MoTTo: Scalable Motif Counting with Time-aware Topology Constraint for Large-scale Temporal Graphs

Jiantao Li*
Ocean University of China
Qingdao, Shandong, China
ljt7826@stu.ouc.edu.cn

Jianpeng Qi*
Ocean University of China
Qingdao, Shandong, China
qijianpeng@ouc.edu.cn

Yueling Huang
Ocean University of China
Qingdao, China
yueling.huang.22@ucl.ac.uk

Lei Cao
University of Arizona
Tucson, United States
caolei@arizona.edu

Yanwei Yu†
Ocean University of China
Qingdao, China
yuyanwei@ouc.edu.cn

Junyu Dong
Ocean University of China
Qingdao, China
dongjunyu@ouc.edu.cn

Abstract

Temporal motifs are recurring subgraph patterns in temporal graphs, and present in various domains such as social networks, fraud detection, and biological networks. Despite their significance, counting temporal motifs efficiently remains a challenge, particularly on moderately sized datasets with millions of motif instances. To address this challenge, we propose a novel algorithm called Scalable Motif Counting with Time-aware Topology Constraint (MoTTo). MoTTo focuses on accurately counting temporal motifs with up to three nodes and three edges. It first utilizes a topology constraint-based pruning strategy to eliminate nodes that cannot participate in forming temporal motifs before the counting process. Then, it adopts a time-aware topology constraint-based pruning strategy to split large-scale datasets into independent partitions and filter out the unrelated ones, ensuring that the counting results remain unaffected. By investigating the second pruning strategy, we also find that MoTTo can be implemented in a multi-thread manner, further accelerating the counting process significantly. Experimental results on several real-world datasets of varying sizes demonstrate that MoTTo outperforms state-of-the-art methods in terms of efficiency, achieving up to a nine-fold improvement in total temporal motif counting. Specifically, the efficiency of counting triangular temporal motifs is enhanced by up to 31 times compared to state-of-the-art baselines.

CCS Concepts

• **Theory of computation** → **Graph algorithms analysis**; • **Information systems** → **Information systems applications**.

*Both Jiantao Li and Jianpeng Qi contributed equally to this work.
†Yanwei Yu is the corresponding author.

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679694

Keywords

Motif Counting, Large-scale Temporal Graphs, Time-aware Topology Constraint, Pruning

ACM Reference Format:
Jiantao Li, Jianpeng Qi, Yueling Huang, Lei Cao, Yanwei Yu, and Junyu Dong. 2024. MoTTo: Scalable Motif Counting with Time-aware Topology Constraint for Large-scale Temporal Graphs. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679694

1 Introduction

Temporal graphs, such as social networks [5, 32], transportation networks, citation networks [6], biological networks [1], and knowledge graphs [9], represent objects and their relationships through nodes and edges. These networks evolve over time, with new connections forming and old ones dissolving. Temporal motifs are small, recurring sub-graph patterns within these dynamic graphs [13, 21, 23]. They play a crucial role in characterizing user behavior in social and communication networks [20], detecting fraud [3], and representing structure and function of biological networks [22].

Despite their importance, identifying and counting temporal motifs in large-scale graphs poses significant challenges due to scalability issues. As the size of the network increases, the computational complexity of motif detection and counting grows exponentially [4, 7]. This is because the process requires examining numerous sub-graphs and their temporal properties, making it a time-consuming task. For example, in a network with millions of nodes and edges, the number of potential motifs to be examined can reach billions, making traditional counting methods infeasible.

Several approaches [26] have been proposed to address these challenges, including implicit counting methods, parallel counting strategies, and algorithmic optimizations. However, these methods often struggle to handle the large search spaces and the complexities introduced by temporal and topological information. Therefore, the problem of temporal motif counting remains an area of active research, requiring continuous optimization to achieve better performance and scalability.

In this paper, we focus on classical temporal motifs with up to three nodes and three edges and propose a scalable and exact temporal motif instance counting algorithm, called MoTTo. MoTTo employs a topology constraint-based pruning strategy, utilizing several

mathematical matrix tools to remove unrelated nodes that cannot form motifs. It then applies a time-aware topology constraint-based pruning strategy to split the large-scale dataset into multiple independent time-spans (or partitions). By doing so, numerous unrelated partitions about the target nodes (motifs) can be filtered out. Meanwhile, the second strategy can also enable MoTTo to function as a parallel method, where partitions can be handled by different computing threads. By utilizing these two pruning strategies and parallelization, the search space can be reduced, and the speed of identifying and counting motifs can be significantly improved. We conduct extensive experiments on multiple real-world datasets to demonstrate the superiority of our proposed framework compared to state-of-the-art (SOTA) methods. Our results show that MoTTo achieves substantial speedup, making it a promising solution for temporal motif counting in large-scale networks.

This work makes the following contributions:

- We propose MoTTo, an exact counting method for temporal motifs with up to three nodes and three edges, incorporating topology constraint-based and time-aware topology constraint-based pruning strategies. This highly efficient and accurate algorithm is designed for identifying and counting temporal motifs in large-scale datasets.
- We formalize the pruning strategies for both topology and time-aware topology constraints mathematically and accurately, specifically for three-node and three-edge motifs. The topology constraint-based strategy filters out numerous unrelated nodes given a target node, while the time-aware topology constraint-based strategy removes numerous independent data partitions. The methodology involved is also applicable to other types of motifs.
- Our time-aware topology constraint-based pruning strategy also enables multi-thread parallelization, accelerating the counting process while maintaining accuracy. Each thread can analyze several different data partitions simultaneously.
- We conduct extensive experiments on several real-world datasets to demonstrate the effectiveness of our proposed algorithm. The results show that MoTTo achieves up to a 31.2× speedup compared to SOTA baselines in counting times.

2 Related Work

Recent years have seen a surge in research focusing on the enumeration of temporal motifs, reflecting a growing interest in understanding the temporal dynamics of complex systems.

Kovanen et al. [13] laid the groundwork by introducing the *concept of temporal motifs*. Building upon this foundation, Paranjape et al. [23] defined our target motifs as δ-temporal motifs, characterized by up to three nodes and three edges. They proposed the EX algorithm, which employs subgraph enumeration techniques to efficiently count these motifs. Subsequent work has explored various approaches to *enhance the efficiency and accuracy* of temporal motif counting. Mackey et al. [19] introduced a backtracking strategy, improving search efficiency by imposing temporal order constraints on subgraphs. Kumar et al. [14] extended classical graph cycle detection algorithms to the temporal domain with their 2scent method. Pashanasangi et al. [24] leveraged degeneracy ordering and combinatorial arguments to design an efficient algorithm for counting temporal triangles. Gao et al. [7] proposed three counters tailored to different types of temporal motifs, enabling more efficient retrieval and enumeration. Yuan et al. [33] introduced a GPU-accelerated counting method, capitalizing on the parallel computing power of GPUs to optimize workload execution.

Additionally, several studies have focused on *approximating* temporal motif counts. Notable contributions include works by Liu et al. [18], Wang et al. [30], and Sarpe et al. [28]. Other methods have explored temporal motifs with more nodes and edges, as seen in studies by Seshadhri et al. [29], Bressan et al. [2], Iyer et al. [10], Jain et al. [11], Jha et al. [12], Pinar et al. [25], and Wang et al. [31]. Lee et al. [15, 16] studied motifs in hypergraphs.

