Scalable Motif Counting for Large-scale Temporal Graphs

Zhongqiang Gao[1]*, Chuanqi Cheng[1]*, Yanwei Yu[1,✉], Lei Cao[2], Chao Huang[3], Junyu Dong[1]

[1]College of Computer Science and Technology, Ocean University of China, Qingdao, China
[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, USA
[3]Department of Computer Science, The University of Hong Kong, Hong Kong, China
{gaozhongqiang, chengchuanqi}@stu.ouc.edu.cn, {yuyanwei, dongjunyu}@ouc.edu.cn,
lcao@csail.mit.edu, chuang@cs.hku.hk

***Abstract*—One fundamental problem in temporal graph analysis is to count the occurrences of small connected subgraph patterns (*i.e.*, motifs), which benefits a broad range of real-world applications, such as anomaly detection, structure prediction, and network representation learning. However, existing works focused on exacting temporal motif are not scalable to large-scale temporal graph data, due to their heavy computational costs or inherent inadequacy of parallelism. In this work, we propose a scalable parallel framework for exactly counting temporal motifs in large-scale temporal graphs. We first categorize the temporal motifs based on their distinct properties, and then design customized algorithms that offer efficient strategies to exactly count the motif instances of each category. Moreover, our compact data structures, namely triple and quadruple counters, enable our algorithms to directly identify the temporal motif instances of each category, according to edge information and relationship between edges, therefore significantly improving the counting efficiency. Based on the proposed counting algorithms, we design a hierarchical parallel framework that featuring both inter- and intra-node parallel strategies, and fully leverages the multi-threading capacity of modern CPU to concurrently count all temporal motifs. Extensive experiments on sixteen real-world temporal graph datasets demonstrate the superiority and capability of our proposed framework for temporal motif counting, achieving up to $538\times$ speedup compared to the state-of-the-art methods. The source code of our method is available at: https://github.com/steven-ccq/FAST-temporal-motif.**

I. Introduction

Many real-world applications are naturally represented in a graph data structure such as social networks, traffic networks, citation networks, biology networks, and knowledge graphs, where objects and the relationships among them are respectively represented by nodes and edges. In real-world scenarios, many networks constantly evolve over time with their structures dynamically evolving as new relationships constantly emerges. Such dynamic networks are termed as *temporal graphs* [1], [2] composed of a set of nodes and a series of timestamped edges between nodes, or temporal edges. Examples include email networks, communication networks, financial transactions, and E-commercial networks.

*Authors contribute this work equally. Corresponding author: Yanwei Yu.

Counting patterns in graph data is one of fundamental problems in graph data mining, widely used in a variety of network analytical tasks such as anomaly detection [3], role discovery [4], and community detection [5]. An especially useful case is counting *motifs* (or *graphlets*) – a category of frequent subgraph patterns, which are used in range of disciplines, including social network analysis [6], neuroscience [7] and computational biology [8]. For example, social network analysis often uses communication motifs mined from large dynamic networks to understand how human communication unfold [9]. Moreover, because motif is effective in capturing local high-order network structures, recently leveraging motif to improve the quality of network embedding has attracted great attention [10]–[13].

In this work, we target on designing a *scalable*, parallel solution to efficiently count temporal motifs from large-scale dynamic graphs. The existing works that attempt to count temporal motifs provide either *exact* results or *approximations*. The existing exact algorithms can hardly handle large-scale network, due to their heavy computation costs. Paranjape *et al.* [1] formally define the notion of δ*-temporal motifs*, and propose an exact algorithm (EX) for counting 2- and 3-node, 3-edge, δ-temporal motifs by leveraging subgraph enumeration. Kumar and Calders [14] present an efficient algorithm called 2SCENT to find all temporal cycles in a directed interaction network. Mackey *et al.* [15] propose an efficient backtracking (BT) algorithm for temporal subgraph isomorphism, which can exactly count temporal motifs by enumerating all of them. However, because they have to enumerate all edges and circles, these works still suffer from high computation complexity, hence not scalable to big graph. For example, when counting temporal motifs on the RedditComments data with more than 600 million edges, it takes EX 7,968 seconds to find all 2- and 3-node, 3-edge, δ-temporal motifs. This high response time makes such methods insufficient in handling frequently updated dynamic systems which are very popular in practice.

To reduce response time, several sampling-based algorithms have been proposed to *approximate* the number of motifs [16], [17]. However, these approximate algorithms, either only supporting certain types of motifs such as 2-node 3-edge motifs or suffering from large approximate errors, do not meet the requirements of many applications. Furthermore, the motifs discovered by sampling fail to preserve the local structures of a graph [13]. Therefore, they are not effective when used in network embedding, which is one of the most important emerging applications of motifs.

To address the aforementioned challenges, we propose a scalable parallel framework called HARE for temporal motif counting in large-scale temporal networks. First, based on topological structure, we categorize all possible 2- and 3-node, 3-edge, δ-temporal motifs into three types: *pair temporal motifs*, *star temporal motifs* and *triangle temporal motifs*. Customized to star/pair temporal motifs, our FAST-Star algorithm uses a quadruple counter and a triple counter to compactly encode the number of star motif instances and pair motif instances, respectively. With the designed counters, FAST-Star directly identifies the types of temporal motif instances according to the information of edges and the relationship between edges, therefore, significantly improving the counting efficiency. Our algorithm FAST-Tri, customized to triangle temporal motifs, uses another quadruple counter to record the number of motif instances for the non-isomorphic temporal motifs simultaneously. FAST (general term for FAST-Star and FAST-Tri) recursively treats each node in the given temporal graph as center node, and searches all motif instances in the edge sequence of the center node, achieving a linear time complexity in the number of temporal edges of input graph. Furthermore, the recursive nature of FAST enables us to leverage multi-threading of modern CPU to count the temporal motifs in parallel. This is because in our parallel framework HARE, if different threads pick different centers, each thread will exactly count distinct motifs independently.

We conduct extensive experiments on 16 real-world large-scale temporal graphs, and the experimental results demonstrate that our HARE performs significantly faster than state-of-the-art baselines for counting temporal motifs by up to two orders of magnitudes.

We highlight the key contributions of this work as follows:

- We propose a fast exact algorithm for counting star/pair temporal motifs. The proposed FAST-Star algorithm can directly identify the types of motif instances according to the edge information and relationship between edges, improving the significant detection efficiency.
- We develop a fast algorithm for exactly counting motif instances for triangle temporal motifs. The proposed FAST-Tri algorithm can simultaneously count the number of motif instances for all non-isomorphic triangle temporal motifs with the designed quadruple counter.
- We propose a hierarchical parallel framework HARE for our proposed two exact algorithms, which endows our method with the capability of concurrently counting all temporal motifs for large-scale temporal networks in an efficient way.
- We perform extensive experiments on 16 real-world graph datasets to demonstrate the superiority of our proposed method compared with other baselines. Our proposed HARE results in up to two orders of magnitude faster than state-of-the-art techniques.

II. Related Work

A. Motif Counting in Static Graphs

There have been rich studies on network motifs in static graphs, where these works have proved crucial to understanding the mechanisms driving complex systems [18] and characterizing classes of static graphs [19], [20]. In addition, the motifs are very important for understanding the high-order organization model in the graph [21], [22]. In terms of algorithm, a variety of researches are only used to calculate triangles in undirected static graphs [23]. Ahmed *et al.* [24] propose a fast algorithm for counting motifs of $3, 4$-node that leverages a number of combinatorial arguments. It significantly improves the scalability of motif counting. Santoso *et al.* [25] propose an exact algorithm for enumerating 4-node motifs, such as 4-cycles, 4-cliques and diamonds, by leveraging the most efficient algorithm for triangle enumeration. Since many graphs are not static as the links between nodes dynamically change over time [26], the above methods fail to capture the richness of the temporal information in the data.

B. Motif Counting in Temporal Graphs

Recently, the temporal motif is no longer limited by the snapshot, but has been extended to the network motif with time information [27]. Kovanen *et al.* [28] first present the definition of temporal motif which is widely used in Wikipedia network. Gurukar *et al.* [9] propose COMMIT based on subsequence mining to identify the temporal motifs in the communication network. In [1], Paranjape *et al.* formally define the notion of *δ-temporal motifs*. They also propose exact fast algorithms for counting 2- and 3-node, 3-edge, δ-temporal motifs by leveraging subgraph enumeration in temporal graphs. Kumar and Calders [14] focus on one such fundamental interaction pattern, namely a temporal cycle, and present an efficient algorithm called 2SCENT to find all temporal cycles in a directed interaction network. Mackey *et al.* [15] propose an efficient backtracking algorithm for temporal subgraph isomorphism, which can count temporal motifs exactly by enumerating all of them. Based on the definition of communication motif in [9], Sun *et al.* [29] propose an algorithm called TM-Miner, which can build a canonical labeling system that uses a new lexicographic order and maps the temporal graph to the unique minimum time first search code, to mining temporal motifs in large temporal network.

C. Sampling Methods for Motif Counting

First of all, many sampling methods have been proposed for approximate triangle counting in static graphs, such as subgraph sampling [30], edge sampling [31], wedge sampling [32] and neighborhood sampling [33]. Bera *et al.* [34] propose a sublinear algorithm in the random walk access model to count triangles without seeing the whole static graph. Moreover, sampling methods are also efficient to find more complex motifs, *e.g.*, 4-vertex motifs [35], 5-vertex motifs [36], and k-cliques [37]. However, all above methods do not consider the temporal information, and thus they can not process motif counting in temporal graphs directly. Recently, some sampling methods are proposed to approximately count motifs in temporal graphs. Liu *et al.* [16] develop a sampling framework that sits as a layer on top of existing exact counting algorithms. Wang *et al.* [17] propose an edge sampling algorithm for any temporal motifs and hybridize edge sampling with wedge sampling to count temporal motifs with 3 nodes and 3 edges.

