Finding Relaxed Temporal Network Motifs

Paper ID: 147

Abstract

There is an urgent need to relax temporal motifs to tolerate data errors. In this study, we first propose a proper definition of relaxed temporal motifs by allowing limited label mismatches, to tolerate label errors on a class of temporal networks, where the nodes and edges are fixed, while the edge labels vary regularly with timestamps. Second, we develop a low polynomial time algorithm with two optimized strategies to find all relaxed temporal motifs in a temporal network, based on an in-depth analysis. Third, we develop a theoretically faster incremental solution to efficiently handle the continuous updating scenario of temporal networks, by identifying affected edges and affected temporal motifs. Finally, we present an extensive experimental study to verify the efficiency and effectiveness of both our static and incremental methods.

CCS Concepts

• **Information systems → Network data models**.

Keywords

Temporal networks; temporal motifs; pattern relaxation

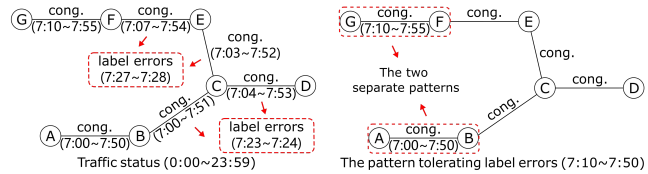
ACM Reference Format:

Paper ID: 147. 2026. Finding Relaxed Temporal Network Motifs. In *Proceedings of International Conference on Management of Data (SIGMOD '26)*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Network (or graph) motif discovery aims at identifying recurrent and statistically significant patterns or subgraphs in networks [42, 54], and is one of the fundamental graph problems [2, 61] (see, *e.g.*, [50, 61] for surveys). Network motifs provide valuable information on network functional abilities [42, 54], and have been extensively studied [5, 7, 11, 13, 30, 32, 38, 45, 56, 60]. To better analyze the dynamics in data analysis systems and applications that could be modeled as graphs or networks, temporal networks have drawn significant attention, suggesting the growing need for the study of temporal motif discovery [17, 35].

Existing studies of temporal motif discovery often reinterpret temporal motifs to meet the various demands of applications. Most studies propose temporal motifs, where the edges are attached with beginning timestamps and durations or only timestamps [8, 16, 19, 23–27, 36, 46, 47, 63]. These typically identify all the isomorphic subgraphs, where the timestamps of edges satisfy the same time orders and are limited in user-defined time windows. There are other



(a) A road network **(b) Traffic patterns**

Figure 1: Temporal motifs need relaxations

studies that propose temporal motifs as induced subgraphs, where node weights evolve in a consistent trend [10, 22], or edge labels and directions remain unchanged over a period [3]. Due to the high computational cost or the need for capturing more sensible patterns, there are studies starting to relax their topology constraints in the definitions of temporal motifs, such as utilizing dual simulation [39] instead of subgraph isomorphism to allow topology mismatches [55], and persistent k-core structures instead of fixed k-clique structures in a period to allow topology changes [31].

However, little attention has been paid to the discovery of temporal motifs in the presence of data errors caused by measurement errors or missing values [9, 14, 21, 37, 45].

Example 1: Consider a real-life example taken from the road network shown in Figure 1(a), where each edge represents a road monitored by a sensor that labels traffic status as either 'congested' (cong.) or 'fast', along with their timestamps at the minute level in one day. For simplicity, 'congested' edge labels are shown with time intervals, and 'fast' edge labels at all other timestamps are omitted. However, the roads 'BC' and 'CD' (between 7:23 and 7:24), as well as the roads 'CE' and 'EF' (between 7:27 and 7:28), encounter sensor faults, and are mislabeled as 'fast'.

A data analyst aims to identify topology patterns that indicate traffic jam areas lasting at least 30 minutes, but finds only two separate patterns (circled by dashed lines in Figure 1(b)) with single roads 'AB' (between 7:00 and 7:50) and 'FG' (between 7:10 and 7:55). This is because during the period, these roads adjacent to or linking these two patterns keep 'congested' status less than 30 minutes due to the sensor faults. In fact, these roads are mislabeled as 'fast' during the period. Hence, the two patterns should be merged into a single pattern (between 7:10 and 7:50) if label errors are handled appropriately. In this case, it is irrelevant to the relaxation of topology constraints [31, 55], and therefore cannot be addressed by existing studies on temporal motifs. To tackle this, a natural idea is to relax the exact road label matching constraint at certain timestamps to tolerate label mismatches. □

In this study, we focus on tolerating data errors for temporal motif discovery by allowing limited label mismatches in our temporal motifs on a significant class of temporal networks, where the nodes and edges are fixed, but the edge labels vary regularly with timestamps [6, 40, 41]. Road networks (as shown in Figure 1(a)) and energy transmission networks [20, 58] typically fall into this category, where sensor data can be noisy in urban computing [64].

However, the study of such temporal motifs faces several challenges. (1) How to propose a proper definition of temporal motifs

SIGMOD '26, May 31–June 05, 2026, Bangalore, India

ACM ISBN 978-1-4503-XXXX-X/25/06
https://doi.org/XXXXXXX.XXXXXXX

to tolerate data errors in real-world data? The existing temporal motifs only relax their topology constraints [31, 55], which cannot address these issues. (2) The efficiency of finding temporal motifs is significant. Most existing methods are computationally intractable, as (subgraph) isomorphism or exponential-time subgraph enumeration are unavoidable (unless P = NP) [3, 10, 16, 19, 22–26, 31, 36, 46, 47, 63]. To achieve speed-up, various techniques are adopted, such as sampling [27, 34, 51, 52, 57], dual simulation [39, 55], and parallel approaches [8, 15]. (3) Dynamic algorithms are essential for handling dynamic and continuously updated temporal networks, which have not been considered by existing studies.

Contributions & roadmap. We propose a concept of relaxed temporal motifs (or simply RTMs) to tolerate data errors for temporal networks, where the nodes and edges are fixed, but the edge labels vary regularly with timestamps, and develop both static and incremental algorithms to efficiently identify RTMs.

(1) We propose a proper definition of (maximal and non-expandable) RTMs (*i.e.,* subgraphs appearing continuously in sufficiently large time intervals with edge label mismatches occurring at a limited number of timestamps) to reinterpret the recurrent and statistically significant nature of motifs in temporal networks (Section 2). Different from most existing temporal motifs, our RTMs can tolerate data errors, and can be computed in low polynomial time.

(2) We develop an efficient algorithm to compute all the maximal and non-expandable RTMs in a temporal network in low polynomial time, based on the analyses of RTMs, and further develop two optimization strategies (Section 3).

(3) We develop a theoretically faster incremental algorithm with a better time complexity than its static counterpart. By identifying affected edges and affected RTMs, it efficiently handles the continuous updates of temporal networks (Section 4).

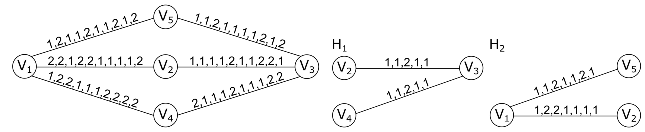
(4) Using both real-life and synthetic datasets, we conduct an extensive experimental study (Section 5). (a) Our static algorithm runs fast even on the large temporal networks with 400 thousand nodes and 1.6 billion edges. (b) Our incremental algorithm is much faster than its static counterpart, and is on average (2.72, 1.88, 2.05) times faster on datasets (BJData, EURData, SYNData), respectively. (c) Our RTMs are effective for practical applications with case studies on real-life datasets BJData and EURData.

The codes, datasets and full version (proofs and extra experiments) are available at https://github.com/CSeCodesData/FRTM.

2 Preliminary

In this section, we introduce the basic concepts and the problem statement, and we use the term 'intervals' instead of 'time intervals' for simplicity in the sequel.

Temporal networks. A *temporal network* $G(V, E, T_b, T_f, L)$ is a weighted undirected network with edge labels varying with timestamps (positive integers), where (1) V is a finite set of nodes, (2) $E \subseteq V \times V$ is a finite set of edges, in which (u, v) or $(v, u) \in E$ denotes an undirected edge between nodes u and v, (3) $[T_b, T_f]$ is an interval containing $(T_f - T_b + 1)$ timestamps, in which $T_b \le T_f$ are the beginning and finishing timestamps, respectively, and (4) L is a set of label functions such that for each timestamp $t \in [T_b, T_f]$, and L^t is a function that maps each edge $e \in E$ to a label (an integer



(a) Temporal network G with interval $[1, 10]$ **(b)** RTMs H_1 **and** H_2 **of** G **with intervals** $[3, 7]$ **and** $[3, 9]$**, respectively**

Figure 2: Running example

for simplicity). When it is clear from the context, we simply use $G(V, E, L)$ or $G(V, E, T_b, T_f)$ to denote a temporal network.

In temporal networks, *e.g.,* road networks, the nodes and edges are fixed, but the edge labels vary *w.r.t.* timestamps [6, 40, 41]. Intuitively, (1) a temporal network $G(V, E, L)$ essentially denotes a sequence of $T = T_f - T_b + 1$ standard networks, *i.e.,* $\langle G_1(V, E, L^{T_b}), \dots, G_{t-T_b+1}(V, E, L^t), \dots, G_{T_f-T_b+1}(V, E, L^{T_f})\rangle$, and (2) its edge labels $L^t(e)$ ($t \in [T_b, T_f]$) specify the states, distances or traveling duration [17, 58] of edges at a timestamp in a discrete way, *e.g.,* using integers to denote the fast/congested road traffic states.

We also call the temporal network $G(V, E, L)$ at timestamp t (*i.e.,* $G_t(V, E, L^t)$, $t \in [T_b, T_f]$) its *snapshot* at timestamp t.