Despite these advancements, existing research has not fully addressed the complexity of counting problems on large-scale temporal graphs, nor has it proposed an accurate and highly efficient data pruning method.

3 Preliminary

In this section, we first introduce the definitions of temporal graphs and δ-temporal motifs based on [7, 23]. Then, we provide the problem definition. Finally, we list two basic definitions used in our study, which are the fundamentals used to accelerate the counting processes.

Definition 1 (Temporal Graph [7]). *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.*

Definition 2 (δ-temporal Motif [23]). *A k-node, l-edge, δ-temporal motif is a sequence of l temporal edges in chronological order within a time constraint δ, $M = \langle (u_1, v_1, t_1), (u_2, v_2, t_2), \ldots, (u_l, v_l, t_l) \rangle$ $(u_i, v_i \in \mathcal{V})$, i.e., $t_1 \leq t_2 \leq \cdots \leq t_l$ and $t_l - t_1 \leq \delta$, such that the induced static graph from these edges is connected and include k nodes.*

Problem. *Given a temporal graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}\}$ and time constraint δ, our goal is to count all instances of 3-edge temporal motifs with up to three nodes, i.e., k-node, l-edge, δ-temporal motifs where $k \leq 3$ and $l = 3$.*

According to Def. 2, all qualifying motifs with two nodes are shown in Figure 1 and are denoted as $Pair_1 \sim Pair_4$. All qualifying motifs with three nodes are shown in Figure 2 and Figure 3. The motifs that form a triangle are denoted as $Tri_1 \sim Tri_8$, and the motifs where one of the three nodes has a degree of three, acting as the core of a star, are denoted as $Star_1 \sim Star_{24}$.

Figure 1: All Pair temporal motifs with 2 nodes and 3 edges

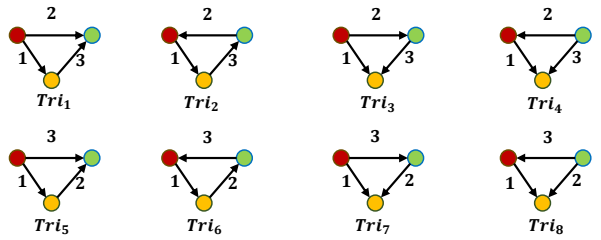


Figure 2: All Triangle temporal motifs with 3 nodes and 3 edges

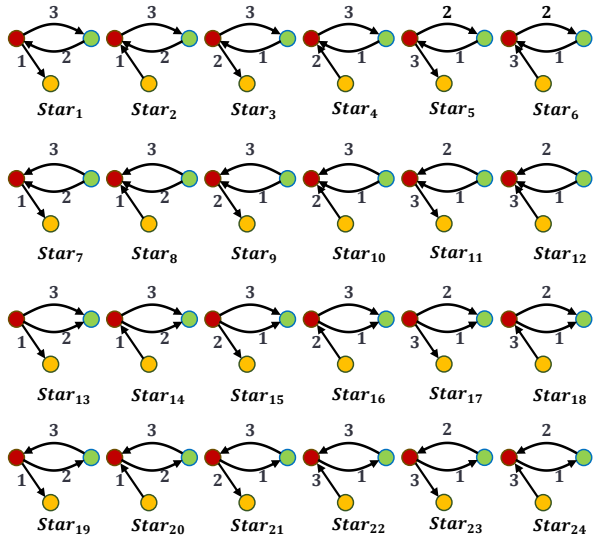
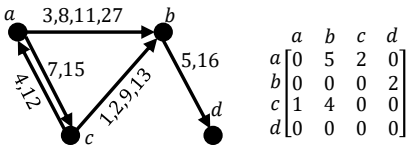


Figure 3: All Star temporal motifs with 3 nodes and 3 edges

A Toy Example. Figure 4 illustrates a temporal graph with 4 nodes and 14 temporal edges. Given $\delta = 8$, the sequence $\langle(a, b, 3), (a, b, 8), (a, b, 11)\rangle$ is a 2-node, 3-edge δ-temporal motif instance of $Pair_1$. The sequence $\langle(c, a, 4), (a, c, 7), (a, b, 11)\rangle$ is a 3-node, 3-edge δ-temporal motif instance of $Star_5$. Conversely, the sequence $\langle(c, b, 1), (c, a, 4), (a, b, 11)\rangle$ is not a 3-node, 3-edge δ-temporal motif instance because the timestamp interval between the first edge and third edge is 10, which exceeds δ.



$$\begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 5 & 2 & 0 \\ b & 0 & 0 & 0 & 2 \\ c & 1 & 4 & 0 & 0 \\ d & 0 & 0 & 0 & 0 \end{array}$$

Figure 4: An example of a temporal graph with 4 nodes and 14 temporal edges and its adjacency matrix

Gao et al. [7] introduce a novel motif counting method, the FAST method, as delineated in Definition 3. The efficiency of this method is highlighted through experimental results. Motivated by these insights, we explore using the topological structure to assess triangle formation potential before node-based motif counting, streamlining calculations to enhance efficiency, which underpins our MoTTo algorithm's design.

Definition 3 (Node-Based Counting Method). *The node-based counting method, denoted as* FAST$(\mathcal{V}, \mathcal{E}, \mathcal{T})$*, systematically traverses all unvisited nodes within set* $\mathcal{V}$*. During the counting process, it identifies relevant instances of temporal motifs for each individual node.*

Finally, we introduce some some properties of graphs that will be used in the following sections.

In graph theory, the adjacency matrix $\mathbf{A}$ of a directed graph is an $n \times n$ matrix, where n is the number of nodes. Each entry a_{ij} in the matrix represents the number of directed edges from node i to node j. The powers of the adjacency matrix are significant in graph theory and have well-defined interpretations:

(1) The second power: $\mathbf{A}^2$, where the entry $a_{ij}^{(2)}$ denotes the number of distinct paths of length 2 from node i to node j.
(2) Higher powers: the entry $a_{ij}^{(k)}$ of $\mathbf{A}^k$ represents the number of distinct paths of length k from node i to node j.

4 Methodology

In this section, we provide a detailed introduction to our precise counting method, MoTTo, for the 36 types of temporal motifs mentioned in Section 3. The algorithm first performs a topological assessment before counting and then employs a time-constraint partitioning approach to optimize the counting process. Furthermore, MoTTo is designed with an efficient parallel manner.

4.1 Topology Constraint-Based Pruning

Before delving into the counting of temporal motifs, this section introduces two essential lemmas that facilitate the pruning of nodes unable to generate static motifs based on a topological constraint matrix: one targets triangle temporal motifs (Lemma 1), and the other addresses star and pair temporal motifs (Lemma 2). Subsequently, we delineate a method to exclude nodes that are incapable of participating in the formation of temporal motifs.