III. Problem Definition

In this section, we first introduce key notations used in this work and then formally define the studied problem.

Definition 1 (Temporal Graph). *A temporal graph is a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$, where $\mathcal{V}$ is the collection of nodes, $\mathcal{E}$ is the collection of edges between the nodes, and $\mathcal{T}$ is the collection of timestamps. Each edge e_{ij}^t is a timestamped directed edge from node v_i to node v_j, denoted by (v_i, v_j, t), where $v_i, v_j \in \mathcal{V}$ and $t \in \mathcal{T}$. We term each edge as a temporal edge.*

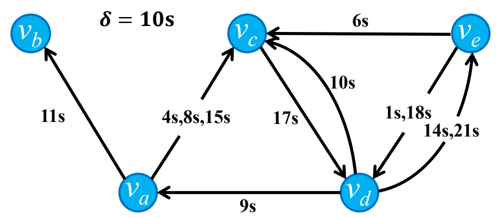


Fig. 1: An example of a temporal graph.

Fig. 1 shows a toy example of a temporal graph with 5 nodes and 12 temporal edges, and each edge is directed and has a timestamp in seconds.

Definition 2 (δ-temporal Motif). *A k-node, l-edge, δ-temporal motif is a sequence of l temporal edges in chronological order within a δ time constraint, $M = \langle(u_1, v_1, t_1), (u_2, v_2, t_2), \ldots, (u_l, v_l, t_l)\rangle$, i.e., $t_1 \le t_2 \le \cdots \le t_l$ and $t_l - t_1 \le \delta$, such that the induced static graph from these edges is connected and include k nodes.*

Definition 3 (Motif Instance). *A collection of temporal edges selected from a given temporal graph is called a motif instance of a δ-temporal motif M, if it matches the same edge pattern (i.e., same direction and time order) and all of the edges occur within δ time interval.*

In this paper, we focus on 2- and 3-node, 3-edge, δ-temporal motifs, which are considered as the most common types of motifs [22]. As shown in Fig. 2, there are 32 kinds of 3-node, 3-edge, δ-temporal motifs and 4 kinds of 2-node, 3-edge, δ-temporal motifs. These 36 kinds of δ-temporal motifs constitute the basic motifs in temporal graphs [1]. In the toy example shown in Fig. 1, the edge sequence $S = \langle(v_a, v_c, 4s), (v_a, v_c, 8s), (v_d, v_a, 9s)\rangle$ is a motif instance of temporal motif M_{63}, $S = \langle(v_e, v_c, 6s), (v_d, v_c, 10s), (v_d, v_e, 14s)\rangle$ is a motif instance of temporal motif M_{46}, and $S = \langle(v_d, v_e, 14s), (v_e, v_d, 18s), (v_d, v_e, 21s)\rangle$ is a motif instance of 2-node pair temporal motif M_{65}.

And it also allows users to specify the types of motifs to their interest in Def. 2

Problem 1 (δ-temporal Motif Counting)**.** *Give a temporal graph $\mathcal{G}$ and a time interval δ, temporal motif counting is to exactly count the number of motif instances for all 2-node and 3-node, 3-edge, δ-temporal motifs in $\mathcal{G}$.*

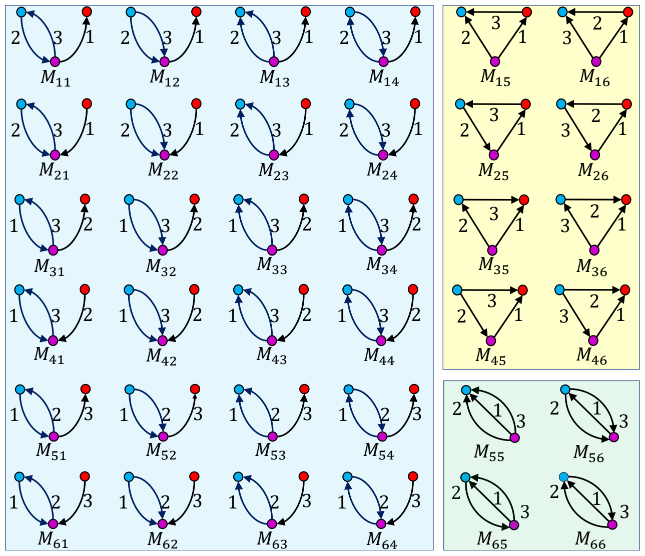


Fig. 2: All 2- and 3-node, 3-edge, δ-temporal motifs.

Key notations used in the paper are summarized in Table I.

TABLE I: Main notations and their definitions.

Notation	Definition
$\mathcal{G}$	input temporal network
$\mathcal{V}, \mathcal{E}, \mathcal{T}$	node/edge/timestamp set of $\mathcal{G}$
(v_i, v_j, t)	an edge from v_i to v_j with timestamp t
M_{ij}	a δ-temporal motif
u	center node
$e = (t, v, dir)$	an edge *w.r.t.* current center node u
$e.t, e.v, e.dir$	timestamp/node on the other side/direction of e
S_u	the edge sequence of center node u, sorted by time
$Star[\cdot,\cdot,\cdot,\cdot]$	the quadruple counter for star temporal motifs
$Pair[\cdot,\cdot,\cdot]$	the triple counter for pair temporal motifs
$Tri[\cdot,\cdot,\cdot,\cdot]$	the quadruple counter for triangle temporal motifs
δ	the time constraint for δ-temporal motif
thr_d	the degree threshold for hierarchical parallel framework

IV. Methodology

In this section, we propose two fast algorithms for exactly counting motif instances of all 2- and 3-node, 3 edge, δ-temporal motifs in a given temporal graph. According to topological structures, we first divide all 2- and 3-node, 3-edge, δ-temporal motifs (in Fig. 2) into three categories: *pair temporal motifs* (with green background), *star temporal motifs* (with blue background) and *triangle temporal motifs* (with yellow background). Pair temporal motifs only include 2 nodes with 3 temporal edges. Star and triangle temporal motifs contain 3 nodes with 3 temporal edges forming star structure and triangle structure respectively. Considering the structural similarity between pair temporal motifs and star temporal motifs, we use an unified fast exact algorithm to count both of them.

A. Proposed Method for Star and Pair Temporal Motifs

In fact, there are 4 non-isomorphic pair temporal motifs and 24 non-isomorphic star temporal motifs (in Fig. 2). We first focus on the 24 non-isomorphic star temporal motifs. In each kind of star temporal motif, we denote the node with the largest degree (connected with 3 edges) as the center node u. Each edge connected to center node u in graph $\mathcal{G}$ can be defined by: (i) the timestamp t of the edge, (ii) another node v linked to the edge, and (iii) the direction dir *w.r.t.* center node u (outward from or inward to u). That is, each edge connected to center node u can be expressed as $e = (t, v, dir)$.

1) Three Types of Star Temporal Motifs: Star temporal motifs are not centrosymmetric – 2 edges both connect to two nodes while one isolated edge connects to another node. If we ignore the directions of edges, but only consider the time order of edges, we can divide these 24 kinds of star temporal motifs into three types based on the time order of the isolated edge (see Fig. 3):

- **Star-I**: The first edge in time order is isolated and connects to one node. The second and third edges both connect to another node.
- **Star-II**: The second edge in time order is isolated and connects to one node. The first and third edges both connect to another node.
- **Star-III**: The third edge in time order is isolated and connects to one node. The second and third edges both connect to another node.

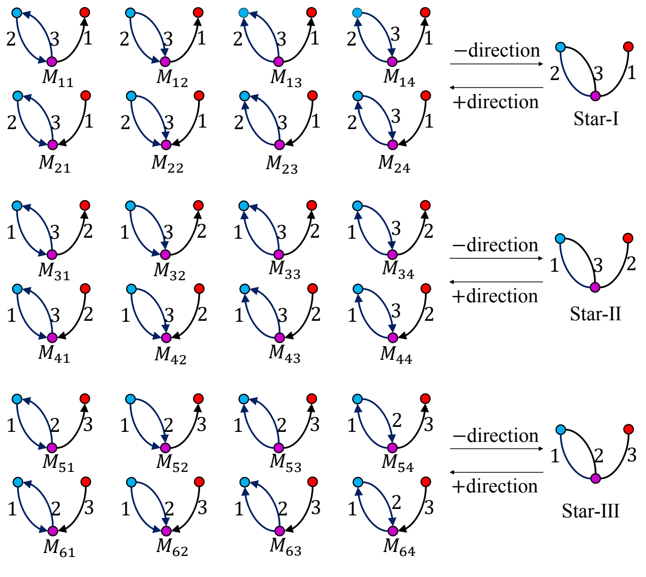

M_{11} M_{12} M_{13} M_{14}
M_{21} M_{22} M_{23} M_{24}
−direction
+direction
Star-I
M_{31} M_{32} M_{33} M_{34}
M_{41} M_{42} M_{43} M_{44}
Star-II
M_{51} M_{52} M_{53} M_{54}
M_{61} M_{62} M_{63} M_{64}
Star-III

Fig. 3: Types of star temporal motifs.

2) Quadruple Counter for Star Temporal Motifs: We now introduce a quadruple counter $Star[Type, dir_1, dir_2, dir_3]$ for counting the number of motif instances of all star temporal motifs, which considers both motif type and directions of three edges. Specifically, the first dimension (*i.e.*, $Type$) indicates the type of star temporal motif (*i.e.*, Star-I, Star-II, or Star-III). The next three dimensions, *i.e.*, dir_1, dir_2 and dir_3, represent the direction of first edge, second edge, and third edge in each motif type, respectively. Notice that each edge has two direction options: outward from or inward to its center node u, which is denoted by o for outward or in for inward respectively. Therefore, this quadruple counter can record $3 \times 2 \times 2 \times 2 = 24$ kinds of motifs, which correspond to 24 non-isomorphic star temporal motifs in Fig. 2. For example, $Star[\text{I}, in, o, in]$ records the number of motif instances of M_{24}, because edge pattern of M_{24} follows that of Star-I, and the first and third edges all link inward to the center node, while the second edge moves outward from center node.