We next define our *relaxed temporal motifs.*

Relaxed temporal motifs (or simply RTMs). A relaxed temporal motif $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f, L_s)$ is a connected subgraph of $G(V, E, T_b, T_f, L)$ ($\hat{T}_b, \hat{T}_f \in [T_b, T_f]$) that satisfies the following:

(1) G_s lasts at least k continuous snapshots, *i.e.,* $\hat{T}_f - \hat{T}_b + 1 \ge k$, where k is referred to as the frequency threshold,

(2) the edge labels at timestamps $\hat{T}_b$ and $\hat{T}_f$ are equal, *i.e.,* $L^{\hat{T}_b}(e) = L^{\hat{T}_f}(e)$ for each edge $e \in E_s$,

(3) the total number of label mismatches for each edge $e \in E_s$, *i.e.,* $L^t(e) \ne L^{\hat{T}_b}(e)$ with $t \in [\hat{T}_b, \hat{T}_f]$, does not exceed $\delta \times (\hat{T}_f - \hat{T}_b + 1)$, where δ is referred to as the global relaxation bound, and

(4) the number of continuous label mismatches for each edge $e \in E_s$ does not exceed c, which is referred to as the local relaxation bound. That is, for any edge e and interval $[\hat{t}_b, \hat{t}_f]$ in $[\hat{T}_b, \hat{T}_f]$ such that $L^t(e) \ne L^{\hat{T}_b}(e)$ for all $t \in [\hat{t}_b, \hat{t}_f]$, $\hat{t}_f - \hat{t}_b + 1 \le c$.

Here, $0 \le c < k$ are integers, and $0 \le \delta \le 1$. We next illustrate the rationale for introducing the two relaxation bounds.

Example 2: Recall the road network in Example 1.

(1) The global relaxation bound δ ensures that the total label mismatches of each edge only occur at a limited number of timestamps.

With the global relaxation bound $\delta = 5\%$, one can verify that the entire traffic pattern within interval [7:10, 7:50] in Figure 1(b) can be identified, instead of two separate ones. In this way, the mislabeled roads 'BC' and 'CD' between 7:23 and 7:24, as well as the mislabeled roads 'CE' and 'EF' between 7:27 and 7:28 are handled.

(2) The local relaxation bound c ensures that each occurrence of edge label mismatches continues only within a limited number of timestamps. With the global relaxation bound $\delta = 5\%$, one can also identify a traffic pattern that is exactly the same as the pattern in Figure 1(b) except that each road is labeled as 'fast' within interval [0:00, 23:59]. However, this pattern essentially treats all the edge labels within [0:00, 23:59] as 'fast', and it does not make sense for the edges labeled as 'congested' within [7:10, 7:50].

With any local relaxation bound $2 \le c < k$, one can verify that this unreasonable pattern cannot be identified any more. That is, the local relaxation bound remedies the side effects of the global relaxation bound for tolerating data errors. □

Maximal RTMs. An RTM $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f)$ is maximal if and only if there exist no RTMs $G_{s'}(V_{s'}, E_{s'}, \hat{T}_b, \hat{T}_f)$ such that G_s is a subgraph of $G_{s'}$, and $G_{s'}$ has edges not in G_s.

Expandable RTMs. An RTM $G_s(V_s, E_s, \hat{t}_b, \hat{t}_f)$ is expandable if and only if there is another RTM $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f)$ such that $\hat{T}_b \le \hat{t}_b$ and $\hat{t}_f \le \hat{T}_f$, and for each edge $e \in E_s$, $L^{\hat{t}_b}(e) = L^{\hat{T}_b}(e)$.

We also say that an RTM is *non-expandable* if it is not expandable.

Example 3: Consider the temporal network G (possibly with noisy labels) with 5 nodes, 6 edges and 10 timestamps in Figure 2(a).

Let the frequency threshold $k = 5$, and relaxation bounds $\delta = 30\%$ and $c = 2$. One can identify two RTMs H_1 and H_2 shown in Figure 2(b) with intervals $[3, 7]$ and $[3, 9]$, respectively.

RTM H_1 is maximal, as it has no adjacent edges e not in H_1 such that $L^3(e) = L^7(e)$, and the label mismatches of e satisfy the relaxation bounds δ and c in $[3, 7]$. However, H_1 is expandable, as $L^2(e) = L^7(e)$ for each edge $e \in H_1$, and the label mismatches of e satisfy the relaxation bounds δ and c in $[2, 7]$. One can verify that RTM H_2 in Figure 2(b) is both maximal and non-expandable. □

Problem statement. Given a temporal network $G(V, E, T_b, T_f, L)$, a frequency threshold k, a global relaxation bound δ, and a local relaxation bound c, the problem is to find all the maximal and non-expandable RTMs $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f)$ in G.

Example 4: Given the temporal network G in Figure 2(a), the frequency threshold $k = 5$, the global relaxation bound $\delta = 3\%$ and the local relaxation bound $c = 2$, H_2 in Figure 2(b) is the maximal and non-expandable RTM returned. □

Applications. Temporal motifs are generally used to identify frequent patterns of interactions in various networks under specific constraints [35]. Our proposed RTMs can be used to discover traffic patterns with long-term congested or bottleneck roads in road networks [28, 44], and energy consumption patterns with continuous high energy demand signals in transmission networks [20, 48]. Furthermore, our RTMs are suitable to tolerate data errors, such as short-term traffic mitigation and sudden sensor faults.

Remarks. (1) Different from existing studies on temporal motifs, we focus on a special but significant class of temporal networks, where the nodes and edges are fixed, but the edge labels vary regularly with timestamps [6, 40, 41]. However, our RTMs can be extended to handle varying edges by treating missing edges as edges with virtual labels, and to handle varying numerical edge weights by discretizing them into ranges (*i.e.,* labels) using appropriate quantiles. (2) We focus on RTMs, *i.e.,* subgraphs appearing continuously in sufficiently large time intervals with edge label mismatches occurring at a limited number of timestamps. Most existing studies on temporal motifs [3, 10, 16, 19, 22–26, 31, 36, 46, 47, 63] are computationally intractable, and do not consider data errors, while our RTMs can be computed in low polynomial time. (3) Different from those studies on relaxing topology constraints in [31, 55], our RTMs focus on relaxing label constraints to tolerate data errors, which has never been considered before.

3 Finding Relaxed Temporal Motifs

In this section, we explain how to find all the maximal and non-expandable RTMs, given a temporal network $G(V, E, T_b, T_e, L)$, a frequency threshold k, a global relaxation bound δ and a local relaxation bound c. Without loss of generality, we assume that $T_b = 1$ and $T_e = T$ in the sequel. The main result is stated below.

Theorem 1: *Given a temporal network $G(V, E, 1, T, L)$, a frequency threshold k, and relaxation bounds δ and c, there exists an algorithm that finds all maximal and non-expandable* RTMs *in $O(avgI \cdot T^2|E|)$ time. Here, avgI is the average number of intervals to be checked when determining whether an* RTM *is expandable or not.* □

3.1 Analyses of RTMs

To obtain any maximal and non-expandable RTM $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f)$ ($\hat{T}_b, \hat{T}_f \in [1, T]$), we need to ensure the following: (1) for each edge $e \in E_s$, $L^{\hat{T}_b}(e) = L^{\hat{T}_f}(e)$ and the label mismatches of e satisfy the relaxation bounds δ and c, (2) the maximal property of G_s, and (3) the non-expandable property of G_s. These lead to three challenges.

Challenge 1: The first one is how to efficiently identify the edges $e \in E$ such that $L^{\hat{T}_b}(e) = L^{\hat{T}_f}(e)$, and the label mismatches of the edge e satisfy the relaxation bounds δ and c in interval $[\hat{T}_b, \hat{T}_f]$.

Let $\mathsf{S}[m, i]$ ($1 \le m \le T - k + 1$ and $m + k - 1 \le i \le T$) be the set of edges $e \in E$ such that $L^m(e) = L^i(e)$, and the label mismatches of the edge e satisfy the relaxation bounds δ and c in interval $[m, i]$. When $\delta = 0$ or $c = 0$, one can verify that $\mathsf{S}[m, h] \subseteq \mathsf{S}[m, i]$ for $m + k - 1 \le i < h \le T$. However, this property does not hold if $\delta, c > 0$, as shown below.

Proposition 1: *For any $\delta, c > 0$, there exists an edge e such that $e \in \mathsf{S}[m, h]$ but $e \notin \mathsf{S}[m, i]$ ($m + k - 1 \le i < h \le T$).* □

Proposition 1 shows that S edge sets have no containment relations due to the relaxation bounds δ and c, which complicates the computation of these edge sets. Hence, we next define R edge sets.

Let $\mathsf{R}[m, i] = \mathsf{S}[m, i] \setminus \bigcup_{i<h\le T} \mathsf{S}[m, h]$ for $m + k - 1 \le i < T$, and $\mathsf{R}[m, T] = \mathsf{S}[m, T]$. In other words, $\mathsf{R}[m, i]$ is the set of edges e such that $e \in \mathsf{S}[m, i]$ but $e \notin \mathsf{S}[m, j]$ for any $j > i$. Since R edge sets are easier to compute, the first challenge is transformed into obtaining these edge sets. As will be seen shortly, we design a data structure DEL-Table to facilitate the computation.

Challenge 2: Once we obtain R edge sets, the next challenge is how to efficiently generate maximal RTMs by these edge sets.

For convenience, we denote $\bigcup_{i\le h\le T} \mathsf{R}[m, h]$ as $\mathsf{S}^*[m, i]$ for $m + k - 1 \le i \le T$. We have the following observation.

Proposition 2: *The label mismatches of each edge $e \in \mathsf{S}^*[m, i]$ satisfy the relaxation bound c in all the intervals $[m, j]$ for $j \le i$.* □

Proposition 2 reveals that, to generate maximal RTMs, we just need to remove the edges e from $\mathsf{S}^*[m, i]$ such that $L^m(e) \ne L^i(e)$ or the label mismatches of edge e violate the relaxation bound δ.

A trivial approach is to examine each edge $e \in \mathsf{S}^*[m, i]$ to determine whether the condition $L^m(e) \ne L^i(e)$ holds, or whether the label mismatches of e violate the relaxation bound δ. However, it is unnecessary to check all these edges. As will be seen shortly, we can record auxiliary information when computing the R edge sets, allowing us to directly obtain these edges that need to be removed.

