88
Bitcoin	1175	985.9/1.2x	269.3/4.4x	3.09
EquinixChicago	481.2	45.50/10.8x	5.666/84.9x	1.33
RedditComments	✗	6739/–	2262/–	–

edges. The static version of the motif appears is important in networks from a variety of domains, including sociology [64], neuroscience [6], gene regulation [10], and circuit design [41]. Since this motif has four nodes and four edges, our new fast algorithm and the fast algorithm of Paranjape et al. cannot be used. Thus, we focus on accelerating the backtracking algorithm with (parallelized) sampling. For these experiments, we set $\delta = 3600$, as running the algorithm with $\delta = 86400$ exceeds the alloted memory on the Bitcoin and RedditComments dataset. Table 3 shows the results.

Again, the parallel sampling procedure provides a substantial speed-up over the baseline algorithm. We emphasize that our simple parallelization technique is a property of the sampling procedure and not a property of the exact algorithm. In fact, the exact algorithm is inherently sequential, and the sampling framework enables parallelism in a trivial way with minimal loss in accuracy. Moreover, since the sampling algorithm only examines a portion of the graph at a time, it uses much less memory than the exact counting algorithm. For example, with the RedditComments dataset, the exact algorithm ran out of memory, while the sampling algorithms completed successfully (thus no relative error is reported). This feature is useful in streaming applications, where memory is limited.

6 ADDITIONAL RELATED WORK

Our sampling framework relies on importance sampling, which is used for finding motifs in gene sequence analysis [9, 16, 34, 55]; here "motif" refers to short string patterns in DNA. (The term "motif" in the context of network analysis is borrowed from this domain [54].) We have already covered much of the related work in sampling algorithms for pattern counting in static graphs, so we summarize additional research related to various definitions of temporal motifs here. Some of these are for sequences of static snapshot graphs [25, 31, 65], which is a different data model than the one in this paper, where edges have timestamps in a continuum. For the data in our paper, there are motifs based on "adjacent events" that require each new edge in a sequence to be within a certain timespan of each other [17, 23, 66]. These definitions are slightly more restrictive than the one by Paranjape et al. analyzed here; however, the principles of our techniques could also be adapted to these cases as well. Kovanen et al. use the same notion of event adjacency but also restrict motif instances to cases where the events are consecutive for all nodes involved (i.e., within the span of the motif instance, there can be no other temporal edge adjacent to one of the nodes) [29]. This definition is even more restrictive in the events that it captures but it does allows for much faster exact counting algorithms, e.g., triangle motifs can be counted in linear time in the size of the data. Thus, speeding up computation with sampling is less appealing for this definition. Finally, there is also a line of research in finding dense subgraphs in datasets similar to our model, which is a specific type of motif [15, 61, 62].

7 DISCUSSION

We have developed a sampling framework for estimating the number of instances of temporal motifs in temporal graphs. Overall, our sampling framework is flexible in several ways. First, the framework is built on top of exact counting algorithms. Improvements in these algorithms can be used directly within our framework, provided it meets the conditions needed by our framework. In fact, we demonstrated this to be the case with the specialized algorithm we developed for 2-node, 3-edge motifs in Section 5, which was faster than existing exact counting methods for that particular motif. Second, our sampling framework provides natural parallelism, which allowed us to achieve orders of magnitude speedups on algorithms that do not have obvious parallel implementations. Finally, the sampling is inherently less memory intensive, which allowed us to estimate motif counts on datasets on which exact algorithms cannot even run (Table 3); thus, our framework makes knowledge discovery feasible in new cases.

An extraordinary amount of research has gone into scalable estimation algorithms for counting patterns in static graphs. Our paper takes this line of research in a new direction by considering richer patterns that arise when temporal information is incorporated into the graph. We anticipate that our work will open new challenges

for algorithm designers while simultaneously providing a solution for domain scientists working with large-scale temporal networks.

ACKNOWLEDGEMENTS

This research was supported in part by NSF grant DMS-1830274, NSF grant CCF-1617577, a Google research award, and a Simons Investigator Award. Additionally, we thank Eugene Y.Q. Shen for donating the computing resources that made some of the larger datasets possible.