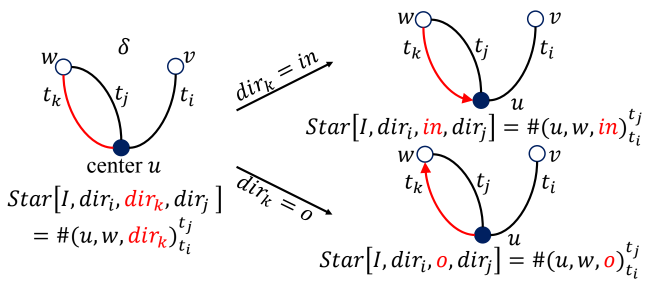
3) Fast Counting Algorithm for Star and Pair Temporal Motifs: For counting all motif instances of all star and pair temporal motifs, we treat each node in the given temporal graph as the center node u recursively. Specifically, for each node u, we list its all linked edges in time order as $S_u = \langle (t_1, v_1, dir_1), (t_2, v_2, dir_2), \ldots, (t_s, v_s, dir_s) \rangle$, where $t_1 \le t_2 \le \cdots \le t_s$. We aim to detect all motif instances from the edge sequence S_u *w.r.t.* center node u. Intuitively, every edge in $S - \langle (t_{s-1}, v_{s-1}, dir_{s-1}), (t_s, v_s, dir_s) \rangle$ can be regraded as the first edge of a star temporal motif. Suppose that (t_i, v_i, dir_i) is currently selected as the first edge for a star temporal motif, then (t_j, v_j, dir_j) is selected as the third edge one by one from t_i until $t_j - t_i \ge \delta$. In each process, by scanning all edges between the first edge and the third edge, the type of star temporal motif can be determined, and the corresponding number of motif instances can also be recorded.

Next, we elaborate the details of our proposed FAST-Star algorithm for counting all star and pair temporal motifs. Given an edge sequence S_u *w.r.t.* the center node u, we now choose $e_1 = (t_i, v, dir_i) \in S_u, e_3 = (t_j, w, dir_j) \in S_u$ $(t_j - t_i \le \delta)$ as the first edge and third edge respectively. Let $e_2 = (t_k, x, dir_k)$ $(t_i \le t_k \le t_j)$ be the second edge candidate.

We first discuss the different cases for temporal motif counting according to the choose of the second edge:

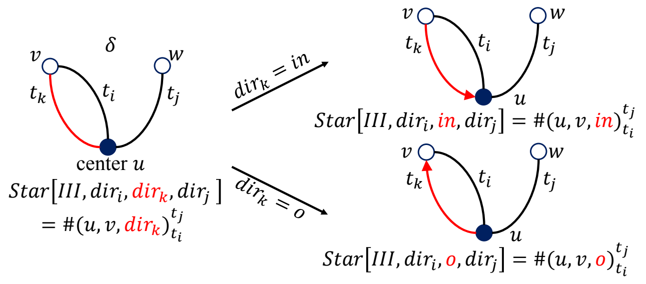
(a) If $v \ne w$ (*i.e.*, $e_1.v \ne e_3.v$), only the edges connected to v or w can be selected as the second edge candidates to form a star temporal motif. According to types of star temporal motifs shown in Fig. 3, if $x = w$, then the formed motif belongs to **Star-I**, and if $x = v$, it belongs to **Star-III**. Specifically, the specific kind of star temporal motif depends on the exact directions of the three edges. The number of motif instances for each specific motif kind is equal to the number of second edge candidates with same x and dir_k. As shown in Fig. 4 and Fig. 5, the number of motif instances $Star[\text{I/III}, dir_i, dir_k, dir_j]$ is equal to the number of the second edge candidates $\#(u, w/v, dir_k)_{t_i}^{t_j}$, where $\#(u, w/v, dir_k)_{t_i}^{t_j}$ denotes the number of edges between center node u and w/v with direction dir_k *w.r.t.* u and timestamp $t_k \in [t_i, t_j]$.

(b) If $v = w$, all edges between the first edge and the third



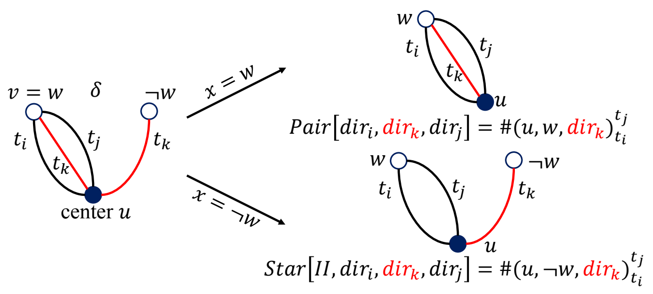
δ
w
v
t_k
t_j
t_i
center u
$Star[I, dir_i, dir_k, dir_j] = \#(u, w, dir_k)_{t_i}^{t_j}$
$dir_k = in$
$dir_k = o$
$Star[I, dir_i, in, dir_j] = \#(u, w, in)_{t_i}^{t_j}$
$Star[I, dir_i, o, dir_j] = \#(u, w, o)_{t_i}^{t_j}$

Fig. 4: Counting Star-I motifs by the second edge.



δ
v
w
t_k
t_i
t_j
center u
$Star[III, dir_i, dir_k, dir_j] = \#(u, v, dir_k)_{t_i}^{t_j}$
$dir_k = in$
$dir_k = o$
$Star[III, dir_i, in, dir_j] = \#(u, v, in)_{t_i}^{t_j}$
$Star[III, dir_i, o, dir_j] = \#(u, v, o)_{t_i}^{t_j}$

Fig. 5: Counting Star-III motifs by the second edge.



$v = w$
δ
$\neg w$
t_i
t_k
t_j
center u
$x = w$
$x = \neg w$
$Pair[dir_i, dir_k, dir_j] = \#(u, w, dir_k)_{t_i}^{t_j}$
$Star[II, dir_i, dir_k, dir_j] = \#(u, \neg w, dir_k)_{t_i}^{t_j}$

Fig. 6: Counting Star-II and Pair Temporal Motifs.

edge can be chose as the second edge to form a temporal motif. It is worth noting that when $x = w = v$, the currently formed motif is a **pair temporal motif**. In other cases, the formed motifs belong to **Star-II**. Similarly, the specific kind of pair/star temporal motif depends on the directions of the three edges, and the number of motif instances is equal to the number of second edge candidates. Following the counter $Star[Type, dir_1, dir_2, dir_3]$, we use a triple counter $Pair[dir_1, dir_2, dir_3]$ to record the number of motif instances for pair temporal motifs. As shown in Fig. 6, the number of pair motif instances $Pair[dir_i, dir_k, dir_j]$ is equal to the number of the second edge candidates $\#(u, w, dir_k)_{t_i}^{t_j}$, and the number of star motif instances $Star[\text{II}, dir_i, dir_k, dir_j]$ is equal to the number of the second edge candidates $\#(u, \neg w, dir_k)_{t_i}^{t_j}$, where $\#(u, \neg w, dir_k)_{t_i}^{t_j}$ denotes the number of edges between node u and other nodes except w with direction dir_k *w.r.t.* u and timestamp $t_k \in [t_i, t_j]$.

Notice that there are only 4 kinds of non-isomorphic pair temporal motifs, while here we count the number of motif instances for 8 kinds of pair temporal motifs in counter $Pair[\cdot, \cdot, \cdot]$. This is because each pair of them are isomorphic, *i.e.*, $Pair[in, in, in] \cong Pair[o, o, o] \cong M_{55}$, $Pair[in, o, o] \cong Pair[o, in, in] \cong M_{56}$, $Pair[in, o, in] \cong Pair[o, in, o] \cong M_{65}$, and $Pair[in, o, in] \cong Pair[o, in, o] \cong M_{66}$.

Taking the temporal graph shown in Fig. 1 as an example, suppose that $\delta = 10$ seconds and node v_a is selected as the center node u, the edge sequence of node v_a is $S_a = \langle (4s, v_c, o), (8s, v_c, o), (9s, v_d, in), (11s, v_b, o), (15s, v_c, o) \rangle$.

First, let $e_1 = S_a[1] = (4s, v_c, o)$ and $e_3 = S_a[3] = (9s, v_d, in)$. So the second edge candidate can only choose $S_a[2] = (8s, v_c, o)$. Since $e_1.v \neq e_3.v$ (*i.e.*, $v_c \neq v_d$) and $e_2.v = v_c = e_1.v$, the currently formed motif is a Star-III motif, hence $Star[\text{III}, o, o, in]$ += 1. Next, e_1 remains unchanged, and let $e_3 = S_a[4] = (11s, v_b, o)$. Since $e_1.v \neq e_3.v$, there is only one candidate for the second edge, *i.e.*, $(8s, v_c, o)$. This is because only the edges connected to $e_1.v$ (*i.e.*, v_c) or $e_3.v$ (*i.e.*, v_b) can be selected as the second edge candidates to form 3-node star motif. According to the second edge $((8s, v_c, o))$, the formed motif is also a Star-III motif, and thus $Star[\text{III}, o, o, o]$ += 1. Then, e_1 remains unchanged, and let $e_3 = S_a[5] = (15s, v_c, o)$. Because $15s - 4s = 11s > \delta$, $(15s, v_c, o)$ and the edges after it can no longer be chosen as the third edge *w.r.t.* the current e_1.