For an edge set $\mathsf{S}^*[m, i]$, we can derive a temporal subgraph $G_s(V(\mathsf{S}^*[m, i]), \mathsf{S}^*[m, i], m, i)$, denoted as $G_s(\mathsf{S}^*[m, i])$. There is a close connection between maximal RTMs and connected components (or simply ccs), as shown below.

Proposition 3: *After removing edges e from $G_s(\mathsf{S}^*[m, i])$ ($1 \le m \le T - k + 1$ and $m + k - 1 \le i \le T$) such that $L^m(e) \ne L^i(e)$ or the*

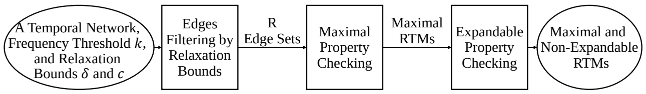


Figure 3: The overall algorithm framework

label mismatches of edge e violate the relaxation bound δ, each newly generated cc *is a maximal* RTM *with interval* $[m, i]$. □

By Proposition 3, we have an efficient scheme to generate maximal RTMs. We adopt the *right-to-left (R2L)* scheme, *i.e.,* generating maximal RTMs from intervals $[m, T]$ to $[m, m+k-1]$. For each interval $[m, i]$, we can add edges from the set $\mathsf{R}[m, i]$ to the ccs of the subgraph $G_s(\mathsf{S}^*[m, i+1])$ $(i < T)$ or an empty graph $(i = T)$, and then remove edges e such that $L^m(e) \neq L^i(e)$ or the label mismatches of e violate the relaxation bound δ when $i < T$. After that, we can obtain the maximal RTMs by newly generated ccs. Note that there is a *left-to-right (L2R)* scheme, *i.e.,* generating maximal RTMs from intervals $[m, m+k-1]$ to $[m, T]$. The *L2R* scheme adds the same number of edges as the *R2L* scheme, but removes more edges ($\sum_{m+k-1 \leq i < T} |\mathsf{R}[m, i]|$). Hence, the *R2L* scheme is better.

Challenge 3: If a maximal RTM with interval $[m, i]$ is expandable, the label mismatches of each edge in the RTM may satisfy bounds δ and c in any other intervals containing $[m, i]$. Hence, the final challenge is how to reduce the intervals to be checked, when determining whether a maximal RTM is expandable or not.

Proposition 4: *If a generated maximal* RTM *with interval* $[m, i]$ *contains an edge* $e \in \mathsf{R}[m, i]$, *it cannot appear in any maximal* RTM*s with intervals* $[m, h]$ *for* $i+1 \leq h \leq T$. □

Proposition 4 reveals that it is unnecessary to check the intervals $[m, h]$ for $i+1 \leq h \leq T$, when determining whether a maximal RTM containing edges in $\mathsf{R}[m, i]$ is expandable or not.

We further reduce the number of intervals to be checked in other cases. Assume that an edge $e \in \mathsf{R}[p_1, j_1], \ldots, \mathsf{R}[p_x, j_x]$ $(p_1, \ldots, p_x \leq m$ and $j_1, \ldots, j_x \geq i)$. Here, x is the number of R edge sets containing the edge e. We denote scope$[e]$ as the interval $[\min(p_1, \ldots, p_x), \max(j_1, \ldots, j_x)]$, and make the following observation.

Proposition 5: *Let* $[l, r]$ *be the intersection of* scope$[e]$ *for all edges e in the* RTM. *We can determine whether a maximal* RTM *with interval* $[m, i]$ *is expandable by checking whether it appears in other maximal* RTM*s with intervals* $[n, h]$ *for* $l \leq n \leq m$ *and* $i \leq h \leq r$. □

Proposition 5 reveals that we can compute the intersection of scope$[e]$ for all edges e in the RTM to reduce the number of intervals to be checked. Note that the label mismatches of each edge e in the RTM satisfy the relaxation bound c in any intervals $[n, h]$, as defined by the R edge sets. Therefore, we only need to verify two conditions: $L^n(e) = L^h(e)$, and the label mismatches of edge e satisfy the relaxation bound δ. As will be seen shortly, we use DEL-Table to facilitate this computation. Moreover, since the left endpoints of these intervals to be checked are not greater than m, a *top-to-bottom (T2B)* scheme is appropriate, *i.e.,* computing RTMs with intervals $[m, m+k-1], \cdots, [m, T]$ for each m from 1 to $T-k+1$.

By Propositions 3 & 5, we know that the *top-to-bottom (T2B) and right-to-left (R2L)* scheme is a better choice to compute the maximal and non-expandable RTMs for all the intervals.

3.2 Overall Algorithm Framework

Proposition 1 indicates that it is difficult to compute the S edge sets directly, so we choose to compute the R edge sets instead. Moreover, based on Propositions 2 & 3, we generate maximal RTMs for an interval $[m, i]$, by removing edges e from $G_s(\mathsf{S}^*[m, i])$ such that either $L^m(e) \neq L^i(e)$ or the label mismatches of e violate the relaxation bound δ. By comparing the *R2L* and *L2R* schemes, we choose the *R2L* scheme, *i.e.,* generating maximal RTMs from intervals $[m, T]$ to $[m, m+k-1]$. Based on Propositions 4 & 5, we reduce the number of intervals to be checked when determining whether a maximal RTM is expandable by computing intervals scope$[e]$ for edges e in the RTM.

Our overall algorithm framework is shown in Figure 3. Given a temporal network, a frequency threshold k, and relaxation bounds δ and c, it adopts the *T2B and R2L* scheme to compute RTMs falling into intervals $[m, i]$, for each m from 1 to $T-k+1$, and for each i from T to $m+k-1$. For each m, (1) it first filters edges $e \in E$ such that $L^m(e) = L^i(e)$, and the label mismatches of e satisfy the relaxation bounds δ and c to obtain edge sets $\mathsf{R}[m, i]$ for $m+k-1 \leq i \leq T$. After that, it deals with intervals $[m, i]$ for each i from T to $m+k-1$. For each i, (2) it generates the maximal RTMs for each interval $[m, i]$ by checking the maximal property of $G_s(\mathsf{S}^*[m, i])$, *i.e.,* removing edges e such that either $L^m(e) \neq L^i(e)$ or the label mismatches of e violate the relaxation bound δ. (3) Once it generates the maximal RTMs for an interval $[m, i]$, it immediately checks the expandable property of the RTMs. Finally, it obtains the maximal and non-expandable RTMs. The details are introduced in the following Sections 3.3-3.5.

3.3 Edges Filtering by Relaxation Bounds

We first introduce how to filter all edges $e \in E$ such that $L^m(e) = L^i(e)$, and the label mismatches of e satisfy relaxation bounds δ and c. This allows us to obtain the edge sets $\mathsf{R}[m, i]$ for $m+k-1 \leq i \leq T$. As defined by the R edge sets, for each edge $e \in \mathsf{R}[m, i]$, $L^m(e) = L^i(e)$ and the label mismatches of e satisfy relaxation bounds δ and c in interval $[m, i]$, while any of these conditions are violated in intervals $[m, h]$ for all $h > i$. Hence, by such an interval $[m, i]$ (referred to as *maximum valid intervals*), we know that $e \in \mathsf{R}[m, i]$ if the interval exists, otherwise $e \notin \mathsf{R}[m, h]$ for all $h \in [m+k-1, T]$. To efficiently identify these intervals, we design a data structure called DEL-Table. It can be constructed in $O((T + |L|)|E|)$ time and is independent of any specific values of δ, c and k.

DEL-Table. It is a $(T + |L|) \times |E|$ table, which stores the label perseveration information of each edge *w.r.t.* timestamps in different time orders. For each edge e and timestamp t, it stores (1) the label lab_t of edge e at timestamp t, (2) the number dif_t of labels different from $L^t(e)$ in interval $[1, t]$, and (3) the numbers bef_t and aft_t which can be either positive or negative. The positive number bef_t (resp. aft_t) indicates the number of consecutive timestamps such that the label of edge e is lab_t until timestamp t in time order (resp. in reverse order). The negative number bef_t (resp. aft_t) indicates the time difference between timestamp t and the most recent timestamp t' before t (resp. after t) such that $L^{t'}(e) = L^t(e)$. Note that it stores negative numbers bef_t (resp. aft_t) at each timestamp when edge e changes its label in time order (resp. in reverse order). Moreover, for each edge e and label lab, it stores the last timestamp $tail_{lab}$ when the label of edge e is lab, which is used for dynamic updates.

Figure 4 depicts DEL-Table for edges (v_1, v_2) and (v_1, v_5) only in Figure 2(a), where $bef_t = -\infty$ (resp. $aft_t = -\infty$) indicates that there are no labels matching lab_t before (resp. after) timestamp

Edge		t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10		lab=1	lab=2
(v_1,v_2)	lab_t	2	2	1	2	2	1	1	1	1	2	$tail_{lab}$	9	10
	bef_t	-∞	2	-∞	-2	2	-3	2	3	4	-5			
	aft_t	2	-2	-3	2	-5	4	3	2	-∞	-∞			
	dis_t	0	0	2	1	1	4	4	4	4	5			
(v_1,v_5)	lab_t	1	2	1	1	2	1	1	2	1	2	$tail_{lab}$	9	10
	bef_t	-∞	-∞	-2	2	-3	-2	2	-3	-2	-2			
	aft_t	-2	-3	2	-2	-3	2	-2	-2	-∞	-∞			
	dis_t	0	1	1	1	3	2	2	5	3	6			

Figure 4: DEL-Table **for edges** (v_1, v_2) **and** (v_1, v_5)

t. Given a timestamp t, we can identify the interval $[n, h] = [t - \max(bef_t, 1)+1, t+\max(aft_t, 1)-1]$ containing t such that the edge label matches lab_t. We can also identify the most recent timestamp before (resp. after) this interval, where the label matches lab_t (if it exists), by $n + bef_n$ (resp. $h - aft_h$). Moreover, for any intervals $[m, i]$, where $lab_m = lab_i$, we can obtain the number of labels different from lab_m in $[m, i]$ by $dif_i - dif_m$. Note that any RTMs in the interval $[m, i]$ do not consist of edges with $lab_m \neq lab_i$.