For triangle temporal motifs, from a topological perspective and disregarding timestamp and directional information, all triangle motifs in the structure can manifest as a simple triangle. In the static graph of $\mathcal{G}$, using the adjacency matrix $\mathbf{A}$ of its undirected, unweighted basic graph, we can employ $\mathbf{M} = (\mathbf{A}+\mathbf{A}^\intercal)^2 \odot (\mathbf{A}+\mathbf{A}^\intercal)$ to depict the existing triangle topologies of $\mathcal{G}$ accurately. Here, $\odot$ signifies the *Hadamard* product, representing the logical *AND*, where the topological patterns of one matrix coincide with those of another.

Lemma 1. *Given the triangle topology matrix* $\mathbf{M}$*, for node* v_i*, if* $\mathbf{M}_{i,j} = 0$ *for any* j*, it signifies that nodes* v_i *and* v_j *cannot form a triangle temporal motif instance.*

Proof. To prove Lemma 1, we start by defining the matrix $\mathbf{M}$ as $\mathbf{M} = (\mathbf{A} + \mathbf{A}^\intercal)^2 \odot (\mathbf{A} + \mathbf{A}^\intercal)$. Here, $\mathbf{A} + \mathbf{A}^\intercal$ ensures the graph is treated as undirected by combining edges in both directions. The expression $(\mathbf{A}+\mathbf{A}^\intercal)^2$ counts the number of two-step paths between any two nodes i and j. A two-step path from node i to node j via node k indicates potential nodes k that can form a triangle with i and j. The Hadamard product $\odot$ with $(\mathbf{A} + \mathbf{A}^\intercal)$ again ensures that

for any non-zero entry in $\mathbf{M}_{i,j}$, there must be a direct link between nodes i and j, forming a closed triangle. Thus, $\mathbf{M}_{i,j} \neq 0$ indicates the presence of a triangle motif involving nodes i and j, facilitated by at least one other node k. If $\mathbf{M}_{i,j} = 0$, it implies that either there are no two-step paths from i to j (i.e., no intermediary node k connects i and j to form a triangle), or there is no direct connection between i and j as required for a triangle. Therefore, the absence of these paths and direct connections precludes the existence of any triangle involving i and j in the graph, regardless of the graph's temporal aspects. This concludes the proof that the zero entries in $\mathbf{M}$ correspond to node pairs v_i and v_j that lack the necessary connectivity to participate in any triangle temporal motif. □

Similarly, *for star and pair temporal patterns*, there exists a topological matrix $\mathbf{S} = \mathbf{A} \odot \mathbf{A}^\intercal$. We can determine the potential formation of topological structures by inspecting certain rows and columns of $\mathbf{S}$ and $\mathbf{A}$ of the temporal graph. To clarify, we provide an example in Figure 5, where the blue node (v_i) represents the target node, i.e., assessing whether this node can generate star or pair temporal motifs. A value greater than or equal to 2 in the i-th row and j-th column of $\mathbf{A}$ indicates at least two edges from node i to node j as shown in the second graph. Similarly, a value greater than or equal to 2 in the j-th row and i-th column of $\mathbf{A}$ represents at least two edges from node j to node i as illustrated in the third graph. In the example, a non-zero value in the i-th row of $\mathbf{S}$ indicates that there is a bidirectional edge between the blue node and another node as depicted in the structure.

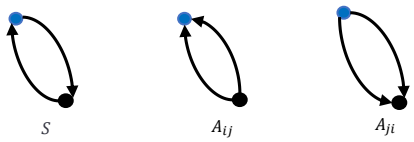


Figure 5: Three pattern examples represented by star and pair temporal motifs.

Further, we can determine whether the blue node i can form these patterns by checking Lemma 2.

Lemma 2. *Given the star and pair topological matrix* $\mathbf{S}$ *and the adjacency matrix* $\mathbf{A}$ *of a temporal graph, for any node* v_i*, if the following conditions are met, node* i *can act as the central point of a star temporal pattern or as a node in a pair temporal pattern:*

(1) Non-zero values exist in the i*-th row of* $\mathbf{S}$ *for node* v_j*, or the value of* $\mathbf{A}_{ij}$ *in the* i*-th row and* j*-th column of matrix* A *is greater than 2, or the value of* $\mathbf{A}_{ji}$ *in the* j*-th row and* i*-th column is greater than 2;*

(2) The degree of the node is at least 3.

Proof. First, we understand the definition of the matrix $\mathbf{S}$ as $\mathbf{S} = \mathbf{A} \odot \mathbf{A}^\intercal$. A non-zero value in the i-th row of $\mathbf{S}$ indicates that node i has bidirectional edges with node j. This is critical for star formations where node i could potentially act as the central node, connecting bidirectionally with multiple nodes. For pair temporal patterns, the adjacency matrix $\mathbf{A}$ plays a pivotal role: The condition $\mathbf{A}_{ij} > 2$ suggests there are multiple directed edges from i to j, which can indicate a pair motif where multiple interactions are directed from i to j. Similarly, $\mathbf{A}_{ji} > 2$ indicates multiple interactions from j to i, reinforcing the possibility of a pair motif but in the opposite direction. The second condition ensures that node i has a degree of at least 3, which is a minimum requirement for a node to act as a central point in a star topology or to have significant interactions in pair motifs. This implies a sufficient level of connectivity for node i to participate actively in the specified motifs. Given these conditions, we conclude that node i meets the structural prerequisites to form or participate in star and pair temporal motifs as defined. This fulfills the requirements set forth in Lemma 2. □

At this point, we can clearly understand the meaning of the values in the topological constraint matrix $\mathbf{M}$ and $\mathbf{S}$. To align with the subsequent counting algorithm, if Lemma 1 and/or Lemma 2 are satisfied, we can prune the nodes that cannot participate in forming motifs. Formally, based on these two lemmas, we have Corollary 1.

Corollary 1. *Given a temporal graph* $\mathcal{G}$ *and its adjacency matrix* $\mathbf{A}$*, for node* v_i*, for any* j*, if* $\mathbf{M}_{i,j} = 0$*, then* v_j *should be pruned in the process of counting triangle temporal motifs w.r.t. node* v_i*; If* $\mathbf{S}_{i,j} = 0$ *or the degree of node* v_i *is less than 3, then* v_i *should be pruned in the process of counting star temporal motifs and pair temporal motifs.*

Given the adjacency matrix $\mathbf{A}$ of the temporal graph $\mathcal{G}$, Algorithm 1 shows the topology constraint-based pruning pseudo-code. Lines 1-2 first obtain $\mathbf{M}$ and $\mathbf{S}$.

Lines 3-8 follow Corollary 1 to identify unrelated nodes that need to be pruned. According to the previous deduction process, if all elements in the i-th row of the matrix are 0, then node i definitely cannot generate a triangle (Line 4). Additionally, in lines 3~~8~~, if $Flag1 \wedge Flag2$ equals True, the node should be pruned. To efficiently mark them, we use two hash maps Map_M and Map_S to record if they should be pruned (True) or not (False). Finally, line 10 returns two mappings indicating whether nodes are pruned, along with $\mathbf{M}$, which can be used to determine whether u_i and u_j can form a triangle temporal motif instances based on the value of $\mathbf{M}_{i,j}$.