Therefore, we need to move the first edge e_1 to $S_a[2] = (8s, v_c, o)$, and select edge $S_a[4] = (11s, v_b, o)$ as the initial third edge e_3. The traversal process of the second edge is the same as above. Since edge $S_a[3] = (9s, v_d, in)$ does not meet the requirements of the node (*i.e.*, v_c or v_b), it can not constitute a motif with the current e_1 and e_3. Next, keep e_1 unchanged, and move e_3 to $S_a[5] = (15s, v_c, o)$. Since $e_1.v = e_3.v = v_c$, all edges between e_1 and e_3 can be selected as the second edge. Thus, the formed motifs by $(9s, v_d, in)$ and $(11s, v_b, o)$ are both Star-II motifs, and $Star[\text{II}, o, in, o]$ += 1 and $Star[\text{II}, o, o, o]$ += 1. So far, the star and pair temporal motif counting *w.r.t.* the node v_a is over.

In this way, we treat each node as the center node in turn, and then we can count all motif instances for all star temporal motifs as well as pair temporal motifs.

4) Complexity Analysis: Algorithm 1 shows the pseudo-code of the proposed FAST-Star for counting all kinds of star and pair temporal motifs.

As depicted in lines 7-28 in Algorithm 1, instead of traversing all edges between the first edge and the third edge for the choice of the second edge, we leverage the traversal of the third edge to realize the scan of the second edge, which significantly reduces the computation cost. More specifically, in the process of determining the third edge we use two HashMaps (*i.e.*, m_{in} and m_{out}) to simultaneously record the direction and the number of all the second edge candidates, and thus we no longer need to traverse all the edges again between the first and the third edges. Since m_{in} and m_{out} stores the number of edges connecting different nodes between the first and third edges (m_{in} and m_{out} respectively records two different directions relative to center node u), the number of the second edge candidates (*i.e.*, $\#(u, v, in/o)_{t_i}^{t_j}$, $\#(u, w, in/o)_{t_i}^{t_j}$) can be queried in m_{in} and m_{out} only based on the first edge and the third edge. Notice that m_{in} and m_{out}) always are initialized to be empty at the beginning of each iteration.

Algorithm 1 FAST algorithm for Star/Pair Temporal Motifs

```
Input: Temporal graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$, and time constraint $\delta$.
Output: Star counter $Star[\cdot,\cdot,\cdot,\cdot]$, pair counter $Pair[\cdot,\cdot,\cdot]$.
1:  for each node $u \in \mathcal{V}$ do
2:      get $S_u = \langle (t_1, v_1, dir_1), (t_2, v_2, dir_2), \ldots, (t_s, v_s, dir_s) \rangle$;
3:      for $i = 1$ to $s - 2$ do
4:          $e_1 \leftarrow S_u[i]$;
5:          $\#e_{in} \leftarrow 0$; $\#e_{out} \leftarrow 0$;
6:          $m_{in}, m_{out} \leftarrow HashMap.Init()$;
7:          for $j = i + 1$ to $s$ do
8:              $e_3 \leftarrow S_u[j]$;
9:              if $e_3.t - e_1.t > \delta$ then
10:                 break;
11:             end if
12:             if $e_3.v == e_1.v$ then
13:                 $Pair[dir_i, in, dir_j]$ += $m_{in}[e_1.v]$;
14:                 $Pair[dir_i, o, dir_j]$ += $m_{out}[e_1.v]$;
15:                 $Star[\text{II}, dir_i, in, dir_j]$ += $\#e_{in} - m_{in}[e_1.v]$;
16:                 $Star[\text{II}, dir_i, o, dir_j]$ += $\#e_{out} - m_{out}[e_1.v]$;
17:             else
18:                 $Star[\text{I}, dir_i, in, dir_j]$ += $m_{in}[e_3.v]$);
19:                 $Star[\text{I}, dir_i, o, dir_j]$ += $m_{out}[e_3.v]$);
20:                 $Star[\text{III}, dir_i, in, dir_j]$ += $m_{in}[e_1.v]$);
21:                 $Star[\text{III}, dir_i, o, dir_j]$ += $m_{out}[e_1.v]$);
22:             end if
23:             if $e_3.dir == i$ then
24:                 $m_{in}[e_3.v]$ += 1; $\#e_{in}$ += 1;
25:             else
26:                 $m_{out}[e_3.v]$ += 1; $\#e_{out}$ += 1;
27:             end if
28:         end for
29:     end for
30: end for
31: return $Star[\cdot,\cdot,\cdot,\cdot], Pair[\cdot,\cdot,\cdot]$;
```

We now analyze the time complexity of our proposed FAST-Star for counting star and pair temporal motifs. We suppose that the degree of the i-th node in the temporal graph is d_i, and the average number of edges connected to the i-th node within time interval δ is d_i^δ. For each node, the worst time complexity of the counting process is $O(d^2)$. Therefore, the worst time complexity for whole graph is $O(\sum_{i=1}^{|\mathcal{V}|}(d_i^2))$.

If we take time constraint δ into consideration, then the traversal space of the third edge is reduced to d_i^δ. Therefore, the time complexity of counting process for i-th node becomes $O(d_i d_i^\delta)$, and the worst time complexity for whole graph is $O(\sum_{i=1}^{|\mathcal{V}|}(d_i d_i^\delta))$. Since $d_i^\delta << d_i$, assuming that d_i^δ of all nodes is approximately equal, denoted by d^δ, thus $O(\sum_{i=1}^{|\mathcal{V}|}(d_i d_i^\delta)) \approx O(d^\delta \sum_{i=1}^{|\mathcal{V}|}(d_i)) \approx O(2d^\delta|\mathcal{E}|)$, where $|\mathcal{E}|$ is the number of temporal edges in graph $\mathcal{G}$. Namely, our FAST-Star algorithm achieves the time complexity linear in the number of temporal edges of the input graph.

B. *Proposed Method for Triangle Temporal Motifs*

In this section, we present the details of our proposed exact counting algorithm FAST-Tri for counting the number of motif instances for all triangle temporal motifs.

1) Three Types of Triangle Temporal Motifs: To identify potential triangle motifs, we first choose a node in graph $\mathcal{G}$ as the center node u, and then determine two different temporal edges connected to center node u such that the other two connected nodes are different. We denote these two temporal edges as $e_i = (t_i, v, dir_i)$ and $e_j = (t_j, w, dir_j)$, where $t_j - t_i \leq \delta$, and dir_i and dir_j are the directions of e_i and e_j *w.r.t.* center node u, respectively. Each edge between v and w and the two edges above may form triangle temporal motifs, depending on whether the time constraint δ is satisfied. Therefore, we next consider each edge e_k between nodes v and w that satisfy the time constrain δ. Let t_k be the timestamp of edge e_k and dir_k be the direction of edge e_k *w.r.t.* node v (*i.e.*, o indicates from v to w and in denotes from w to v). Unlike FAST-Star, the order of three edges is determined by their timestamps. Then according to the order of three edges, we determine the types of triangle temporal motif and use corresponding counter to count the triangle temporal motif.

According to the time order of three edges (*i.e.*, t_k, t_i and t_j), we divide all kinds of triangle temporal motifs into three types (see Fig. 7):

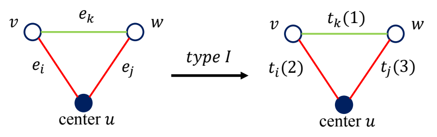
- **Triangle-I**: If $t_k < t_i$ and $t_j - t_k \leq \delta$, then the formed triangle temporal motif belongs to Triangle-I type motif.
- **Triangle-II**: If $t_i \leq t_k \leq t_j$, then the formed triangle temporal motif belongs to Triangle-II type motif.
- **Triangle-III**: If $t_j < t_k$ and $t_k - t_i \leq \delta$, then the formed triangle temporal motif belongs to Triangle-III type motif.

2) Fast Counting Algorithm for Triangle Temporal Motifs: Considering the directions of the three edges for each type of triangle temporal motifs, we also introduce a quadruple counter $Tri[Type, dir_i, dir_j, dir_k]$ for counting the number of motif instances for all types of triangle temporal motifs. Notice that there are $3 \times 2 \times 2 \times 2 = 24$ counters in $Tri[Type, dir_i, dir_j, dir_k]$, while we have only 8 non-isomorphic triangle temporal motifs (see Fig. 2). Namely, there exit some triangle temporal motifs recorded in $Tri[Type, dir_i, dir_j, dir_k]$ are isomorphic.

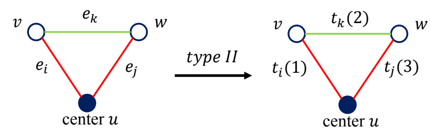
As shown in Fig. 8, the triangle temporal motifs corresponding to every three counters in $Tri[Type, dir_i, dir_j, dir_k]$ are isomorphic. Therefore, we only need to merge these isomorphic counters at the end for outputting the final result.

Next, we use a specific example to illustrate how our algorithm counts triangle temporal motifs with time constraint δ=10 seconds. Suppose that the current center node is node e in the temporal graph shown in Fig. 1. We already have the edge sequence $S_e = \langle (1s, v_d, o), (6s, v_c, o), (14s, v_d, in), (18s, v_d, o), (21s, v_d, in) \rangle$ when counting the star/pair temporal motifs.

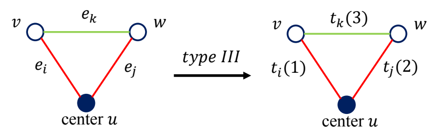
We first select $S_e[1]$ (*i.e.*, $(1s, v_d, o)$) as e_i, thus edge e_j should meet two constraints: (i) $t_j - t_i \leq \delta$, and (ii) $e_j.v \neq e_i.v$. Therefore, only $S_e[2] = (6s, v_c, o)$ is qualified candidate for e_j. If $S_e[2]$ is selected as e_j, we then get the edge set



(a) Triangle-I



(b) Triangle-II



(c) Triangle-III

Fig. 7: Types of triangle temporal motifs.