REFERENCES

[1] The CAIDA UCSD Anonymized Internet Traces – February 17, 2011. http://www.caida.org/data/passive/passive_dataset.xml.
[2] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella. Graph sample and hold: a framework for big-graph analytics. In *Proceedings of KDD*, 2014.
[3] L. Akoglu, M. McGlohon, and C. Faloutsos. OddBall: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining*. 2010.
[4] H. Avron. Counting triangles in large graphs using randomized matrix trace estimation. In *Workshop on Large-scale Data Mining: Theory and Applications*, volume 10, 2010.
[5] F. Battiston, V. Nicosia, M. Chavez, and V. Latora. Multilayer motif analysis of brain networks. *Chaos*, 27(4):047404, 2017.
[6] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.
[7] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Counting Graphlets: Space vs Time. In *Proceedings WSDM*. ACM Press, 2017.
[8] S. Chakrabarti, M. Ester, U. Fayyad, J. Gehrke, J. Han, S. Morishita, G. Piatetsky-Shapiro, and W. Wang. Data mining curriculum: A proposal. *Intensive Working Group of ACM SIGKDD Curriculum Committee*, 140, 2006.
[9] H. P. Chan, N. R. Zhang, and L. H. Chen. Importance sampling of word patterns in dna and protein sequences. *Journal of Computational Biology*, 2010.
[10] R. Dobrin, Q. Beg, A.-L. Barabási, and Z. Oltvai. Aggregation of topological motifs in the *Escherichia coli* transcriptional regulatory network. *BMC Bioinformatics*, 2004.
[11] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, jan 2017.
[12] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed estimation of graph 4-profiles. In *Proceedings of WWW*, 2016.
[13] M. Farajtabar, Y. Wang, M. G. Rodriguez, S. Li, H. Zha, and L. Song. Coevolve: A joint point process model for information diffusion and network co-evolution. In *Advances in Neural Information Processing Systems*, pages 1954–1962, 2015.
[14] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.
[15] N. Gaumont, C. Magnien, and M. Latapy. Finding remarkably dense sequences of contacts in link streams. *Social Network Analysis and Mining*, 6(1), sep 2016.
[16] M. Gupta and J. G. Ibrahim. Variable selection in regression mixture modeling for the discovery of gene regulatory networks. *Journal of the American Statistical Association*, 2007.
[17] S. Gurukar, S. Ranu, and B. Ravindran. COMMIT: A Scalable Approach to Mining Communication Motifs from Dynamic Networks. In *Proceedings of SIGMOD*, 2015.
[18] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.
[19] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. RolX: structural role extraction & mining in large graphs. In *Proceedings of KDD*, 2012.
[20] J. Hessel, C. Tan, and L. Lee. Science, askscience, and badscience: On the coexistence of highly related communities. In *ICWSM*, 2016.
[21] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 2012.
[22] Y. Hu, J. Trousdale, K. Josić, and E. Shea-Brown. Motif statistics and spike correlations in neuronal networks. *BMC Neuroscience*, 13(Suppl 1):P43, 2012.
[23] Y. Hulovatyy, H. Chen, and T. Milenković. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinformatics*, 2015.
[24] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using Turán's theorem. In *Proceedings of WWW*. ACM Press, 2017.
[25] R. Jin, S. McCallen, and E. Almaas. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *Proceedings of ICDM*, 2007.
[26] D. Kondor, I. Csabai, J. Szüle, M. Pósfai, and G. Vattay. Inferring the interplay between network structure and market effects in Bitcoin. *New J. of Physics*, 2014.
[27] G. Kossinets, J. Kleinberg, and D. Watts. The structure of information pathways in a social communication network. In *Proceeding of KDD*, 2008.
[28] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Summarizing and understanding large graphs. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(3):183–202, may 2015.
[29] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki. Temporal motifs in time-dependent networks. *J. of Stat. Mech.: Theory and Experiment*, 2011.
[30] L. Kovanen, K. Kaski, J. Kertesz, and J. Saramaki. Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences. *PNAS*, 2013.