Example 5: Take edge (v_1, v_5) as an example. Given a timestamp $t = 7$, we know that the edge (v_1, v_5) keeps the label $lab_7 = 1$ in the interval $[7-\max(bef_7, 1)+1, 7+\max(aft_7, 1)-1] = [6, 7]$. The most recent timestamp before (resp. after) $[6, 7]$, where the edge (v_1, v_5) has the same label '1' is $6+bef_6 = 4$ (resp. $7-aft_7 = 9$). The number of labels different from the label '1' in $[1, 7]$ is $dif_7 - dif_1 = 2$. □

We next explain how to use DEL-Table to identify the R edge set for edge e. Starting from the timestamp m, we scan DEL-Table for edge e. For the current timestamp t, if $L^t(e) = L^m(e)$ (resp. $L^t(e) \neq L^m(e)$), we check the last timestamp $\hat{t} = t+\max(aft_t, 1)-1$ (resp. $t - aft_{t-1} - 2$) in consecutive timestamps during which the label of e matches $L^m(e)$ (resp. continuous label mismatches) after t, while maintaining the following information in interval $[m, \hat{t}]$: (1) the maximum number $intvL$ of continuous label mismatches for edge e, and (2) the last timestamp $lastT$ such that $L^{lastT}(e) = L^m(e)$ and the label mismatches of e satisfy the relaxation bounds δ and c in interval $[m, lastT]$. After that, the timestamp t is updated to $\hat{t} + 1$. The process continues until one of the following conditions is satisfied: (1) $intvL > c$ (*i.e.,* violating the relaxation bound c), and (2) $t = T$ (*i.e.,* the last timestamp). If $lastT \geq m + k - 1$, edge $e \in$ R$[m, lastT]$; otherwise, $e \notin$ R$[m, h]$ for all $h \geq m+k-1$. Hence, we do not need to check all timestamps to identify R edge sets.

We next use an example to show the above process.

Example 6: We consider frequency threshold $k = 5$, relaxation bounds $\delta = 0.3$ and $c = 2$, and the starting timestamp $m = 1$.

Take the edge (v_1, v_2) in Figure 4 as an example. (1) For timestamp $t = m = 1$, we check the timestamp $1 + aft_1 - 1 = 2$, and update $intvL = 0$, $lastT = 2$. After that, t is updated to 3. (2) For $t = 3$, we check the timestamp $3-aft_2-2 = 3$, and update $intvL = 1$. After that, t is updated to 4. (3) For $t = 4$, we check the timestamp 5, and update $lastT = 5$. After that, t is updated to 6. (4) For $t = 6$, we check the timestamp 9, and update $intvL = 4$. The process stops ($4 > c$). As $lastT \geq m + k - 1$, $(v_1, v_2) \in$ R$[1, lastT]$, *i.e.,* R$[1, 5]$. □

Procedure edgeFilter. We next present the details of the procedure edgeFilter to compute the edge sets R$[m, i]$ ($i \in [m + k - 1, T]$) for each $m \in [1, T - k + 1]$, as shown in Figure 5. The procedure takes as input the DEL-Table for the temporal network G, the frequency threshold k, relaxation bounds δ and c, arrays maxIntv, checkT and vioT, and the timestamp m, and returns the edge sets R$[m, i]$ and the updated arrays maxIntv, checkT and vioT. Here, for each edge $e \in E$ and each label $L^m(e)$, (1) the array maxIntv maintains all the maximum valid intervals $[n, h]$ ($n \leq m$) such that $L^n(e) = L^m(e)$, (2) the array checkT maintains the last checked timestamp when the process of scanning DEL-Table stops, and (3) the array vioT maintains intervals containing all timestamps $t \in [m+k-1, \text{checkT}[e, L^m(e)]]$ such that $L^t(e) \neq L^m(e)$, or the label mismatches of e violate the relaxation bounds δ and c in $[m, t]$. The arrays maxIntv and vioT are initialized to $\emptyset$, and the array checkT is initialized to 0. We dynamically maintain these arrays to reduce the time cost of computing the R edge sets for different m, and to facilitate the generation of maximal RTMs (will be introduced in Section 3.4).

Procedure edgeFilter
Input: DEL-Table for $G(V, E, 1, T, L)$, frequency threshold k, relaxation bounds δ and c, arrays maxIntv, checkT and vioT, and timestamp m.
Output: R edge sets and updated arrays maxIntv, checkT and vioT.

1. R$[m, i] := \emptyset$ for all $i \in [m + k - 1, T]$;
2. **for each** edge e **in** E
3.   **if** $m = 1$ **then** /*Case 1*/
4.   Obtain the R edge set for e by scanning DEL-Table from timestamp 1 and update maxIntv, checkT and vioT;
5.   **elseif** $L^m(e) = L^{m-1}(e)$ **then** /*Case 2*/
6.   Update vioT$[e, L^m(e)]$ by DEL-Table;
7.   Obtain the R edge set for e directly by maxIntv and updated vioT;
8.   **else** Update vioT$[e, L^m(e)]$ by DEL-Table; /*Case 3*/
9.   **if** checkT$[e, L^m(e)] < m$ **then**
10.   Obtain the R edge set for e by scanning DEL-Table from timestamp m and update maxIntv, checkT and vioT;
11.   **else** Obtain the R edge set for e directly by maxIntv and updated vioT;
12. **return** (R, maxIntv, checkT and vioT).

Figure 5: Procedure edgeFilter

When computing R edge sets, there are three cases to consider. Assume that before procedure edgeFilter starts, $[n, j]$ is the interval with the maximum right endpoint in maxIntv$[e, L^m(e)]$ (if it exists).

Case 1: $m = 1$. We can obtain the R edge set for e by scanning DEL-Table from timestamp 1, and update vioT$[e, L^m(e)]$, by adding all the timestamps $t \geq m + k - 1$ such that $L^t(e) \neq L^m(e)$ or the label mismatches of e violate the relaxation bound δ in interval $[m, t]$. When the scanning process stops, we update checkT$[e, L^m(e)]$ to the currently checked timestamp. If $e \in$ R$[m, lastT]$ and $lastT > j$, we need to add the interval $[m, lastT]$ into maxIntv$[e, L^m(e)]$.

Case 2: $m \neq 1$ and $L^m(e) = L^{m-1}(e)$. Since there are more timestamps $t \leq$ checkT$[e, L^m(e)]$ such that the label mismatches of e violate the relaxation bound δ in intervals $[m, t]$, compared with $[m - 1, t]$, we need to first update vioT$[e, L^m(e)]$. After that, we can obtain the R edge set for e by maxIntv and updated vioT. (1) If maxIntv$[e, L^m(e)] = \emptyset$ or $j < m + k - 1$, we know that $e \notin$ R$[m, h]$ for all $h \geq m + k - 1$. (2) Otherwise, we need to find the maximum timestamp $t \leq j$ such that $t \notin$ vioT$[e, L^m(e)]$. If the timestamp t exists, $e \in$ R$[m, t]$, otherwise $e \notin$ R$[m, h]$ for all $h \geq m + k - 1$.

Case 3: $m \neq 1$ and $L^m(e) \neq L^{m-1}(e)$. We also need to update intervals in vioT$[e, L^m(e)]$, as the label mismatches of e may satisfy the relaxation bound δ in intervals $[m, t]$ for certain timestamps t in vioT$[e, L^m(e)]$. We next need to decide how to obtain the R edge set for e. There are two cases to consider. (1) If checkT$[e, L^m(e)] \geq m$, it indicates we scanned DEL-Table and stopped when the label mismatches of e violate the relaxation bound c in $[m, \text{checkT}[e, L^m(e)]]$, or checkT$[e, L^m(e)] = T$. We can obtain the R edge set along the same lines as Case 2. (2) If checkT$[e, L^m(e)] < m$, we need to obtain the R edge set along the same lines as Case 1 from timestamp m.

Procedure edgeFilter first initializes all the R edge sets (line 1), and then executes for each edge $e \in E$ (line 2). For Case 1, it obtains the R edge set for e by scanning DEL-Table from timestamp 1, and updates arrays maxIntv, checkT and vioT (lines 3-4). For Case 2, it updates the array vioT, and obtains the R edge set for e directly by arrays maxIntv and vioT (lines 5-7). For Case 3, it updates the array vioT (line 8), and obtains the R edge set along the same line as Case 1, if checkT$[m, L^m(e)] < m$ (lines 9-10). Otherwise, it obtains the R edge set along the same line as Case 2 (line 11). Finally, it returns R edge sets and updated arrays maxIntv, checkT and vioT (line 12).

We next illustrate procedure edgeFilter with an example.

Example 7: Consider the temporal network G in Figure 2(a), frequency threshold $k = 5$, and relaxation bounds $\delta = 30\%$ and $c = 2$. The procedure executes with $m = 3$. Take edge (v_1, v_5) as an example. In this case, checkT$[(v_1, v_5), 1]$=10, maxIntv$[(v_1, v_5), 1]$=$\{[1, 7]\}$, and vioT$[(v_1, v_5), 1] = \{[5, 5], [8, 10]\}$. As $L^2((v_1, v_5)) \neq L^3((v_1, v_5))$ (Case 3), it updates vioT$[(v_1, v_5), 1]$ to $\{[8, 8], [10, 10]\}$, and then identifies $(v_1, v_5) \in$ R$[3, 9]$ ($9 \notin$ vioT$[(v_1, v_5), 1]$). □

Complexity. Procedure edgeFilter takes $O(T|E|)$ time and $O((T + |L|)|E|) = O(T|E|)$ space. Here, $|L|$ is the number of label types.

3.4 Maximal Property Checking

We next introduce how to generate the maximal RTMs for each interval $[m, i]$ ($1 \leq m \leq T-k+1$ and $m+k-1 \leq i \leq T$) by checking the maximal property of $G_s(\mathsf{S}^*[m, i])$. By Propositions 2 & 3 in Section 3.1, we need to maintain connected components (or simply ccs) of the subgraph $G_s(\mathsf{S}^*[m, i])$ for interval $[m, i]$, and remove edges e such that $L^m(e) \neq L^i(e)$ or the label mismatches of e violate the relaxation bound δ. After that, each generated cc corresponds to a maximal RTM. Note that when dealing with $[m, i]$, we have already obtained all the edge sets R$[m, i]$ for $i \in [m + k - 1, T]$, as well as the arrays maxIntv and vioT for edges in R$[m, i]$.