Algorithm 1 getTCM

Input: A temporal graph $\mathcal{G}$, and its adjacency matrix $\mathbf{A}$
Output: HashMap $Map_M, Map_S[v \rightarrow boolean]$, $\mathbf{M}$;

1: $\mathbf{M} = (\mathbf{A} + \mathbf{A}^\intercal)^2 \odot (\mathbf{A} + \mathbf{A}^\intercal)$
2: $\mathbf{S} = \mathbf{A} \odot \mathbf{A}^\intercal$
3: **for** $v_i \in \mathcal{V}$ **do**
4: $\quad Map_{\mathbf{M}}[v_i] = ((\sum_{j=0}^{|\mathcal{V}|-1} \mathbf{M}_{i,j}) == 0)$; //Lemma 1
5: $\quad Flag1 = ((\sum_{k=0}^{|\mathcal{V}|-1} \mathbf{S}_{i,k}) == 0) \vee (\mathbf{A}_{ij} > 2) \vee (\mathbf{A}_{ji} > 2)$;
6: $\quad Flag2 = (degree(v_i) < 3)$;
7: $\quad Map_{\mathbf{S}}[v_i] = Flag1 \wedge Flag2$; //Lemma 2
8: **end for**
9: **return** $Map_{\mathbf{M}}, Map_{\mathbf{S}}, \mathbf{M}$;

4.2 Time-aware Topology Constraint-Based Pruning

The strategy described in Section 4.1 involves pruning over all time spans of the temporal graph. However, due to the δ constraint, over large time spans, it is unlikely that two edges will be part of

the same temporal motif instance. This motivates us to divide the overall time span into smaller, independent time spans and then perform topology constraint-based pruning within these smaller spans separately, thereby achieving better pruning effects. However, obtaining independent spans is hindered by the double-counting issues around the borders of these spans. Therefore, in this section, we first illustrate the partitioning steps and then address the double-counting issue. Finally, we propose our convenient and efficient division strategy. However, obtaining independent spans is hindered by the double-counting issues between the time spans. Therefore, in this section, we first illustrate the partitioning steps and then address the double-counting issue. Finally, we propose our convenient and efficient division strategy.

To avoid affecting the counting results and to facilitate computation, for any temporal graph, we define a parameter ω, which divides the total time span $\mathcal{W} < t_{start}, t_{end} >$ into smaller time spans. Each small time span has a length of ω, and $\mathcal{Q}$ is the set of all small time spans. Assuming it is divided into s parts, $\mathcal{Q} = \{< t_{start}, t_{start}+\omega-1 >, < t_{start}+\omega-\delta, t_{start}+\omega\times 2-\delta-1 >, \ldots, < t_{start}+\omega\times(s-1)-\delta, t_{start}+\omega\times s-\delta-1 >, < t_{start}+\omega\times s-\delta, t_{end} >\}$. Then, if the timestamps of all three edges of a temporal motif instance fall within the overlapping δ region of two time spans, we only count this temporal motif in the left time span, thereby avoiding double counting.

It is worth noting that during partitioning, two adjacent time spans have overlapping time of δ. This is because if we directly divide the time spans into blocks of fixed length ω without overlap, and when counting a δ-temporal motif instance $\langle (v_a, v_b, 8), (v_c, v_b, 9), (v_a, v_b, 11)\rangle$ in Figure 6, the boundary of the partition (time span) happens to fall between $< 8, 11 >$, then it cannot be counted by both the left and right time spans. Therefore, overlapping is necessary. For another temporal motif instance $\langle (v_a, v_c, 7), (v_a, v_b, 8), (v_c, v_b, 9)\rangle$, where its three edges all fall within the δ region $< 7, 9 >$, we count it only in the left time span to avoid double counting.

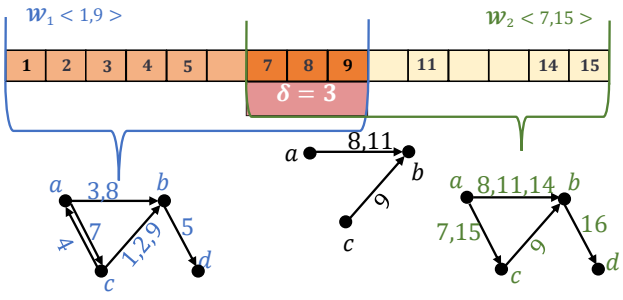


Figure 6: An example of time span partition when $\omega = 9$ and $\delta = 3$.

Figure 6 gives an example of overlapping, which is sourced from the example in Figure 4. We set $\delta = 3$, and the basic partition length $\omega = 9$, resulting in an example of dividing the time span. If we directly divide the time span with each interval length as ω, then in this example, it can be divided into two non-overlapping spans: $< 1, 9 >$ and $< 10, 18 >$. However, this would miss counting temporal motifs that span two time spans, such as the temporal motif $\langle (v_a, v_b, 8), (v_c, v_b, 9), (v_a, v_b, 11)\rangle$. Our designed partition method overlaps each interval with the previous one by a length of δ, dividing the timestamps into $< 1, 9 >$ and $< 7, 15 >$ in the figure, ensuring that no temporal motif instances are missed.

Meanwhile, due to the independence of the different partitions, the algorithm is naturally suitable for parallel strategies. Therefore, we also design a parallel version of MoTTo. In the next section, we straightforwardly present this version.

4.3 The MoTTo Algorithm

Finally, in this section, we demonstrate our proposed MoTTo in Algorithm 2, which is a multi-thread, efficient, and lossless parallel algorithm. In each thread, counting process can promote correctly and independently.

Algorithm 2 MoTTo algorithm

Input: A temporal graph $\mathcal{G}$, $\#threads$, δ, time span length ω
Output: All the counting results $rltMap$

1: Get the set $\mathcal{Q}$ of all time spans $\mathcal{W}$ according to our time span partition method in Sec. 4.2;
2: **OpenMP** for num_threads($\#threads$);
3: **for** $\mathcal{W} \in \mathcal{Q}$ **do**
4: $\mathcal{G}(\mathcal{W}) = \{\{\mathcal{V}_\mathcal{W}, \mathcal{E}_\mathcal{W}, \mathcal{T}_\mathcal{W}\} \subseteq \mathcal{G}$, where $\mathcal{T}_\mathcal{W}$ within $\mathcal{W}\}$;
5: $Map_\mathbf{M}, Map_\mathbf{S}, \mathbf{M} = getTCM(\mathcal{G}(\mathcal{W}))$;
6: $\mathcal{V}_\mathbf{M} = \{ v | v \in \mathcal{V}_\mathcal{W}$ & $Map_\mathbf{M}[v] == True \}$;
7: $rltMap[Tri]$=FAST$(\mathcal{V}_\mathcal{W} - \mathcal{V}_\mathbf{M}, \mathcal{E}_\mathcal{W}, \mathcal{T}_\mathcal{W})$;
8: $\mathcal{V}_\mathbf{S} = \{ v | v \in \mathcal{V}_\mathcal{W}$ & $Map_\mathbf{S}[v] == True \}$;
9: $rltMap[Star], rltMap[Pair]$=FAST$(\mathcal{V}_\mathcal{W} - \mathcal{V}_\mathbf{S}, \mathcal{E}_\mathcal{W}, \mathcal{T}_\mathcal{W})$;
10: **end for**
11: **return** $rltMap$;