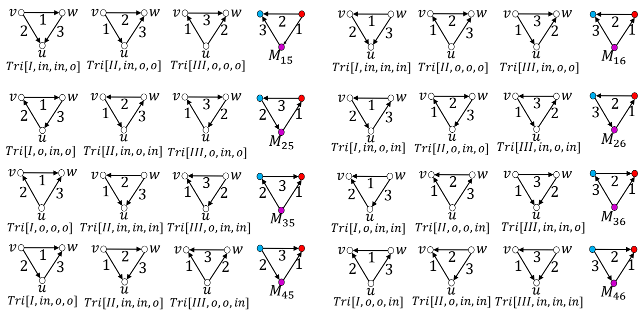


Fig. 8: Illustration of isomorphic triangle temporal motifs.

between v_c and v_d: $E_{(v_c,v_d)} = \{(v_d, v_c, 10s), (v_c, v_d, 17s)\}$. Since $10s - t_i = 9s < \delta$ and $17s - t_i = 16s > \delta$, only $(v_d, v_c, 10s)$ can be selected as e_k to form a Triangle-III motif, and thus $Tri[\text{III}, o, o, o]$ += 1.

Next, e_i moves to $S_e[2]$, thus only $S_e[3] = (14s, v_d, in)$ is qualified candidate for e_j. If $e_j = S_e[3]$, only edge $(v_d, v_c, 10s) \in E_{(v_c,v_d)}$ can be chosen as e_k to form a Triangle-II motif, and thus $Tri[\text{II}, o, in, o]$ += 1. Among the remaining edges, no edge can be selected for e_i, because there is no qualified e_j to be selected for each potential e_i. Now we find out all triangle temporal motifs with node v_e as center node.

3) Complexity Analysis: Algorithm 2 shows the pseudo-code of our proposed FAST-Tri for counting all triangle temporal motifs.

Notice that each triangle temporal motif instance is counted three times repeatedly, because each instance can form three different types of triangle temporal motifs *w.r.t.* its three vertices, respectively. For example, edge sequence $\langle(v_a, v_c, 8s), (v_d, v_a, 9s), (v_c, v_d, 17s)\rangle$ in Fig. 1 forms an instance of triangle temporal motif M_{25}. When v_a is selected as the center node, this instance can be identified as an instance recorded in $Tri[\text{III}, o, in, o]$. When v_c is selected as the center node, this instance will be detected as an instance recorded in $Tri[\text{II}, in, o, in]$. When v_d is selected as the center node, this instance is recognized as an instance recorded in $Tri[\text{I}, o, in, o]$. In fact, the counters $Tri[\text{III}, o, in, o]$, $Tri[\text{II}, in, o, in]$ and $Tri[\text{I}, o, in, o]$ are isomorphic. Therefore, as shown in Fig. 8, each triangle temporal motif can be identified as three distinct triangle types based on its three different vertices respectively. That is, each triangle temporal motif is counted three times.

However, it is easy to handle this issue in a single thread. That is, when the number of motif instances *w.r.t.* center node u has been counted, center node u is removed from $\mathcal{V}$, including all connected edges, to avoid redundant counting. In this work, we target on a counting framework that is natively parallel. To achieve this, we have to avoid any dependency among different processes/threads, while eliminating the repeated counting will inevitably introduce dependency. Therefore, in multi-threading environment, to avoid any dependency among different processes/threads, we do not do any processing, but repeatedly count and finally divide by three.

Algorithm 2 FAST algorithm for Triangle Temporal Motifs

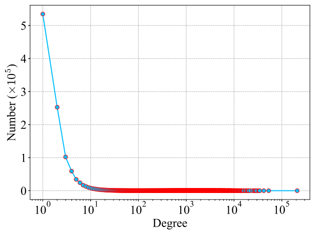
```
Input: Temporal graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$, and time constraint $\delta$.
Output: Triangle counter $Tri[\cdot,\cdot,\cdot,\cdot]$.
1: for each node $u \in \mathcal{V}$ do
2:     get $S_u = \langle(t_1, v_1, dir_1), (t_2, v_2, dir_2), \ldots, (t_s, v_s, dir_s)\rangle$;
3:     for $i = 1$ to $s-1$ do
4:         $e_i \leftarrow S_u[i]$;
5:         for $j = i+1$ to $s$ do
6:             $e_j \leftarrow S_u[j]$;
7:             if $e_j.t > e_i.t + \delta$ then
8:                 break;
9:             end if
10:            if $e_j.v == e_i.v$ then
11:                continue;
12:            end if
13:            for each edge $e \in E_{(e_i.v, e_j.v)}$ do
14:                if $t_j - \delta \le e.t < t_i$ then
15:                    $Tri[\text{I}, dir_i, dir_j, e.dir]++$;
16:                else if $t_i \le e.t \le t_j$ then
17:                    $Tri[\text{II}, dir_i, dir_j, e.dir]++$;
18:                else if $t_j < e.t \le t_i + \delta$ then
19:                    $Tri[\text{III}, dir_i, dir_j, e.dir]++$;
20:                else if $e.t > t_i + \delta$ then
21:                    break;
22:                end if
23:            end for
24:        end for
25:    end for
26:    $\mathcal{V} \leftarrow \mathcal{V} - \{u\}$; $\mathcal{E} \leftarrow \mathcal{E} - \{e_u\}$; //Only for single threading
27: end for
28: return $Tri[\cdot,\cdot,\cdot,\cdot]$
```

We then analyze the time complexity of FAST-Tri algorithm. Similarly, we use d_i to denote the degree of the i-th node and d_i^δ to represent the average number of edges within δ for the i-th node. For each center node, the worst time complexity of searching two edges for constructing potential triangle motifs is $O(d_i d_i^\delta)$. In the worst case, every combination of two edges can form a potential triangle motif. Hence, the worst time complexity for one node is $O(d_i d_i^\delta \xi)$, where ξ denotes the average number of edges between two nodes. For the whole graph, the worst time complexity is $O(\sum_{i=1}^{|\mathcal{V}|} d_i d_i^\delta \xi)$. Using some implementation tricks, ξ can be reduced to the number of edges between two nodes within δ time interval, *i.e.*, $\xi \leq d_i^\delta$. Therefore, the worst time complexity for the whole graph is less than $O(\sum_{i=1}^{|\mathcal{V}|} d_i (d_i^\delta)^2)$.

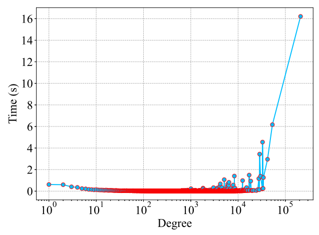
Because $d_i^\delta << d_i$, we assume that d_i^δ of all nodes is approximately equal, denoted by d^δ, thus $O(\sum_{i=1}^{|\mathcal{V}|} d_i (d_i^\delta)^2) \approx O((d^\delta)^2 \sum_{i=1}^{|\mathcal{V}|}(d_i)) \approx O(2(d^\delta)^2|\mathcal{E}|)$. That is, our FAST-Tri also achieves linear time complexity with the input graph.

C. Hierarchical Parallel Framework

As described in Algorithm 1 and Algorithm 2, our algorithms recursively tread each node in temporal graph as center node to detect all motif instances, which have no direct data dependency. Therefore, our proposed FAST (general term for FAST-Star and FAST-Tri) naturally has high parallelism, namely, our FAST converts the temporal motif counting into an embarrassingly parallel problem. Nevertheless, we find that simply employing multi-threading do not achieve the desired effect (*e.g.*, approximately linear speedup) on same graph datasets. This is because the degree distribution of the nodes in most temporal graphs is extremely unbalanced, which leads to the load unbalanced problem of multi-threading. Even if some balanced load strategies are considered at the node level (*e.g.*, dynamic schedule), the expected results cannot be achieved. Fig. 9 shows the degree distribution of all nodes in *WikiTalk* graph, and the average time consumption of counting all motif instances required per node with corresponding degree. We observe that although the degree distribution of the graph has a typical long-tailed distribution, the few nodes with higher degrees (*e.g.*, top ten node) account for the dominant part of the time consumption of whole graph. Fortunately, from Algorithm 1 and Algorithm 2, it is not difficult to see that temporal motif counting process inside the node in our FAST also has high parallelism.



(a) Degree distribution of nodes



(b) Average run-time per node

Fig. 9: Data statistics on WikiTalk.

To address the above issue, we propose a Hierarchical pArallel fRamEwork (HARE) for the proposed FAST to accelerate the counting of temporal motifs powerfully. Specifically, our framework consists of two parallel strategies: *inter-node parallel* and *intra-node parallel* mechanisms. More specifically, since our proposed FAST has inherent parallelism both within and among nodes, we implement these two parallel mechanisms for our HARE based on OpenMP.

Inter-node parallel: because the temporal motif counting of different nodes are independent of each other, we can assign one or more nodes to a thread, and assign all nodes to different threads to realize parallel motif counting for all nodes. In addition, we use the dynamic scheduling mode provided by OpenMP to effectively solve the problem of load imbalance.

TABLE II: Basic statistics of twelve temporal networks.

Dataset	#nodes	#temporal edges	Time span (day)
Email-Eu	986	332,334	803
CollegeMsg	1,899	20,296	193
Bitcoinotc	5,881	35,592	1,903
Bitcoinalpha	3,783	24,186	1,901
Act-mooc	7,143	411,749	29
SMA-A	44,090	544,817	338
FBWALL	45,813	855,542	1,591
MathOverflow	24,818	506,550	2,350
AskUbuntu	159,316	964,437	2,613
SuperUser	194,085	1,443,339	2,773
Rec-MovieLens	283,228	27,753,444	1,128
WikiTalk	1,140,149	7,833,140	2,320
StackOverflow	2,601,977	63,497,050	2,774
IA-online-ads	15,336,555	15,995,634	2,461
Soc-bitcoin	24,575,382	122,948,162	2,584
RedditComments	8,036,164	613,289,746	3,686

Intra-node parallel: inside each node, the motif counting process can be divided into subtasks according to different starting edges. In fact, data reading and writing operations within the node are more vulnerable to multi-threading, which can result in unpredictable results. To ensure parallelism within the node, we use the reduction tool provided by OpenMP to copy the variables within node, so that each thread keeps the backup of these variables, and then reduce and output the final result after all tasks are completed.