To generate the maximal RTMs efficiently, we dynamically maintain the cc set CC$[i, T]$ for $G_s(\mathsf{S}^*[m, i])$, and the cc set checkCC$[i, T]$ to record the ccs in CC$[i, T]$ that have edges to be removed. For each cc, we dynamically maintain a set vioTS and an interval ccScope. (1) The set vioTS maintains the intervals containing all the timestamps within $[m + k - 1, i]$ from vioT$[e, L^m(e)]$ for each edge e in the cc, where each element is a pair $\langle e, [l, r]\rangle$ of an edge e and an interval $[l, r]$. If vioTS contains an element for e that includes the timestamp i, the edge e needs to be removed. Hence, we can obtain those edges to be removed directly. (2) The interval ccScope maintains the intersection of scope$[e]$ (its definition is provided in Section 3.1) for each edge e in the cc, where scope$[e]$ can be computed as the union of intervals with right endpoints larger than i in maxIntv$[e, L^m(e)]$. Note that the interval ccScope is used in the expandable property checking of maximal RTMs (will be introduced in Section 3.5).

Procedure maxCheck. We next present the details of procedure maxCheck shown in Figure 6. It takes as input the edge set R$[m, i]$, the arrays maxIntv, vioT and scope, the interval $[m, i]$, and the cc sets CC$[i + 1, T]$ and checkCC$[i + 1, T]$, and returns the cc sets CC$[i, T]$ and checkCC$[i, T]$, the set maxCC of ccs corresponding to maximal RTMs for $[m, i]$, and the updated array scope. For convenience, CC$[T + 1, T]$ and checkCC$[T + 1, T]$ are empty sets.

Procedure maxCheck first initializes the cc sets CC$[i, T]$ to CC$[i+1, T]$, checkCC$[i, T]$ to checkCC$[i + 1, T]$, and maxCC to an empty set (line 1). It next dynamically maintains ccs in CC$[i, T]$ by adding edges in R$[m, i]$, and adds updated ccs to checkCC$[i, T]$ for maximal property checking (lines 2-11). There are three cases for each edge $e \in$ R$[m, i]$. (1) If e is disconnected from any cc in CC$[i, T]$, it creates a new cc G_s having edge e only, initializes vioTS and ccScope by vioT$[e, L^m(e)]$ and maxIntv$[e, L^m(e)]$, respectively, and adds G_s to CC$[i, T]$ and checkCC$[i, T]$ (lines 3-5). (2) If e is connected to only one cc $G_s \in$ CC$[i, T]$, it adds edge e to G_s, and updates G_s.vioTS and G_s.ccScope by vioT$[e, L^m(e)]$ and maxIntv$[e, L^m(e)]$, respectively. It next adds G_s to checkCC$[i, T]$ if $G_s \notin$ checkCC$[i, T]$ (lines 6-8). (3) If e is connected to two ccs G_s and $G_{s'}$, it creates a new cc $G_{ss'}$ by combining e with G_s and $G_{s'}$, and computes vioTS by G_s.vioTS $\cup$ $G_{s'}$.vioTS and vioT$[e, L^m(e)]$, as well as ccScope by G_s.ccScope $\cap$ $G_{s'}$.ccScope and maxIntv$[e, L^m(e)]$ (lines 9-10). Then it replaces G_s and $G_{s'}$ with $G_{ss'}$ in CC$[i, T]$ and checkCC$[i, T]$ (line 11). For each cc $G_s \in$ checkCC$[i, T]$, it updates G_s.ccScope, G_s.vioTS and scope$[e]$ for all edges $e \in G_s$, and obtains the set E_{rm} of edges to be removed by G_s.vioTS (lines 12-14). If E_{rm} is an empty set, it moves G_s from checkCC$[i, T]$ to maxCC, as G_s corresponds to a maximal RTM for $[m, i]$ (lines 15-16). Otherwise, it adds the recomputed ccs in $G_s \setminus E_{rm}$ to maxCC (line 17). Note that G_s in checkCC$[i, T]$ remains unchanged, as it may correspond to a maximal RTM for other intervals. Finally, it returns sets maxCC, CC$[i, T]$, checkCC$[i, T]$, and the updated array scope (line 18).

Procedure maxCheck
Input: the edge set R$[m, i]$, cc sets CC$[i+1, T]$ and checkCC$[i+1, T]$, arrays scope, maxIntv and vioT, and the interval $[m, i]$.
Output: cc sets maxCC, CC$[i, T]$ and checkCC$[i, T]$, and the updated array scope.

```
1.  CC[i,T] := CC[i+1,T], checkCC[i,T] := checkCC[i+1,T], maxCC := ∅;
2.  for each e in R[m,i]
3.     if e is disconnected from any cc in CC[i,T] then     /*Case 1*/
4.        G_s := a new cc having edge e only, with initialized vioTS and ccScope;
5.        Add G_s into checkCC[i,T] and CC[i,T];
6.     if e is connected to only one cc G_s in CC[i,T] then     /*Case 2*/
7.        Add e into G_s, and update G_s.vioTS and G_s.ccScope;
8.        Add G_s into checkCC[i,T] if G_s ∉ checkCC[i,T];
9.     if e is connected to two ccs G_s and G_s' in CC[i,T] then   /*Case 3*/
10.       G_ss' := the cc consisting of G_s, G_s' and edge e with
                updated vioTS and ccScope;
11.       Replace G_s and G_s' with G_ss' in checkCC[i,T] and CC[i,T];
12. for each cc G_s in checkCC[i,T]
13.    Update G_s.ccScope, G_s.vioTS and scope[e] for each e ∈ G_s if changes;
14.    let E_rm := edges e ∈ G_s, which need to be removed;
15.    if E_rm = ∅ then
16.       Move G_s from checkCC[i,T] to maxCC;
17.    else Add each recomputed cc in G_s \ E_rm with ccScope to maxCC;
18. return maxCC, CC[i,T], checkCC[i,T], and the updated array scope.
```

Figure 6: Procedure maxCheck

We next illustrate procedure maxCheck with an example.

Example 8: We consider the same setting as Example 7. Assume that the procedure executes for interval $[3, 7]$. Note that CC$[8, 10]$ = $\{G_{s1}$ with (v_1, v_2) and (v_1, v_5), G_{s2} with $(v_3, v_4)\}$, G_{s1}.ccScope = $[3, 9]$, G_{s2}.ccScope = $[2, 8]$, G_{s2}.vioTS = $\emptyset$, G_{s1}.vioTS = $\{\langle[7, 8], (v_1, v_2)\rangle, \langle[8, 8], (v_1, v_5)\rangle\}$, checkCC$[8, 10] = \emptyset$, R$[3, 7] = \{(v_2, v_3)\}$, maxIntv$[(v_2, v_3), 1] = [1, 10]$ and vioT$[(v_2, v_3), 1] = \{[8, 10]\}$.

It first initializes CC$[7, 10]$ = CC$[8, 10]$, then generates one cc G_{s3} by combining G_{s1} and G_{s2} and edge (v_2, v_3) in R$[3, 7]$, and adds G_{s3} into checkCC$[7, 10]$. G_{s3}.vioTS is updated to $\{\langle[7, 7], (v_1, v_2)\rangle\}$ by removing timestamps $t > i = 7$, G_{s3}.ccScope is updated to $[3, 8]$, and scope$[(v_1, v_5)]$ is updated to $[1, 9]$. As (v_1, v_2) needs to be removed, two ccs G_{s4} with edges (v_2, v_3) and (v_3, v_4), and G_{s5} with edge (v_1, v_5) are generated, and are added into maxCC. □

Complexity. Procedure maxCheck takes $O(T|E_{m,T}|)$ time and $O(\max E_{m,T})$ space to generate all the maximal RTMs with intervals $[m, i]$ ($i \in [m + k - 1, T]$) for each m. Here, $E_{m,T}$ is $\mathsf{S}^*[m, m + k - 1]$, and $\max E_{m,T}$ is the maximum number of $|E_{m,T}|$ for each m.

Procedure expCheck
Input: the edge set $\mathsf{R}[m,i]$, the cc set maxCC, the relaxation bound δ, DEL-Table, and the interval $[m,i]$.
Output: the set $\mathsf{TF}[m,i]$ of maximal and non-expandable RTMs for $[m,i]$.

```
1.  TF[m, i] := ∅;
2.  for each cc G_s in maxCC
3.     let [l, r] := G_s.ccScope;
4.     if there exists edge e in G_s, e ∈ R[m, i] and l = m then
5.        Add G_s to TF[m, i];    //G_s is non-expandable
6.     else Check whether L^n(e) = L^h(e) and the mismatches of e satisfy bound δ
              in any intervals [n, h] containing [m, i] for each edge e ∈ G_s;
7.           Add G_s to TF[m, i] if G_s is non-expandable;
8.  return TF[m, i].
```

Figure 7: Procedure expCheck

3.5 Expandable Property Checking

We next introduce how to check the expandable property of a maximal RTM $G_s(V_s, E_s, m, i)$, once we generate maximal RTMs (*i.e.,* connected components, or simply ccs, in maxCC) for interval $[m,i]$ using procedure maxCheck. By Proposition 5 in Section 3.1, we can check all the intervals within the intersection of the intervals scope$[e]$ for all edges $e \in E_s$, which is dynamically maintained in procedure maxCheck, *i.e.,* G_s.ccScope.

Assume that $G_s.\text{ccScope} = [l,r]$. If there is an interval $[n,h]$ containing $[m,i]$ ($n \geq l$ and $h \leq r$) such that $L^n(e) = L^h(e)$ for each edge $e \in G_s$, and the label mismatches of e satisfy the relaxation bound δ in $[n,h]$, then G_s is expandable. Note that we can check these by verifying whether the conditions $lab_n = lab_h$ and $dif_h - dif_n \leq \delta(h-n+1)$ both hold in DEL-Table for edge e.