Algorithm 2 inputs a temporal graph $\mathcal{G}$, the number of *threads*, δ, and time span length ω. In line 1, we can get the set $\mathcal{Q}$ of all time spans $\mathcal{W}$ according to our time span division method. In line 2, it sets the number of threads using OpenMP. From lines 3 -10, it iterates over all time spans $\mathcal{W}$. In line 4, for each $\mathcal{W}$, it obtains the subgraph $\mathcal{G}(\mathcal{W})$ of the temporal graph $\mathcal{G}$ within the time span $\mathcal{W}$. In line 5, it obtains the mappings $Map_\mathbf{M}$ and $Map_\mathbf{S}$ from Algorithm 1, which indicate whether nodes are pruned, and the matrix $\mathbf{M}$, which reflects whether two nodes form a triangle temporal motif. In line 6, the set of pruned nodes $V_\mathbf{M}$ is derived from $Map_\mathbf{M}$. In line 7, only unpruned nodes are retained for the FAST [7] algorithm to count triangle temporal motifs. Similarly, in lines 8 to 9, unpruned nodes from $Map_\mathbf{S}$ are used in the FAST algorithm to count star and pair temporal motifs. The algorithm finally returns the counting results.

4.4 Time Complexity Analysis

Algorithm 1 has a time complexity of $O((ab+1)|\mathcal{V}| + |\mathcal{E}|)$, where a and b are the number of rows and columns, respectively, of the sparse form of the adjacency matrix A. Here, $|\mathcal{V}|$ and $|\mathcal{E}|$ represent the number of nodes in the node set $\mathcal{V}$ and the number of edges in the temporal graph $\mathcal{G}$, respectively.

Within the main loop of MoTTo, the algorithm iterates over the time spans in $\mathcal{Q}$, and for each span, extracts the corresponding subgraph by iterating over the temporal edges, leading to $O(|\mathcal{E}|)$

Table 1: Basic statistics of 8 temporal networks.

Dataset	# nodes	# temporal edges	Time span
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
SMS-A	44,090	544,817	338
FBWALL	45,813	855,542	1,591
Rec-MovieLens	283,228	27,753,444	1,128

operations per span. The time complexity of this step is influenced by the number of vertices and edges. The initialization steps involving *getTCM* have a time complexity of $O((ab+1)|\mathcal{V}_{\mathcal{W}}| + |\mathcal{E}_{\mathcal{W}}|)$. The time span partitioning and the main loop over time spans both contribute $O(|\mathcal{E}|)$ each. The time complexity of this part is $O((ab+1)|\mathcal{V}_{\mathcal{W}}| + |\mathcal{E}_{\mathcal{W}}| + \bar{d}(|\mathcal{V}_{\mathcal{W}}| - \epsilon))|\mathcal{Q}|)$, where $\bar{d}$ denotes the average degree of the nodes within time constraint δ, ϵ represents the number of nodes pruned due to time-aware topology constraints, and $|\mathcal{V}_{\mathcal{W}}|$ and $|\mathcal{E}_{\mathcal{W}}|$ are the number of nodes and edges in $\mathcal{G}(\mathcal{W})$, respectively. Since the value of ϵ is significant compared to $|\mathcal{V}_{\mathcal{W}}|$, it greatly mitigates the exponential impact of δ on the time complexity.

Thus, we summarize the time complexity of MoTTo as $O(|\mathcal{E}| + ((ab+1)|\mathcal{V}_{\mathcal{W}}| + |\mathcal{E}_{\mathcal{W}}| + \bar{d}(|\mathcal{V}_{\mathcal{W}}| - \epsilon))|\mathcal{Q}|)$.

5 Experiments

5.1 Experiment Setup

Datasets. All datasets were sourced from open-source websites [17, 27], as detailed in Table 1, where the unit for *Time span* is in days. These datasets encompass multiple domains and scales, ranging from large datasets with up to 27 million edges to smaller ones with approximately 20,000 edges.

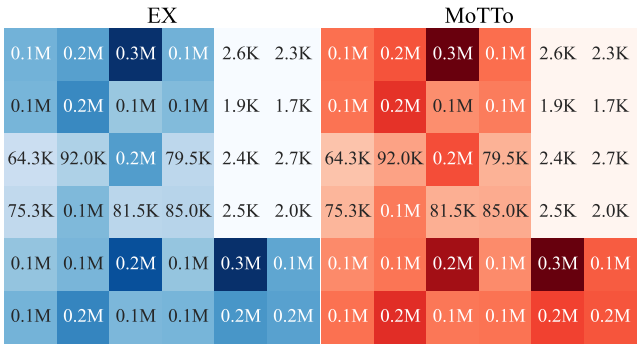
Baselines. We use EX [23] and the SOTA method FAST [7] as baselines in our experiments. To investigate the speed of our algorithm compared to FAST, we compare our approach with the fastest FAST-Tri method in terms of counting triangle temporal motifs.

Metrics. We use accuracy and counting time as our evaluation criteria for the experiments. Accuracy refers to the clear and precise counting of each of the 36 types of temporal motifs. Counting time measures the computational efficiency of the algorithm in terms of the time taken to complete the counting process.

Implementations. We conduct our experiments on a server with a 40-core 2.30GHz Intel Xeon E5-2650 v3 processor and 128 GB of RAM. The experimental programming language was C++17 with GCC v8.3.0. We utilize C++ OpenMP for parallelism and the Eigen V3 [8] library functions to assist with sparse matrix multiplication operations. Unless otherwise specified, our experiments are conducted with δ = 3600*s* and #threads = 32.

5.2 Accuracy Verification

MoTTo's counting results for 36 types of temporal motifs are compared with the EX algorithm, as shown in Figure 7. Blue represents



(a) CollegeMsg

(b) SMS-A

Figure 7: Exact counts of δ-temporal motifs with δ = 3600s.

the counting results of EX, while red represents ours. In the provided 6×6 heatmaps, the first four columns represent the counts of all Star temporal motifs. In the last two columns, the first four rows indicate the counts of Triangle temporal motifs, and the last two rows represent the counts of Pair temporal motifs.

From Figure 7, we can clearly see that our counting results for the 36 types of temporal motifs are consistent with those of the EX. By comparing the counting results on the three provided datasets, we can also observe that our algorithm accurately counts temporal motifs even in datasets with significant variations in motif counts. For example, on the SMS-A dataset, we observe that it contains more of the four types of pair motifs compared to other types of temporal motifs, while the count of triangle temporal motif instances is very low, around 1K. Despite such variations in motif counts, our algorithm is able to accurately count the number of each temporal motif instance on this dataset. Similarly, on larger datasets like SMS-A, our counting results is still exact.

5.3 Pruning Efficiency

In our count experiments across all datasets, we calculate the percentage of nodes pruned by our two pruning strategies topology constraint-based (TCP) method, and time-aware topology constraint-based (TTCP) method relative to the total number of nodes. The experimental results are shown in Figure 8. As observed, even before performing the count, we could predict that a large portion of nodes would not generate temporal motif instances, which significantly

Table 2: Running time in seconds of all algorithms on all datasets. δ = 3600s and #threads = 32 (4 for EX).