In particular, we set a degree threshold thr_d for our hierarchical parallel framework. For the nodes with a degree greater than thr_d, we will enable intra-node parallel mode, otherwise, perform inter-node parallel strategy.

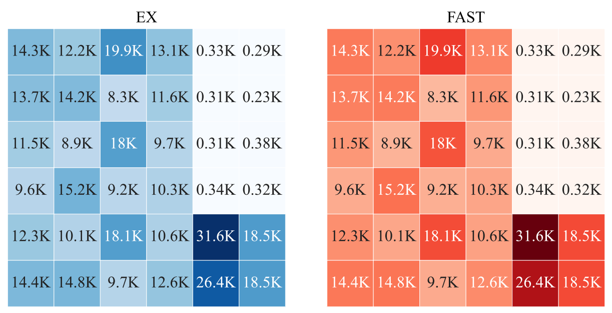
It is worth noting that HARE is able to directly use OpenMP tool to handle load imbalance in both inter-node and intra-node parallel modes, because we design HARE to be natively parallel. There is no any dependency among different threads.

V. Experiment

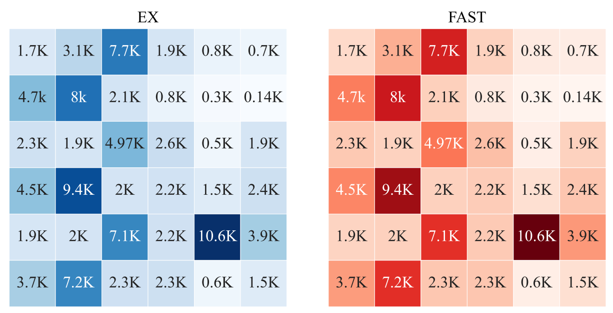
A. Datasets

We conduct extensive experiments on sixteen real-world temporal networks, which are available publicly in [38] [39].

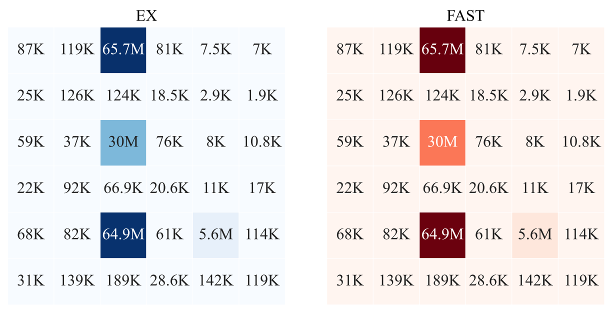
- **Email-Eu** is a collection of internal email records from a European research institution. An edge (u, v, t) signifies that person u sent person v an email at time t.
- **CollegeMsg** is a network of private messages sent on an online social network at the University of California, Irvine.
- **Bitcoinotc** and **Bitcoinalpha** are Bitcoin OTC and Bitcoin Alpha web of trust network respectively. An edge (u, v, t) in these datasets denotes that bitcoin was transferred from address u to address v at time t.
- **Act-mooc** is a collection of student actions on a popular MOOC platform. The actions are represented as a directed, temporal network.



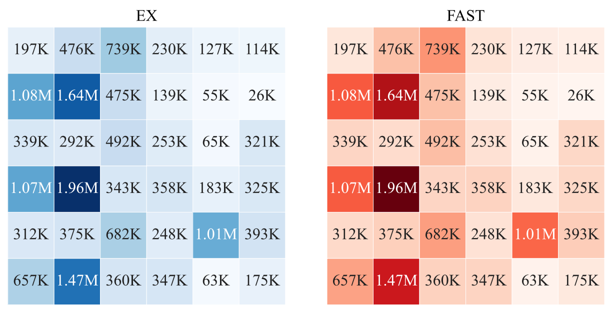
(a) CollegeMsg



(b) Superuser



(c) WikiTalk



(d) StackOverflow

Fig. 10: Counts of motif instances of all 2-node and 3-node, 3-edge δ-temporal motifs with $\delta = 600s$. For each dataset, counts in the i-th row and j-th column is the number of motif instances of M_{ij}. The color for motif M_{ij} indicates the fraction over all M_{ij} on a linear scale – darker blue/red means a higher count.

- **SMS-A** is a texting service provided on mobile phones. In this dataset, an edge (u, v, t) means that person u sent an SMS message to person v at time t.
- **FBWALL** is derived from the social network Facebook located in the New Orleans region, where the edges are wall posts between users.
- **MathOverflow**, **Askubuntu**, **Superuser** and **StackOverflow** 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.
- **Rec-MovieLens** is a rating dataset from the MovieLens website, where an edge (u, v, t) signifies that user u rated the movie v at time t.
- **WikiTalk** is a network of Wikipedia users making edits on each others' "talk pages". An edge (u, v, t) indicates that user u edited user v's talk page at time t.
- **IA-online-ads** contains information about the product related advertisements a user has clicked. An edge (u, v, t) indicates that user u clicked on advertisement v at t.
- **Soc-bitcoin** is a large-scale bitcoin transaction network where each edge (u, v, t) denotes that bitcoin was transferred from address u to address v at time t.
- **RedditComments** is constructed from a large collection of comments made by users on a popular social media platform https://www.reddit.com. An edge (u, v, t) indicates that a comment from user u to user v at time t.

Specifically, we collect seven large-scale datasets with more than one millions of temporal edges to better analyze the performance of our framework, especially, Soc-bitcoin and RedditComments have more than 100M temporal edges. The detailed statistics of these datasets are summarized in Table II.

B. Baselines

We compare our algorithms against the following baselines:

- **EX** [1] – An exact algorithm for counting all 2-node and 3-node, 3 edge temporal motifs. This algorithm is most relevant to our problem and is our main competitor in some experiments.
- **2SCENT** [14] – An algorithm for enumerating simple temporal cycle, *i.e.*, triangle temporal motifs.
- **BT** [15] – A backtracking algorithm for temporal subgraph isomorphism. Since there is no public code, we use the BT algorithm implemented by [16] for counting pair temporal motifs.
- **BTS** [16] – An approximate algorithm based on interval sampling for temporal motif counting with exact algorithm **BT** used as a subroutine.
- **EWS** [17] – An approximate algorithm based on edge and wedge sampling for counting temporal motifs with 3 nodes and 3 edges.

Notice that we use *-Pair and *-Tri to denote the variants of * for counting pair temporal motifs and triangle temporal motifs, respectively.

C. Experimental Setting

All experiments are conducted on a server running Ubuntu 18.10 with 40-core 2.30GHz Intel Xeon E5-2650 v3 processor

TABLE III: Running time in seconds of all algorithms on all temporal networks. $\delta = 600$s and $\#threads = 1$.

Dataset	EX	EWS	FAST		BT-Pair	BTS-Pair	FAST-Pair		2SCENT-Tri	FAST-Tri	
	Time (s)	Time (s)	Time (s)	speedup	Time (s)	Time (s)	Time (s)	speedup	Time (s)	Time (s)	speedup
Email-Eu	0.4739	0.3685	0.4324	1.1x	0.6903	0.1754	0.0679	10.1x	11.2839	0.1885	59.8x
CollegeMsg	0.0847	0.0621	0.0560	1.5x	0.1604	0.0312	0.0186	8.6x	1.4527	0.0218	66x
Bitcoinotc	0.1147	0.0280	0.0170	6.5x	0.1108	0.0037	0.0054	20.5x	1.1021	0.0074	148x
Bitcoinalpha	0.0703	0.0195	0.0113	6.2x	0.0772	0.0106	0.0051	15.1x	0.75487	0.0046	164x
Act-mooc	2.9573	0.6978	1.4810	1.9x	0.6823	0.3587	0.2032	3.4x	3.5363	0.8942	3.9x
SMS-A	0.4060	0.5008	0.1956	2.1x	0.8308	0.1262	0.0925	8.9x	4.6176	0.0351	131x
FBWALL	1.2045	0.7607	0.1339	8.8x	1.0071	0.1173	0.0583	17.3x	6.2523	0.0578	108x
MathOverflow	2.5498	0.3468	0.1013	25x	1.2335	0.0952	0.0180	68.5x	3.2846	0.0426	77x
Askubuntu	3.0907	0.6341	0.2681	11x	2.4465	0.2393	0.0562	43.5x	5.9988	0.1750	34x
Superuser	5.6000	0.9960	0.4199	13x	3.7189	0.5203	0.0835	26.7x	8.3369	0.2808	29x
WikiTalk	27.1527	12.1908	23.0929	1.2x	18.8316	7.5769	4.5980	4.1x	61.8645	17.3112	3.5x
IA-online-ads	80.5552	48.9832	26.3587	2.6x	28.4565	13.3298	7.1345	3.9x	183.3022	2.1798	84.1x
StackOverflow	503.9162	57.2630	59.5124	8x	97.3559	26.8346	4.9405	19.7x	215.6287	49.9042	4.3x
Rec-MovieLens	1238.8557	149.399	136.2383	9.1x	83.6547	24.9987	20.6102	4x	484.9321	63.6283	7.6x
Soc-bitcoin	3491.3959	595.5708	802.7460	4.3x	698.2412	86.4989	98.8155	7.1x	1795.2359	198.9950	9x
RedditComments	7968.3687	1133.7942	1019.3465	7.8x	1605.1336	123.0630	158.9597	10.1x	2943.4058	360.0926	8.2x

and 128GB RAM. We download the codes of baselines published by the authors and followed the compilation and usage instructions. All algorithms are implemented in C++11 compiled by GCC v8.3.0 with -O3 optimizations. To run in parallel, all baselines use the same OpenMP mode to our method for parallel speedup.

For 2SCENT, we use the algorithm with bloom filter and bundle to find the temporal cycles. For BTS, we use the same parameter sampling probability q_j provided in [16]. For EWS, we set the edge sampling $p = 0.01$ and wedge sampling $q = 1$.