Procedure expCheck. We next present the details of procedure expCheck shown in Figure 7. The procedure takes as input the edge set $\mathsf{R}[m,i]$, the cc set maxCC, the relaxation bound δ, DEL-Table, and the interval $[m,i]$, and returns the set $\mathsf{TF}[m,i]$ of maximal and non-expandable RTMs for $[m,i]$. It first initializes the set $\mathsf{TF}[m,i]$ to an empty set (line 1). For each cc G_s in maxCC, it obtains the interval $[l,r]$ from G_s.ccScope (lines 2-3). If G_s has an edge in $\mathsf{R}[m,i]$ and $l = m$, it adds non-expandable G_s to the set $\mathsf{TF}[m,i]$ (lines 4-5) by Proposition 4 (Section 3.1). Otherwise, it checks whether there is an interval $[n,h]$ ($n \geq l$ and $h \leq r$) containing $[m,i]$ such that $L^n(e) = L^h(e)$, and the label mismatches of e satisfy the relaxation bound δ in $[n,h]$ for each edge $e \in G_s$. It adds non-expandable RTMs to $\mathsf{TF}[m,i]$ (lines 6-7), and finally returns $\mathsf{TF}[m,i]$ (line 8).

We next illustrate procedure expCheck with an example.

Example 9: We consider the same setting as Example 7. Assume that the procedure executes for interval $[3,7]$. There are G_{s4} with (v_2,v_3) and (v_3,v_4), and G_{s5} with (v_1,v_5) in maxCC. $G_{s4}.\text{ccScope} = [1,7]$ and $G_{s5}.\text{ccScope} = [1,9]$. As G_{s4} and G_{s5} can appear in $[2,7]$ and $[3,9]$, respectively, they are expandable. ☐

Complexity. Procedure expCheck takes $O(avgI \cdot |E_{m,T}|)$ time and $O(T\max E_{m,T})$ space to generate maximal and non-expandable RTMs with intervals $[m,i]$ ($i \in [m+k-1,T]$) for each m. Here, $avgI$ is the average number of intervals to be checked when determining whether an RTM is expandable or not.

3.6 The Complete Algorithm

Based on the previous three procedures, we finally present our algorithm FRTM, which takes as input a temporal network $G(V,E,1,T,L)$, a frequency threshold k, and relaxation bounds δ and c, and returns a set TF of all the maximal and non-expandable RTMs.

Algorithm FRTM. It first creates DEL-Table, and initializes the arrays maxIntv and vioT to be empty, and the arrays checkT and scope to 0 and *null*, respectively. Then it deals with intervals $[m,i]$ ($i \in [m+k-1,T]$) for each m from 1 to $T-k+1$. For each m, (1) it invokes edgeFilter to obtain the edge sets $\mathsf{R}[m,i]$ for $i \in [m+k-1,T]$ and the updated arrays maxIntv, vioT and checkT. (2) It then initializes the connected component (or simply cc) sets $\mathsf{CC}[i,T]$ and $\mathsf{checkCC}[i,T]$ for $i \in [m+k-1,T+1]$ to be empty. (3) After that, it deals with intervals $[m,i]$ for i from T to $m+k-1$. For each i, (a) it invokes maxCheck to obtain the cc sets maxCC, $\mathsf{CC}[i,T]$, $\mathsf{checkCC}[i,T]$ and the updated array scope for interval $[m,i]$. (b) It then invokes expCheck to obtain the set $\mathsf{TF}[m,i]$ of maximal and non-expandable RTMs with interval $[m,i]$ by ccScope of each cc. Finally, it returns the set TF.

Proposition 6: FRTM *correctly finds all the maximal and non-expandable* RTMs. ☐

We next illustrate algorithm FRTM with an example.

Example 10: We consider the same setting as Example 7. Assume that FRTM executes for intervals $[3,10]$, $[3,9]$, $[3,8]$ and $[3,7]$.

It invokes edgeFilter and returns $\mathsf{R}[3,7] = \{(v_2,v_3)\}$, $\mathsf{R}[3,8] = \{(v_3,v_4)\}$ and $\mathsf{R}[3,9] = \{(v_1,v_2),(v_1,v_5)\}$. (1) For $[3,9]$: G_{s1} with (v_1,v_2) and (v_1,v_5) is generated in maxCheck with ccScope = $[3,9]$, and added to TF in expCheck (non-expandable). (2) For $[3,8]$: G_{s2} with (v_3,v_4) is generated with ccScope = $[2,8]$ (expandable). (3) For $[3,7]$: G_{s3} with (v_1,v_2), (v_1,v_5), (v_2,v_3) and (v_3,v_4) is generated, but then (v_1,v_2) is removed in maxCheck, as G_{s3}.vioTS contains the timestamp 7 for (v_1,v_2) (its label mismatches violate the relaxation bound δ in $[3,7]$). G_{s4} with (v_2,v_3) and (v_3,v_4) (*i.e.,* H_1 in Figure 2(b)), and G_{s5} with (v_1,v_5) are recomputed ccs in G_{s3} (both are expandable). Finally, G_{s1} (*i.e.,* H_2 in Figure 2(b)) is the maximal and non-expandable RTM returned. ☐

Complexity & Correctness. Algorithm FRTM takes $O(avgI \cdot T^2\max E_{m,T} + T^2|E|) = O(avgI \cdot T^2|E|)$ time and takes $O(T|E| + T^2\max E_{m,T})$ space. Here, $\max E_{m,T}$ is the maximum number of $|\mathsf{S}^*[m,m+k-1]|$ for each m. By Proposition 6 and the above complexity analysis, we have finally proved Theorem 1.

3.7 Optimization Strategies

In this section, we introduce two optimization strategies.

Strategy 1: common edge identifying. In algorithm FRTM, procedure maxCheck generates maximal but expandable RTMs for interval $[m,i]$ if the label mismatches of edges satisfy the relaxation bound δ in $[m-1,i]$, *i.e.,* there are common edges in $\mathsf{S}^*[m,i]$ and $\mathsf{S}^*[m-1,i]$ ($i \in [m+k-1,T]$). We propose an optimization strategy to reduce the generation of these expandable RTMs.

Proposition 7: *If each edge e in a connected component (or simply* cc*) of* $G_s(\mathsf{S}^*[m,i])$ *satisfies* $L^{m-1}(e) = L^m(e)$*, any of these edges cannot form a maximal and non-expandable* RTM *with interval* $[m,i]$. ☐

Proposition 7 reveals that we can reduce the generation of expandable RTMs by checking edge labels for each cc in $G_s(\mathsf{S}^*[m,i])$.

We next incorporate this strategy into algorithm FRTM as follows. For each $m \in [2,T-k+1]$, let $\mathsf{R}^+[m]$ be the set of edges $e \in \mathsf{R}[m,i]$ ($i \in [m+k-1,T]$) and $L^{m-1}(e) \neq L^m(e)$. (1) We can obtain the set $\mathsf{R}^+[m]$ directly when obtaining the R edge sets for each edge in the Case 3 of procedure edgeFilter (lines 10-11 in Figure 5). (2) We can initialize a mark (set to '$True$') for each generated cc in procedure maxCheck, and change the mark to '$False$' if it contains an edge $e \in \mathsf{R}^{+}[m]$. For each cc marked as '$True$', we do not add it to the set checkCC in maxCheck (lines 5, 8 & 11 in Figure 6), as its edges cannot generate a non-expandable RTM. (3) After recomputing the new ccs in maxCheck (line 17 in Figure 6), we add only the ccs containing edges $e \in \mathsf{R}^{+}[m]$ into the set maxCC.

Strategy 2: short interval handling. In algorithm FRTM, procedure maxCheck generates maximal RTMs for interval $[m, i]$ ($i \in [m+k-1, T]$) by the maximal property checking of $G_s(\mathsf{S}^*[m, i])$, *i.e.,* removing edges e from G_s such that $L^m(e) \neq L^i(e)$ and the mismatches of e satisfy the relaxation bound δ. However, for certain short intervals, these are unnecessary. We propose an optimization strategy to avoid the maximal property checking for short intervals.

Proposition 8: *For any $k < 1/\delta$, let* $\max L$ *be the maximum integer satisfying $k \leq \max L < 1/\delta$, $\mathsf{S}^*[m, i] = \mathsf{S}[m, i]$ ($m + k - 1 \leq i \leq m + \max L - 1$).* □

Proposition 8 reveals that for all edges e in $G_s(\mathsf{S}^*[m, i])$ ($m + k - 1 \leq i \leq m + \max L - 1$), $L^m(e) = L^i(e)$ holds, and the label mismatches of e satisfy the relaxation bound δ (as defined by the S edge sets in Section 3.1). This means we can generate maximal RTMs for interval $[m, i]$ by $G_s(\mathsf{S}^*[m, i])$ without the maximal property checking. Note that this strategy is valid only for $k < 1/\delta$.

We next incorporate this strategy into algorithm FRTM as follows. (1) For large intervals, *i.e.,* intervals $[m, i]$ for $m + \max L \leq i \leq T$, we generate maximal RTMs along the same lines as FRTM with the setting of $k = \max L + 1$. (2) For short intervals, *i.e.,* intervals $[m, i]$ for $m + k - 1 \leq i < m + \max L$, we can maintain the new cc set $\widetilde{\mathsf{CC}}[i, T]$, having edges in $\bigcup_{i \leq h \leq m + \max L - 1} \mathsf{R}[m, h] \cup \mathsf{S}[m, m + \max L - 1]$, instead of the set $\mathsf{CC}[i, T]$ in maxCheck. For each cc in set $\widetilde{\mathsf{CC}}$, we just need to maintain the interval ccScope along the same lines as FRTM, and store it in the set maxCC directly, *i.e.,* the set checkCC and lines 12-17 in Figure 6 are unnecessary.

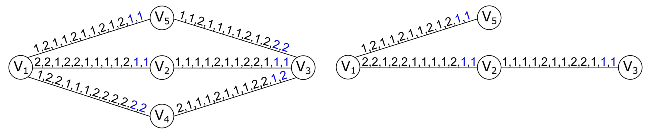
Algorithm FRTM with these two strategies is denoted as FRTM^{+}.