Datasets	EX	FAST	MoTTo		FAST-Tri	MoTTo-Tri	
	Times	Times	Times	Speedup	Times	Times	Speedup
Email-Eu	1.1300	0.6986	0.3356	**2.08x**	0.2625	0.0497	**5.28x**
CollegeMsg	0.2600	0.1642	0.0660	**2.49x**	0.0821	0.0294	**2.79x**
Bitcoinotc	0.3700	0.1036	0.0187	**5.53x**	0.0172	0.0006	**31.20x**
Bitcoinalpha	0.2700	0.0377	0.0137	**2.76x**	0.0098	0.0019	**5.11x**
Act-mooc	15.7600	4.3090	0.4343	**9.92x**	0.3970	0.0382	**10.40x**
SMS-A	1.0200	0.8621	0.1685	**5.12x**	0.1603	0.0055	**29.13x**
FBWALL	3.1400	1.1046	0.3942	**2.80x**	0.3845	0.0574	**6.70x**
Rec-Movielens	3417	640.5000	312.7360	**2.05x**	130.5008	50.5187	**2.58x**

improved the counting efficiency and verified the effectiveness of our proposed strategies. Meanwhile, the pruning efficiency of TCP is notably high, achieving up to 86% on the Act-mooc dataset and over 80% on the SMS-A and Rec-Movielens datasets. This effectively demonstrates the validity of our method. For TTCP, we achieve up to 41% pruning efficiency on the Email-Eu dataset, with other datasets also showing significant pruning results, further proving the effectiveness of our approach. The TTCP method amplifies the pruning capabilities of the TCP method, making it an excellent partitioning technique.

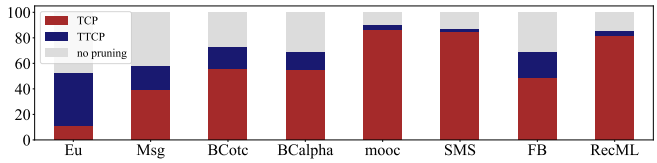


Figure 8: Proportion of nodes pruned by two pruning strategies on all datasets (Dataset names are abbreviated).

5.4 Efficiency Evaluation

We compare our algorithm with EX and FAST, presenting the overall counting time comparison in Table 2. Additionally, we conduct a separate comparison with FAST on speedup specifically for triangle counting time. It is worth noting that both our approach and the baselines only measure the counting time, excluding the preprocessing time. Note that we make sure to set the number of threads for the EX algorithm to 4. We set the number of EX threads to 4 because experiments by Gao et al.[7] have demonstrated that the performance of the EX method is very poor with 32 threads, but performs best with 4 threads. Therefore, we chose the number of threads where the EX method shows the best performance.

From Table 2, we can see that the counting time of MoTTo is not only significantly less than EX but also shows certain advantages over FAST. The most notable advantage is on the Act-mooc dataset, where FAST's counting time is 9.92× ours. On the SMS-A dataset, FAST's counting time is 5.12× ours, and on other datasets, the advantage ranges from 2.05× to 5.53×. Among the datasets, there are some large-scale temporal graphs. For example, Rec-MovieLens is a large-scale temporal graph dataset with 27 million temporal edges. FAST's counting time is 2.05× ours, proving that our algorithm is efficient even on large-scale datasets.

We additionally compare the counting time of triangle temporal motifs with the FAST algorithm. This comparison aims to demonstrate the high pruning efficiency of our algorithm by benchmarking against an excellent triangle temporal motif instance counting algorithm. The experimental results show that, on the given datasets, FAST's counting time for triangle temporal motifs is up to 31.20× ours. Because in datasets like Act-mooc or SMS-A, the number of generated triangle temporal motif instances is very small, and MoTTo has high pruning efficiency for triangles, we performed fewer counting operations compared to other methods on these datasets, resulting in naturally higher counting efficiency.

5.5 Runtime Scalability

To verify the scalability of our method, we conduct comparative experiments on counting time with FAST and EX on different datasets and threads using δ = 600s to get numerous time spans. Our experimental results are shown in Figure 9, where the x-axis represents the number of threads set for the parallel algorithm. We performed multiple experiments with thread counts of 4, 8, 12, 16, 20, 24, 28, and 32, took the average, and obtained three lines representing the counting time. The y-axis unit is milliseconds.

It is evident from the graph that our algorithm's performance improves rapidly with an increase in the number of threads, which is closely related to our efficient parallel model. Our parallel approach effectively partitions large-scale problems into smaller ones, ensuring that the counting results and resources of each part do not interfere with each other. Changing from 4 threads to 8 threads, both our method and FAST show significant improvements in efficiency. It is evident that the efficiency improvement brought by increasing the number of threads is more pronounced in our method. As for the EX algorithm, after 4 threads, the counting time of EX slowly increases as the number of threads increases.

During the process of setting the number of threads from 4 threads to 32 threads, our algorithm exhibits a noticeable decrease in counting time on the Act-mooc, CollegeMsg, Email-Eu, FBWALL, and SMS-A datasets, which demonstrates the scalability of our proposed parallel method based on time-aware topology constraint.

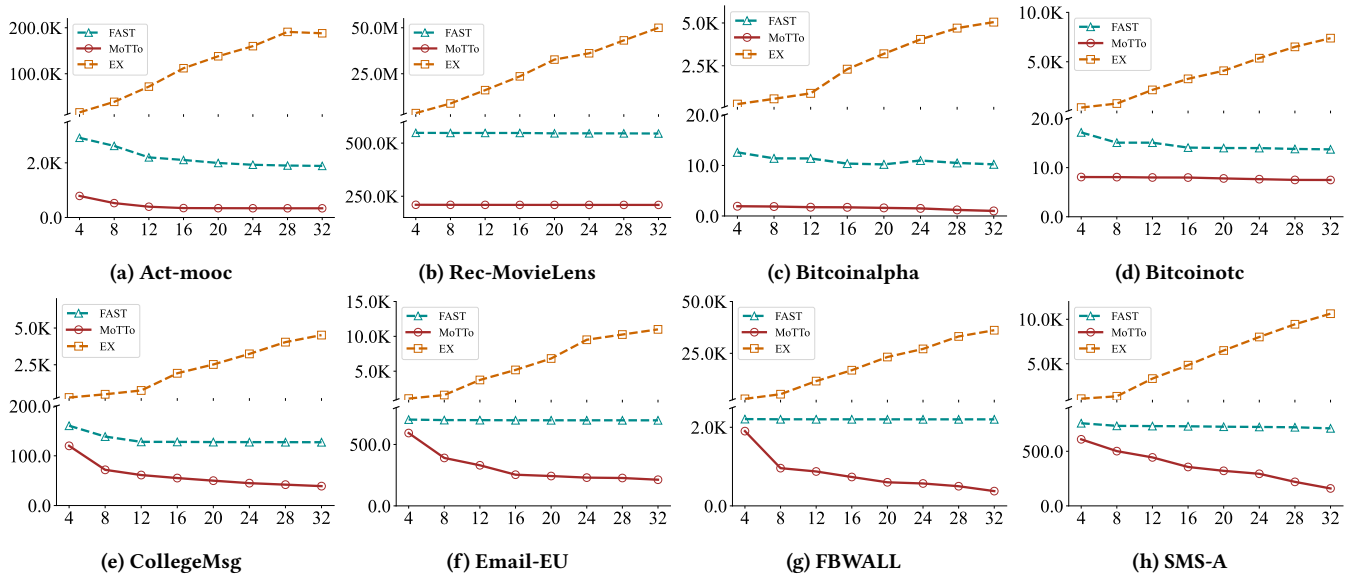


Figure 9: Comparison of counting times in milliseconds for our algorithm with FAST and EX *w.r.t.* #threads