D. Accuracy Evaluation

First, we verify the accuracy of the proposed FAST on four datasets with different degree distributions and different data scales compared with the exact EX algorithm. We report the counts of motif instances of all 2-node and 3-node, 3-edge temporal motifs with $\delta = 600s$ detected by our method and EX algorithm in Fig. 10.

As we can see, the color scale in blue figures and red figures are identical, which means our FAST can find out the same number of motif instances of all kinds of temporal motifs as exact EX algorithm on four tested datasets. For example, both FAST and EX count 302K motif instances of all star temporal motifs on the small dataset CollegeMsg, and both detect 1217K motif instances of all triangle temporal motifs on the large dataset StackOverflow in all. In addition, our FAST can detect accurate number of instances for specific type of temporal motif, *e.g.*, pair temporal motifs. Our FAST detect 31.6K instances and 10.6K instances of pair motif M_{55} on CollegeMsg and Superuser datasets, which are the same as those of EX. Furthermore, our FAST can detect accurate results on extremely unbalanced dataset (*i.e.*, WikiTalk). For example, FAST find out exact 65.7M and 64.9M instances for star temporal motifs M_{13} and M_{53} on WikiTalk compared to EX algorithm. Lastly, this experiment also verifies that our designed triple and quadruple counters for both star/pair and triangle motifs all can accurately identify the kinds of different temporal motifs.

E. Efficiency Evaluation

We next study the efficiency of our FAST in counting temporal motifs on all datasets compared with both exact and sampling baselines in a single-threaded environment. We report the experimental results of all algorithms in Table III. Notice that EX, BT-Pair, and 2SCENT-Tri are exact algorithms, and EWS and BTS-Pair are sampling algorithms. Since our FAST is an exact algorithm, the speedup is calculated by the corresponding exact algorithm. 2SCENT can only detect the triangle motif M_{26}, while FAST-Tri can find out all kinds of triangle temporal motifs.

As shown in Table III, our FAST (including FAST-Pair and FAST-Tri) is significantly faster than all exact baselines on all datasets in the single-threaded environment. In particular, FAST achieves average $7\times$ speedup against EX algorithm across all datasets, reaching up to $25\times$ speedup on MathOverflow. This is because FAST can directly identify the kinds of temporal motifs according to edge information and relationship between edges, and uses two quadruple counters (*i.e.*, $Star[\cdot,\cdot,\cdot,\cdot]$ and $Tri[\cdot,\cdot,\cdot,\cdot]$) and one triple counter (*i.e.*, $Pair[\cdot,\cdot,\cdot]$) to record the number of instances for all kinds of temporal motifs simultaneously. Besides, our FAST also employs the most efficient implementation strategies to achieves the least edge traversal and counter updating operation, which significantly reduces the computation cost. Although EX algorithm also achieves linear time complexity, it maintains more than ten triple and tuple counters and

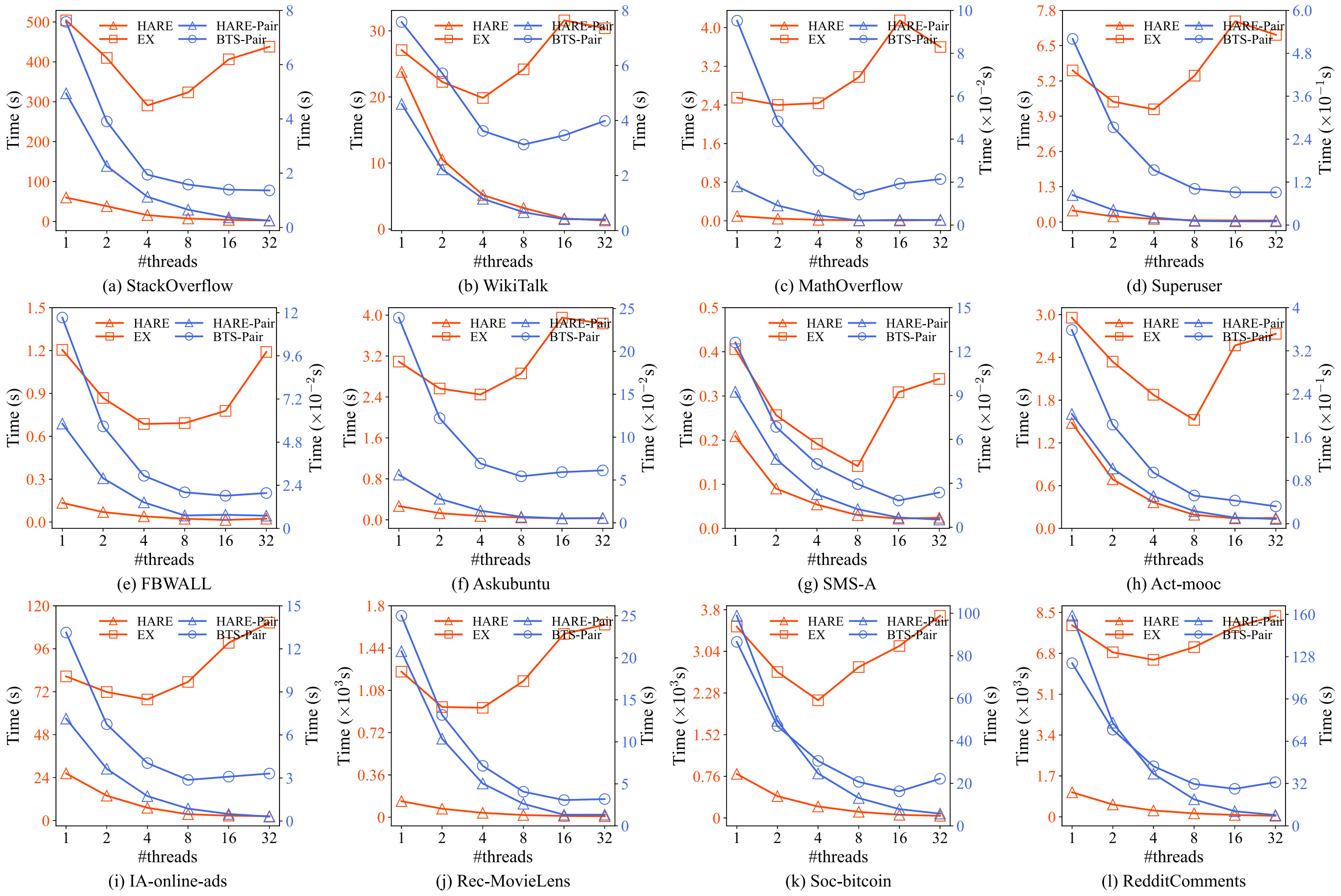


Fig. 11: Running time in seconds of parallel algorithms *w.r.t.* $\#threads$.

requires multiple complex update operations for each temporal edge. Our FAST-Pair and FAST-Tri also perform much better than their corresponding exact algorithms (*i.e.*, BT-Pair and 2SCENT-Tri). Specifically, FAST-Pair and FAST-Tri improve average 18.2× and 65.7× efficiency against their counterparts on all datasets, even if 2SCENT-Tri only count one kind of triangle temporal motif. FAST-Pair achieves more than 100 speedup over 2SCENT-Tri on four datasets (*i.e.*, Bitcoinotc, Bitcoinalpha, SMS-A and FBWALL). The main reason is that 2SCENT-Tri detects the triangle motif M_{26} by enumerating all circles, and thus the time complexity is very large. Our FAST-Tri detects all triangle temporal motifs in the edge sequence of each node recursively, which can achieve a running time linear in the number of temporal edges in the graph.

Moreover, our FAST achieves the comparable performance to state-of-the-art sampling algorithms (*i.e.*, EWS and BTS). On ten of fourteen datasets (*e.g.*, SMS-A, Facebook-wall, MathOverflow, Askubuntu, Superuser and Rec-MovieLens), our FAST even significantly exceeds state-of-the-art sampling algorithm EWS. For pair motif counting, our FAST-Pair also significantly outperforms the sampling algorithm BTS-Pair on almost all datasets, reaching 6.2× speedup on MathOverflow dataset. The possible reason is that our propose algorithm not only designs the ingenious counters, but also achieves the least edge traversal without any redundancy in single threading, so as to minimize computation cost. Due to the large time complexity of the basic algorithms adopted by the sampling methods, the time consumption of the sampling methods is still relatively large. For example, on large-scale StackOverflow and Rec-MovieLens datasets, the sampling methods no longer has any time efficiency advantage in comparison to our FAST.

F. Scalable Evaluation

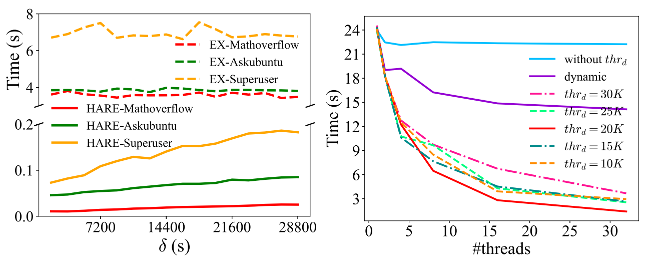
Next, we evaluate the scalability of our proposed hierarchical parallel framework HARE compared with the parallel baselines. We report the experimental results on twelve datasets in Fig. 11. Notice that HARE and EX algorithm refer to left ordinate axis, and HARE-Pair and BTS-Pair refer to right ordinate axis. For our hierarchical parallel framework, we set thr_d to the minimum value of degrees of top 20 nodes in each dataset.