Complexity. FRTM^{+} has the same time and space complexities as FRTM. However, the first strategy reduces the generation of expandable RTMs by identifying common edges, and the second one avoids the maximal property checking when generating maximal RTMs that fall within short intervals. These significantly improve the efficiency, as shown in the experimental study.

4 Incremental Algorithm

In this section, we present an incremental method to efficiently find all the maximal and non-expandable RTMs, given a temporal network $G'(V, E, 1, T + \Delta T)$, a frequency threshold k, relaxation bounds δ and c, a set TF of all the maximal and non-expandable RTMs for $G(V, E, 1, T)$, and two intermediate result sets EIntR and MIntR for G. That is, the temporal network G evolves with ΔT new snapshots. Here, the set EIntR contains edges e that might belong to the RTMs for the intervals $[m, i]$, where $m \in [1, T - k + 1]$ and $i \in [T+1, T+\Delta T]$ (referred to as *affected edges*), as well as the arrays maxIntv, vioT and checkT for e (computed in procedure edgeFilter). The set MIntR contains the RTMs that might be expandable for G' (referred to as *affected* RTMs). The main result is stated below.

Theorem 2: *Given a temporal network $G'(V, E, 1, T + \Delta T, L)$, a frequency threshold k, relaxation bounds δ and c, a set* TF *of maximal and non-expandable* RTMs *for $G(V, E, 1, T, L)$, and two intermediate*



(a) Updated temporal network G' with interval $[1, 12]$ **(b)** RTM H_3 **of G' with interval $[3, 12]$**

Figure 8: Running example for the incremental algorithm

result sets EIntR *and* MIntR *for G, there is an algorithm that finds the set* TF^{+} *of updated maximal and non-expandable* RTMs *in $O(avgI \cdot (T\Delta T|E_{\mathsf{EIntR}}| + (\Delta T)^2|E| + |E_{\mathsf{MIntR}}|))$ time. Here, $|E_{\mathsf{EIntR}}|$ and $|E_{\mathsf{MIntR}}|$ are the numbers of edges in* EIntR *and* MIntR, *respectively.* □

We first analyze unaffected edges and RTMs, as well as affected edges and RTMs, which form the basis of our incremental algorithm.

Fact 1: *Unaffected edges: if the label mismatches of edge e violate the relaxation bound c in interval $[m, T]$, $e \notin \mathsf{R}[m, i]$ for $m \in [1, T - k + 1]$, $i \in [T + 1, T + \Delta T]$,* i.e., *edge e is unaffected.* □

Fact 1 can be proved by the definition of the relaxation bound c.

Proposition 9: *Affected edges: if the label mismatches of edge e satisfy the relaxation bound c in interval $[m, T]$, edge e might belong to sets* $\mathsf{R}[m, i]$ *for $m \in [1, T - k + 1]$, $i \in [T + 1, T + \Delta T]$.* □

Proposition 9 reveals that we can obtain the affected edges easily by checking whether the label mismatches of edges violate the relaxation bound c when computing R edge sets for G (in Section 3.3).

Proposition 10: *Unaffected* RTM*s: if any edge of an* RTM *in set* TF *is unaffected, the* RTM *is unaffected,* i.e., *non-expandable for G'.* □

Proposition 10 reveals that we do not need to check the expandable property of the unaffected RTMs in set TF.

Proposition 11: *Affected* RTM*s: if all the edges of an* RTM *in set* TF *are affected, the* RTM *is affected,* i.e., *might be expandable for G'.* □

Proposition 11 reveals that there might exist expandable RTMs in set TF, which require the expandable property checking.

We next show how to obtain affected edges and affected RTMs in algorithm FRTM. Assume that the algorithm handles intervals $[m, m + k - 1], \cdots, [m, T]$. We denote the sets EIntR and MIntR obtained for these intervals as $\mathsf{EIntR}[m]$ and $\mathsf{MIntR}[m]$, respectively. After obtaining R edge sets by DEL-Table in Cases 1 & 3 of procedure edgeFilter (lines 4 & 10 in Figure 5), we store affected edges e with labels $L^m(e)$ and arrays $\mathsf{maxIntv}[e, L^m(e)]$, $\mathsf{vioT}[e, L^m(e)]$ and $\mathsf{checkT}[e, L^m(e)]$ into $\mathsf{EIntR}[m]$, when edge e with its label $L^m(e)$ is not in EIntR. Moreover, we store each non-expandable affected RTM into $\mathsf{MIntR}[m]$ in procedure expCheck, after verifying that all the edges in the RTM are affected. Note that the above revision does not change the time complexity of FRTM.

Algorithm DFRTM. Based on Propositions 9 & 11, we can design incremental algorithm DFRTM by revising static algorithm FRTM. (1) For intervals $[m, i]$, $m \in [1, T-k+1]$, $i \in [T+1, T+\Delta T]$, DFRTM first initializes arrays maxIntv, vioT and checkT along the same lines as FRTM, and then updates maxIntv, vioT and checkT for each edge $e \in \mathsf{EIntR}[m]$. After that, it invokes procedure edgeFilter, and only the edges in $\bigcup_{n \leq m} \mathsf{EIntR}[n]$ are considered when computing the R edge sets, by Proposition 9. Specifically, for all edges $e \in \mathsf{EIntR}[m]$, it invokes procedure edgeFilter to obtain the R edge set for e by scanning DEL-Table from timestamp $T + 1$, while for all edges $e \in \bigcup_{n < m} \mathsf{EIntR}[n]$, it handles these along the same lines as algorithm FRTM. DFRTM next invokes procedures maxCheck and expCheck along the same lines as algorithm FRTM to compute the set TF^{+}.

Finally, it assigns $\mathsf{TF}^+[m,h] = \mathsf{TF}[m,h]$ for $h \in [m+k-1, T]$ and removes expandable RTMs in $\mathsf{MIntR}[m]$ from TF^+.
(2) For intervals $[m,i], m \in [T-k+2, T+\Delta T-k+1], i \in [m+k-1, T+\Delta T]$, DFRTM handles these along the same lines as FRTM.

We illustrate algorithm DFRTM with an example.

Example 11: Consider the updated temporal network G' in Figure 8a with $T = 10$ and $\Delta T = 2$, $k = 5$, $\delta = 30\%$ and $c = 2$. Assume that DFRTM executes for $m = 3$. $\mathsf{EIntR}[1] = \{(v_1, v_5), (v_2, v_3), (v_3, v_5)\}$, $\mathsf{EIntR}[2] = \{(v_3, v_4)\}$, $\mathsf{EIntR}[3] = \{(v_1, v_2)\}$, and $\mathsf{MIntR}[3] = \{$RTM G_{s1} with (v_1, v_2) and (v_1, v_5) in Example 10$\}$.

It invokes edgeFilter, considering edges in $\bigcup_{1\leq n\leq 3} \mathsf{EIntR}[n]$, and identifies $\mathsf{R}[3, 11] = \{(v_3, v_4)\}$ and $\mathsf{R}[3, 12] = \{(v_2, v_3), (v_1, v_5), (v_1, v_2)\}$. It next invokes maxCheck and expCheck to compute $\mathsf{TF}^+[3, 11]$ and $\mathsf{TF}^+[3, 12]$. As Figure 8(b) shows, H_3 is the RTM in $\mathsf{TF}^+[3, 12]$ with (v_1, v_5), (v_1, v_2) and (v_2, v_3). It assigns $\mathsf{TF}^+[3, i] = \mathsf{TF}[3, i]$ for $i \in [7, 10]$ (only G_{s1}), removes expandable G_{s1} (in $\mathsf{MIntR}[3]$) from $\mathsf{TF}^+[3, 9]$, and finally returns H_3. □

Complexity & Correctness. Let $|E_{\mathsf{EIntR}}|$ and $|E_{\mathsf{MIntR}}|$ are the numbers of edges in sets EIntR and MIntR, respectively. Incremental algorithm DFRTM takes $O(avgI\cdot(T\Delta T|E_{\mathsf{EIntR}}|+(\Delta T)^2|E|+|E_{\mathsf{MIntR}}|))$ time, while static algorithm FRTM takes $O(avgI \cdot (T + \Delta T)^2|E|)$ time. As $|E_{\mathsf{EIntR}}| < |E|$ and $|E_{\mathsf{MIntR}}| < T^2|E|$, DFRTM has better time complexity than FRTM.

The extra cost of DFRTM involves the sets EIntR and MIntR, which are $O(|E_{\mathsf{EIntR}}|) = O(|L||E|)$ and $O(|E_{\mathsf{MIntR}}|) << O((T+\Delta T)^2 \max E_{m,(T+\Delta T)})$, respectively, while FRTM takes $O((T + \Delta T)|E| + (T+\Delta T)^2 \max E_{m,(T+\Delta T)})$ for $G(V, E, 1, T+\Delta T)$. Here, $\max E_{m,(T+\Delta T)}$ is the maximum number of $|\bigcup_{m+k-1\leq h\leq T+\Delta T} \mathsf{R}[m, h]|$ for each m. Hence, DFRTM has the same space complexity as FRTM.

The correctness of DFRTM follows from Propositions 9, 11 & 6.

Remarks. We have proved Theorem 2 by the above complexity and correctness analyses. The incremental algorithm has theoretically better time complexity and the same space complexity with a reasonable extra space cost, compared with its static counterpart. Moreover, our optimization strategies also apply to the incremental algorithm. We denote the optimization algorithm as DFRTM^+.

5 Experimental Study

Using both real-life and synthetic datasets, we conducted extensive experiments of our static algorithms and incremental algorithms.

5.1 Experimental Settings

We first illustrate our experimental settings.