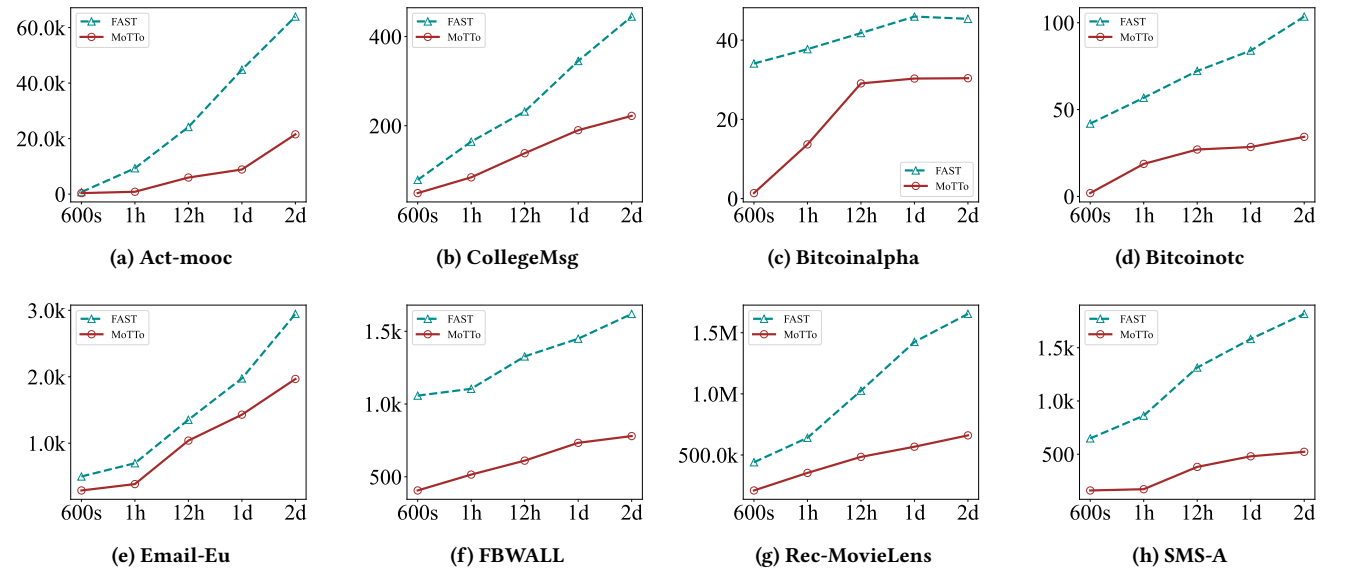


Figure 10: Comparison of counting times in milliseconds for our algorithm with FAST *w.r.t.* δ

On the Bitcoinalpha and Bitcoinotc datasets, our counting time is already quite low at low thread counts, so the decrease in counting time with increasing thread count is not as significant, but still gradually decreased. Our method's advantage over the FAST algorithm lies in the time-aware topology constraint method. On the Rec-MovieLens dataset, regardless of the thread count, the counting time relative to FAST see a significant decrease.

In conclusion, through this experiment, we effectively demonstrate the scalability of our parallel algorithm. The results displayed across different datasets confirm that our method performs well across various scales and validate its effectiveness and potential on large-scale temporal graphs.

5.6 Parameter Sensitivity

At a threads of 32, we compare the counting time of 36 types of temporal motifs with FAST at different values of δ. The unit of the horizontal axis is milliseconds. Additionally, we conducted a comparison of triangle temporal motifs counting time with FAST-Tri.

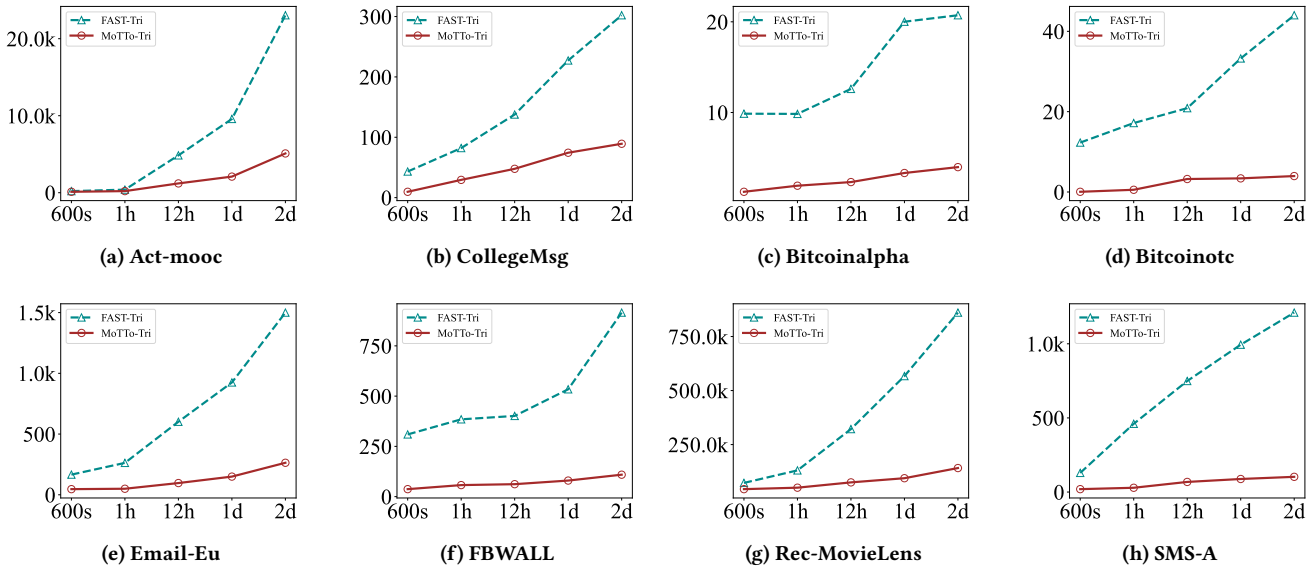


Figure 11: Comparison of counting times in milliseconds of algorithms for triangle temporal motifs *w.r.t.* δ

We selected δ = 600s, 3600s, 43200s, 86400s, and 172800s, represented on the x-axis as 600s, $1h$, $12h$, $1d$, and $2d$, respectively, where h denotes hours and d denotes days.