[31] M. Lahiri and T. Y. Berger-Wolf. Structure prediction in temporal networks using frequent subgraphs. In *IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2007.
[32] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings CHI*, 2010.
[33] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Governance in social media: A case study of the wikipedia promotion process. In *Proceedings of ICWSM*, 2010.
[34] K.-c. Liang, X. Wang, and D. Anastassiou. A sequential monte carlo method for motif discovery. *IEEE transactions on signal processing*, 56(9):4496–4507, 2008.
[35] Y. Lim and U. Kang. MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams. In *Proceedings of KDD*, 2015.
[36] P. Mackey, K. Porterfield, E. Fitzhenry, S. Choudhury, and G. Chin Jr. A chronological edge-driven approach to temporal subgraph isomorphism. *arXiv*, 2018.
[37] S. Mangan and U. Alon. Structure and function of the feed-forward loop network motif. *PNAS*, 2003.
[38] A. McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 2014.
[39] C. Meydan, H. H. Otu, and O. Sezerman. Prediction of peptides binding to MHC class I and II alleles by temporal motif mining. *BMC Bioinformatics*, 2013.
[40] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of evolved and designed networks. *Science*, 2004.
[41] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 2002.
[42] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *Proceedings of KDD*, 2003.
[43] A. B. Owen. *Monte Carlo theory, methods and examples*. 2013.
[44] P. Panzarasa, T. Opsahl, and K. M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *J. of the American Society for Information Science and Technology*, 60(5):911–932, 2009.
[45] A. Paranjape, A. R. Benson, and J. Leskovec. Motifs in temporal networks. In *Proceedings of WSDM*. ACM Press, 2017.
[46] N. Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, jan 2007.
[47] N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, jul 2004.
[48] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*, pages 197–223. 2012.
[49] K. Rohe and T. Qin. The blessing of transitivity in sparse and stochastic networks. *arXiv*, 2013.
[50] R. A. Rossi and N. K. Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, apr 2015.
[51] I. Scholtes. When is a network a network?: Multi-order graphical model selection in pathways and temporal networks. In *Proceedings of KDD*, 2017.
[52] C. Seshadhri, A. Pinar, and T. G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of SDM*, 2013.
[53] H. Shao, M. Marwah, and N. Ramakrishnan. A temporal motif mining approach to unsupervised energy disaggregation: Applications to residential and commercial buildings. In *Proceedings of AAAI*, 2013.
[54] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 2002.
[55] R. Siddharthan. Phylogibbs-mp: module prediction and discriminative motif-finding by gibbs sampling. *PLoS computational biology*, 2008.
[56] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal. TRIÈST: Counting Local and Global Triangles in Fully Dynamic Streams with Fixed Memory Size. *ACM Transactions on Knowledge Discovery from Data*, 11(4):1–50, jun 2017.
[57] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. GraphScope: Parameter-free Mining of Large Time-evolving Graphs. In *Proceedings of KDD*, 2007.
[58] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *Proceedings of KDD*, 2009.
[59] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of WWW*, 2017.
[60] J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of WWW*, 2013.
[61] T. Viard, M. Latapy, and C. Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, jan 2016.
[62] T. Viard, C. Magnien, and M. Latapy. Enumerating maximal cliques in link streams with durations. *Information Processing Letters*, 133:44–48, may 2018.
[63] P. Wang, J. Zhao, X. Zhang, Z. Li, J. Cheng, J. C. Lui, D. Towsley, J. Tao, and X. Guan. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
[64] Q.-M. Zhang, L. Lü, W.-Q. Wang, and T. Z. and. Potential theory for directed networks. *PLoS ONE*, 2013.
[65] X. Zhang, S. Shao, H. E. Stanley, and S. Havlin. Dynamic motifs in socio-economic networks. *Europhysics Letters*, 108(5):58001, dec 2014.
[66] Q. Zhao, Y. Tian, Q. He, N. Oliver, R. Jin, and W.-C. Lee. Communication motifs: a tool to characterize social communications. In *Proceedings of CIKM*, 2010.