As shown in Fig. 11, our HARE consistently outperforms EX algorithm in all tested cases on all tested datasets. As the number of threads increases, the counting time of HARE decreases, almost reaching a linear speedup. In particular, HARE achieves 26.3× and 24× speedup on large-scale StackOverflow and RedditComments datasets as the number of threads increases from 1 to 32. But the time consumed by Ex first decreases and then increases as the number of threads increases. After more than 16 threads, the running time of EX even exceeds the time of single thread on most datasets. This is because our designed FAST realizes the divide and conquer of the nodes in the network, and solves the problem of data dependence, which is very suitable for parallel computing. At the same time, the proposed hierarchical parallel framework HARE achieves simultaneous inter-node and intra-node parallelism, which effectively solves the problem of unbalanced load caused by nodes with a large degree consuming most of the time. Although EX algorithm can be implemented in parallel by dividing different time periods, its algorithm design involves data dependence, such as the counting processing for each edge depends on the situations of the previous edges. According to this design, each thread cannot be completely independent. When more threads are used, the time consumption caused by data dependency would offset the improvement brought by multi-threading. Specifically, our HARE achieves 538×, 203×, 148×, and 137× speedup against EX algorithm when $\#threads = 32$ on MathOverflow, RedditComments, Rec-MovieLens, and SuperUser datasets, respectively.

Furthermore, our HARE-Pair also significantly performs better than the sampling method BTS-Pair for counting pair temporal motifs in multi-threaded environment. As the number of threads increases, the time utilized by both HARE-Pair and BTS-Pair decreases. However, the time consumed by BTS-Pair begins to increase on some datasets (*e.g.*, FBWALL and WikiTalk), while our HARE-Pair is decreasing. In particular, HARE-Pair is faster than BTS-Pair by 10.6 times on large-scale IA-online-ads dataset when $\#threads = 32$. The main reason may be that BT algorithm employed by BTS has large time complexity (*i.e.*, $O(|\mathcal{E}|(d^{\delta})^{|\mathcal{E}|-1})$), while our algorithm for star/pair motifs is linear in the number of temporal edges in the graph, *i.e.*, $O(2d^{\delta}|\mathcal{E}|)$).

In summary, this experiment effectively verifies the scalability of our proposed hierarchical framework on multi-threaded parallelism. Meanwhile, it also illustrates the huge advantages and potential capabilities of our framework on large-scale network datasets.

G. *Parameter Sensitivity*



(a) Time *w.r.t.* δ (b) Time *w.r.t.* thr_d

Fig. 12: Parameter sensitivity *w.r.t.* δ and thr_d

We now investigate the sensitivity of our proposed framework HARE *w.r.t.* two important parameters, *i.e.*, time constraint δ and the degree threshold thr_d. We report the running time of HARE and EX with different δ values on SuperUser, Askubuntu, and MathOverflow datasets when $\#threads = 32$ in Fig. 12(a), and report the running time of HARE *w.r.t.* different thr_d varying $\#threads$ from 1 to 32 on WikiTalk dataset in Fig. 12(b), where 'dynamic' denotes that dynamic scheduling model of OpenMP is used, and 'without thr_d' means that default (*i.e.*, static) mode is used.

From the results in Fig. 12(a), we can see that the performance of EX is almost unaffected by δ, and our HARE increases slightly with the increase of δ. This is consistent with the above time complexity analysis. However, our HARE is still 37-138 times faster than EX algorithm on three datasets when $\delta = 28.8K$ seconds. Fig. 12(b) illustrates the performance of our parallel framework *w.r.t.* the values of degree threshold thr_d. As you can see, our HARE that uses the hierarchical parallel strategy is significantly faster than the parallel variations without it. Specifically, HARE reaches the best performance when $thr_d = 20K$ among all tested cases. When $thr_d = 20K$ and $thr_d = 25K$, our algorithm performance has some decline, but still performs good. That is, our algorithm can obtain an expected speedup when thr_d falls in a large range. Additionally, we also verify the effectiveness of dynamic scheduling model of OpenMP on WikiTalk dataset. As shown in Fig. 12(b), dynamic scheduling indeed significantly outperforms the version without it (*i.e.*, dynamic vs. without thr_d).

VI. Conclusion

In this paper, we propose a scalable solution for temporal motif counting in large-scale temporal networks. Specifically, we design two fast exact algorithms FAST-Star and FAST-Tri for counting motif instances for star/pair temporal motifs and triangle temporal motifs, respectively. Our customized FAST-Star and FAST-Tri both achieve the time complexity linear in the number of temporal edges of input graph. Based on the natural parallelism of our designed two fast algorithms, we eventually propose a hierarchical parallel framework HARE that fully leverages the multi-threading capacity of modern CPU to realize the most efficient temporal motif counting. We perform extensive experiments on sixteen real-world temporal graphs to demonstrate the superiority and scalability of our proposed method compared with other baselines. Our proposed framework HARE achieves up to 538 times faster than state-of-the-art techniques. Although the proposed algorithms are specially optimized for 2- and 3-node, 3-edge temporal motifs, it will be able to efficiently count the higher-order (more nodes) temporal motifs by expanding the number of center nodes and slightly adapting the structure of the counters, which will be our future work.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China under grant Nos. 62176243, 61773331 and 41927805, the National Program on Key Research Project under grant No. 2019YFC1509100, and the National Key Research and Development Program of China under grant No. 2018AAA0100602.

References

[1] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the tenth ACM international conference on web search and data mining*, 2017, pp. 601–610.

[2] P. Liu, V. Guarrasi, and A. E. Sariyuce, "Temporal network motifs: Models, limitations, evaluation," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[3] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[4] N. Ahmed, R. A. Rossi, J. Lee, T. Willke, R. Zhou, X. Kong, and H. Eldardiry, "Role-based graph embeddings," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[5] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, no. 1, pp. 353–392, 2020.

[6] C. Yang, M. Liu, V. W. Zheng, and J. Han, "Node, motif and subgraph: Leveraging network functional blocks through structural convolution," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 47–52.

[7] Y. Hu, J. Trousdale, K. Josić, and E. Shea-Brown, "Motif statistics and spike correlations in neuronal networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2013, no. 03, p. P03012, 2013.

[8] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, pp. e177–e183, 2007.

[9] S. Gurukar, S. Ranu, and B. Ravindran, "Commit: A scalable approach to mining communication motifs from dynamic networks," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 475–489.

[10] Z. Liu, C. Huang, Y. Yu, and J. Dong, "Motif-preserving dynamic attributed network embedding," in *Proceedings of the Web Conference 2021*, 2021, pp. 1629–1638.

[11] H. Huang, Z. Fang, X. Wang, Y. Miao, and H. Jin, "Motif-preserving temporal network embedding." in *IJCAI*, 2020, pp. 1237–1243.

[12] Y. Jin, G. Song, and C. Shi, "Gralsp: Graph neural networks with local structural patterns," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4361–4368.

[13] Y. Yu, Z. Lu, J. Liu, G. Zhao, and J.-r. Wen, "Rum: Network representation learning using motifs," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1382–1393.

[14] R. Kumar and T. Calders, "2scent: an efficient algorithm for enumerating all simple temporal cycles," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1441–1453, 2018.

[15] P. Mackey, K. Porterfield, E. Fitzhenry, S. Choudhury, and G. Chin, "A chronological edge-driven approach to temporal subgraph isomorphism," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3972–3979.

[16] P. Liu, A. R. Benson, and M. Charikar, "Sampling methods for counting temporal motifs," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 294–302.

[17] J. Wang, Y. Wang, W. Jiang, Y. Li, and K.-L. Tan, "Efficient sampling algorithms for approximate temporal motif counting," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1505–1514.

[18] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[19] A. Vazquez, R. Dobrin, D. Sergi, J.-P. Eckmann, Z. N. Oltvai, and A.-L. Barabási, "The topological relationship between the large-scale attributes and local interaction patterns of complex networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. 52, pp. 17 940–17 945, 2004.

[20] Ö. N. Yaveroğlu, N. Malod-Dognin, D. Davis, Z. Levnajic, V. Janjic, R. Karapandza, A. Stojmirovic, and N. Pržulj, "Revealing the hidden language of complex networks," *Scientific reports*, vol. 4, no. 1, pp. 1–9, 2014.

[21] A. R. Benson, D. F. Gleich, and J. Leskovec, "Tensor spectral clustering for partitioning higher-order network structures," in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015, pp. 118–126.

[22] ——, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[23] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical Computer ence*, vol. 407, no. 1-3, pp. 458–473, 2008.

[24] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient graphlet counting for large networks," in *2015 IEEE International Conference on Data Mining (ICDM)*, 2015.

[25] Y. Santoso, V. Srinivasan, and A. Thomo, "Efficient enumeration of four node graphlets at trillion-scale." in *EDBT*, 2020, pp. 439–442.

[26] P. Holme and J. Saramäki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97–125, 2012, temporal Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157312000841

[27] 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 the 19th ACM international conference on Information and knowledge management*, 2010, pp. 1645–1648.

[28] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 11, p. P11005, 2011.

[29] X. Sun, Y. Tan, Q. Wu, B. Chen, and C. Shen, "Tm-miner: Tfs-based algorithm for mining temporal motifs in large temporal network," *IEEE Access*, vol. 7, pp. 49 778–49 789, 2019.

[30] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 837–846.

[31] N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella, "Graph sample and hold: A framework for big-graph analytics," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1446–1455.

[32] A. Turk and D. Turkoglu, "Revisiting wedge sampling for triangle counting," in *The World Wide Web Conference*, 2019, pp. 1875–1885.

[33] A. Pavan, S. Tirthapura *et al.*, "Counting and sampling triangles from a graph stream," 2013.

[34] S. K. Bera and C. Seshadhri, "How to count triangles, without seeing the whole graph," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 306–316. [Online]. Available: https://doi.org/10.1145/3394486.3403073

[35] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura, "Butterfly counting in bipartite networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2150–2159.

[36] 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*, vol. 30, no. 1, pp. 73–86, 2017.

[37] S. Jain and C. Seshadhri, "A fast and provable method for estimating clique counts using turán's theorem," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 441–449.

[38] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[39] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com