The experimental results of counting all temporal motifs are shown in Figure 10. From the figure, it can be seen that our algorithm outperforms the FAST algorithm on all datasets and different δ. This is because the time complexity of the FAST algorithm is $O(2\bar{d}^{\delta}|\mathcal{E}|)$, which is exponential in δ. Therefore, as δ increases, the counting time of FAST increases significantly. In contrast, the overall time complexity of our algorithm is about $O(\bar{d}^{\delta}(|\mathcal{V}_{\mathcal{W}}| - \epsilon)|\mathcal{Q}|)$. Although our algorithm is also affected by δ, due to the time-aware topology constraint method, many nodes that cannot generate temporal motif structures in the topology are pruned before counting begins, and the time complexity of the pruning process is not affected by δ. Therefore, compared to FAST, the counting time is reduced for each δ.

We compare our algorithm with the FAST algorithm in counting triangle temporal motifs under different δ, and the results are shown in Figure 11. The figure indicates that our algorithm outperforms the FAST algorithm on all datasets. For the same reason, since the time complexity of the FAST algorithm is exponential in δ, the time taken by FAST to count triangle temporal motifs also increases significantly as δ increases. In contrast, due to the time-aware topology constraint method, our algorithm prunes nodes before counting begins, and for the algorithm to count triangle temporal motifs, pruning efficiency is significant. Due to pruning, the increase in counting time that should have occurred with increasing δ becomes very slow for MoTTo. For instance, in the SMS-A dataset, the comparison of counting triangle temporal motifs with the FAST algorithm shows strong superiority. This can be observed from Figure 7, where the number of triangles in SMS-A is very low. Additionally, due to our time-aware topology constraint method, the majority of nodes are pruned, resulting in very few triangle temporal motif instance counting operations performed on the SMS-A dataset. In conclusion, we can deduce from the experimental results that our algorithm exhibits a slower growth in counting time as δ increases. This indicates that our algorithm reduces sensitivity to increasing δ.

6 Conclusion

In this work, we propose a scalable and accurate solution for the temporal motif counting problem. Specifically, we design an exact algorithm, MoTTo, for counting temporal motif instances for 36 motifs, including 24 star, four pair, and eight triangle temporal motifs. MoTTo involves two key strategies: One for removing nodes that cannot form temporal motifs based on our defined advanced matrix operations, and another for filtering out unrelated data partitions. Experiments verified that our proposed MoTTo significantly reduces the search space and improves the counting efficiency. Additionally, by leveraging the time-span concept, we also implemented MoTTo in a multi-threaded manner, achieving up to 31 times speedup. Our MoTTo enables the pruning of the topology before counting, while also supporting efficient and accurate parallel counting, making it particularly suitable for large-scale temporal networks.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under grant Nos. 62176243 and 62202438, China Postdoctoral Science Foundation under grant No. 2024T170868, the Postdoctoral Fellowship Program of CPSF under grant No. GZC20232500, and the Postdoctoral Project funded by Qingdao under grant No. QDBSH20240102093.

References

[1] István Albert and Réka Albert. 2004. Conserved network motifs allow protein–protein interaction prediction. *Bioinformatics* 20, 18 (2004), 3346–3352.

[2] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif counting beyond five nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 4 (2018), 1–25.

[3] Ziwei Chai, Yang Yang, Jiawang Dan, Sheng Tian, Changhua Meng, Weiqiang Wang, and Yifei Sun. 2023. Towards learning to discover money laundering subnetwork in massive transaction network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 14153–14160.

[4] Stephen A Cook. 2021. The complexity of theorem-proving procedures (1971). (2021).

[5] David Easley, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets: Reasoning about a highly connected world*. Vol. 1. Cambridge university press Cambridge.

[6] Iztok Fister Jr, Iztok Fister, and Matjaž Perc. 2016. Toward the discovery of citation cartels in citation networks. *Frontiers in Physics* 4 (2016), 240569.

[7] Zhongqiang Gao, Chuanqi Cheng, Yanwei Yu, Lei Cao, Chao Huang, and Junyu Dong. 2022. Scalable motif counting for large-scale temporal graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2656–2668.

[8] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[9] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. 2021. Knowledge graphs. *ACM Computing Surveys (Csur)* 54, 4 (2021), 1–37.

[10] Anand Padmanabha Iyer, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, Vladimir Braverman, and Ion Stoica. 2018. {ASAP}: Fast, approximate graph pattern mining at scale. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 745–761.

[11] Shweta Jain and C Seshadhri. 2020. The power of pivoting for exact clique counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 268–276.

[12] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th international conference on world wide web*. 495–505.

[13] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. 2011. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment* 2011, 11 (2011), P11005.

[14] Rohit Kumar and Toon Calders. 2018. 2scent: An efficient algorithm to enumerate all simple temporal cycles. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1441–1453.

[15] Geon Lee, Jihoon Ko, and Kijung Shin. 2020. Hypergraph motifs: concepts, algorithms, and discoveries. *arXiv preprint arXiv:2003.01853* (2020).

[16] Geon Lee, Seokbum Yoon, Jihoon Ko, Hyunju Kim, and Kijung Shin. 2024. Hypergraph motifs and their extensions beyond binary. *The VLDB Journal* 33, 3 (2024), 625–665.

[17] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[18] Paul Liu, Austin R Benson, and Moses Charikar. 2019. Sampling methods for counting temporal motifs. In *Proceedings of the twelfth ACM international conference on web search and data mining*. 294–302.

[19] Patrick Mackey, Katherine Porterfield, Erin Fitzhenry, Sutanay Choudhury, and George Chin. 2018. A chronological edge-driven approach to temporal subgraph isomorphism. In *2018 IEEE international conference on big data (big data)*. IEEE, 3972–3979.

[20] Shmoolik Mangan and Uri Alon. 2003. Structure and function of the feed-forward loop network motif. *Proceedings of the National Academy of Sciences* 100, 21 (2003), 11980–11985.

[21] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. 2004. Superfamilies of evolved and designed networks. *Science* 303, 5663 (2004), 1538–1542.

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

[23] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. 601–610.

[24] Noujan Pashanasangi and C Seshadhri. 2021. Faster and generalized temporal triangle counting, via degeneracy ordering. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1319–1328.

[25] Ali Pinar, Comandur Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*. 1431–1440.

[26] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. 2021. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.

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

[28] Ilie Sarpe and Fabio Vandin. 2021. Presto: Simple and scalable sampling techniques for the rigorous approximation of temporal motif counts. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 145–153.

[29] Comandur Seshadhri and Srikanta Tirthapura. 2019. Scalable subgraph counting: The methods behind the madness. In *Companion Proceedings of The 2019 World Wide Web Conference*. 1317–1318.

[30] Jingjing Wang, Yanhao Wang, Wenjun Jiang, Yuchen Li, and Kian-Lee Tan. 2020. Efficient sampling algorithms for approximate temporal motif counting. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 1505–1514.

[31] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2017. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2017), 73–86.

[32] Junliang Yu, Hongzhi Yin, Jundong Li, Qinyong Wang, Nguyen Quoc Viet Hung, and Xiangliang Zhang. 2021. Self-supervised multi-channel hypergraph convolutional network for social recommendation. In *Proceedings of the web conference 2021*. 413–424.

[33] Yichao Yuan, Haojie Ye, Sanketh Vedula Wynn Kaza, and Nishil Talati. 2023. Everest: GPU-Accelerated System For Mining Temporal Motifs. *arXiv preprint arXiv:2310.02800* (2023).