Datasets. We obtain three datasets, as shown in Table 1.
(1) BJData [41] is a real-life dataset that records three types of road traffic status (*i.e.,* congested, slow and fast) in Beijing. We extracted the day-level data (November 4th, 2010) with 5 minutes a snapshot.
(2) EURData [20] is a real-life dataset for a renewable European electric power system. Each edge represents a transmission line, and each node represents a merging point of transmission lines with a dynamic energy demand signal (1 hour a snapshot). We transformed the network by switching edges to nodes and nodes to edges, and used 10 labels to discretize energy demand signals (1-10: from lowest to highest, such as '1' means '0 ∼ 200 MWh').
(3) SYNData [6] is produced by the synthetic data generator. All edge labels are −1 at first, and then some of them are activated

Table 1: Statistics of datasets

Datasets	Nodes	Edges	Snapshots	Total Edges
BJData (457 MB)	69, 416	88, 396	288	25, 458, 048
EURData (3.15 GB)	2, 707	7, 334	26, 304	192, 913, 536
SYNData (30.6 GB)	400, 000	800, 000	2, 000	1, 600, 000, 000

(transformed to 1) at random. Each activated edge activates its neighboring edges and its copy in the next snapshot with decayed probabilities np_r and tp_r, respectively. The process stops when the percentage of activated edges reaches the activation density ad_r. We fixed tp_r, ad_r and np_r to 0.9, 0.3 and 0.3, respectively.

Note that we treat edges with different labels at different timestamps as different edges, which is common in temporal networks [12].

Algorithms. We tested six algorithms: our baseline static algorithm FRTM and incremental algorithm DFRTM, algorithms FRTM and DFRTM equipped with the optimization strategy of common edge identifying (denoted as FRTM(OPT1) and DFRTM(OPT1), respectively), as well as optimization algorithms FRTM^+ and DFRTM^+.

Implementation. We implemented all algorithms with C++, and conducted experiments on a PC with 2 Intel Xeon E5-2640 2.6GHz CPUs, 64GB RAM, and a Windows 7 operating system. All tests were repeated over 3 times and the average is reported.

5.2 Experimental Results

We next present our findings.

Exp-1: Tests of static algorithms. In the first set of tests, we test the running time and memory cost of static algorithms FRTM, FRTM(OPT1) and FRTM^+ *w.r.t.* the relaxation bound δ, the relaxation bound c, and the frequency threshold k.

Exp-1.1. To evaluate the impacts of the bound δ, we varied δ from 1% to 5%, fixed $c = 3$ and $k = 10$, and used the entire networks for all the datasets. The results are reported in Figures 9(a)-9(c).

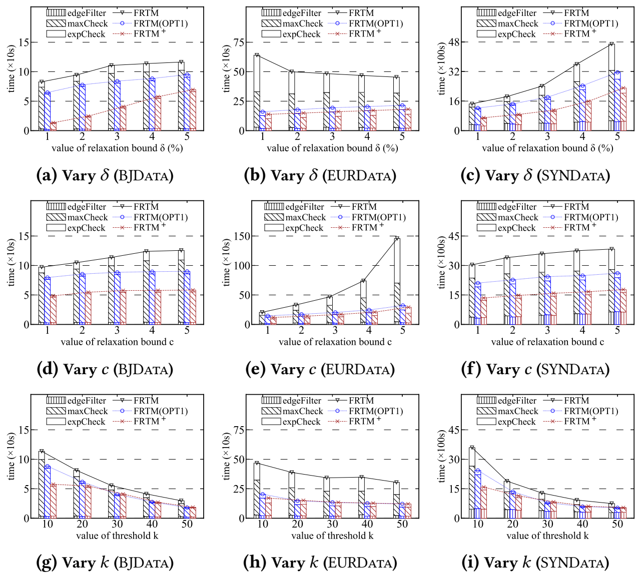
When varying the bound δ, the running time of most algorithms increases with the increment of δ, except for FRTM on EURData. This is because the number of intervals checked in expCheck decreases for larger δ on EURData, which has more impacts on the running time than the increasing edge number in FRTM, compared with other algorithms. Moreover, FRTM^+ runs in 699.89 seconds on SYNData with $k = 10$, $c = 3$, $\delta = 1\%$, and is on average (3.27, 3.24, 2.13) and (2.59, 1.19, 1.59) times faster than FRTM and FRTM(OPT1) on (BJData, EURData, SYNData), respectively.

When varying the bound δ, the running time of most procedures increases with the increment of δ, except for the procedure expCheck of FRTM on EURData for the same reason as above. Procedure maxCheck takes the most time, except for FRTM on EURData with $\delta = 0.01$, and is on average (85%, 57%, 62%) for FRTM, (92%, 67%, 70%) for FRTM(OPT1) and (84%, 57%, 59%) for FRTM^+ on (BJData, EURData, SYNData), respectively.

Exp-1.2. To evaluate the impacts of the bound c, we varied c from 1 to 5, fixed $\delta = 4\%$ and $k = 10$, and used the entire networks for all the datasets. The results are reported in Figures 9(d)-9(f).

When varying the bound c, the running time of all three algorithms increases with the increment of c, as there are more RTMs with more edges meeting bound c for larger c. Moreover, FRTM^+ is on average 3.07, 2.25) and (1.57, 1.18, 1.52) times faster than FRTM and FRTM(OPT1) on (BJData, EURData, SYNData), respectively.

When varying the bound c, the running time of all three procedures increases with the increment of c. Procedure maxCheck



(a) Vary δ (BJData) **(b) Vary δ (EURData)** **(c) Vary δ (SYNData)**

(d) Vary c (BJData) **(e) Vary c (EURData)** **(f) Vary c (SYNData)**

(g) Vary k (BJData) **(h) Vary k (EURData)** **(i) Vary k (SYNData)**

Figure 9: Algorithms FRTM, FRTM(OPT1) and FRTM$^+$

Table 2: Memory cost of static algorithms

Datasets	FRTM	FRTM(OPT1)	FRTM$^+$
BJData (457 MB)	1.60 GB	1.61 GB	1.57 GB
EURData (3.15 GB)	13.92 GB	13.92 GB	13.71 GB
SYNData (30.6 GB)	38.01 GB	38.07 GB	37.57 GB

takes the most time, and is on average (85%, 57%, 60%) for FRTM, (92%, 67%, 71%) for FRTM(OPT1) and (88%, 56%, 59%) for FRTM$^+$ on (BJData, EURData, SYNData), respectively.

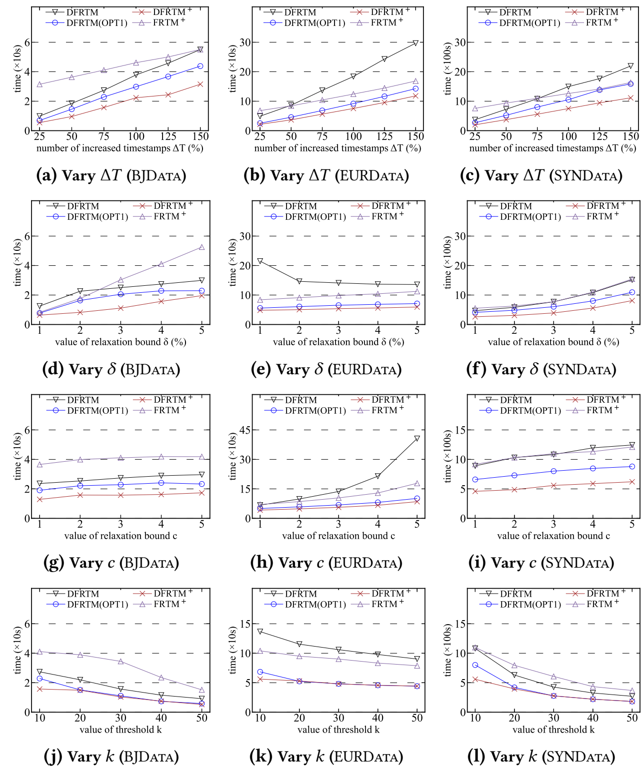
Exp-1.3. To evaluate the impacts of the threshold k, we varied k from 10 to 50, fixed $\delta = 4\%$ and $c = 3$, and used the entire networks for all the datasets. The results are reported in Figures 9(g)-9(i).

When varying the threshold k, the running time of all three algorithms decreases with the increment of k, as there are fewer edges meeting frequency thresholds in large intervals and fewer RTMs for larger k. Moreover, FRTM$^+$ is on average (1.59, 2.62, 1.67) and (1.13, 1.04, 1.13) times faster than FRTM and FRTM(OPT1) on (BJData, EURData, SYNData), respectively. Note that the strategy of short interval handling is invalid for $k \geq 25$ in FRTM$^+$.

When varying the threshold k, the running time of all three procedures decreases with the increment of k. Procedure maxCheck takes the most time, except for large k on SYNData (FRTM for $k = 50$, and FRTM(OPT1) and FRTM$^+$ for $k \geq 40$), in which case procedure edgeFilter takes the most time. This is because the number of edges in SYNData which meet the bounds δ and c decreases in large intervals. Procedure maxCheck is on average (80%, 58%, 45%) for FRTM, (88%, 61%, 51%) for FRTM(OPT1) and (87%, 58%, 49%) for FRTM$^+$ on (BJData, EURData, SYNData), respectively.

Exp-1.4. To evaluate the space cost, we tested the memory cost of static algorithms in practice, under the same settings as Exp-1.3, while fixed $k = 10$. The results are reported in Table 2.

Algorithm FRTM(OPT1) uses (0.77%, 0.03%, 0.15%) more memory than FRTM on all the datasets for more space of the set R$^+$ and the marks for the connected components, while FRTM$^+$ uses (1.66%, 1.46%, 1.16%) less memory than FRTM on (BJData, EURData, SYNData), respectively, for less space of connected components. These



(a) Vary ΔT (BJData) **(b) Vary ΔT (EURData)** **(c) Vary ΔT (SYNData)**

(d) Vary δ (BJData) **(e) Vary δ (EURData)** **(f) Vary δ (SYNData)**

(g) Vary c (BJData) **(h) Vary c (EURData)** **(i) Vary c (SYNData)**

(j) Vary k (BJData) **(k) Vary k (EURData)** **(l) Vary k (SYNData)**

Figure 10: Algorithms DFRTM, DFRTM(OPT1) and DFRTM$^+$

are consistent with the space complexity analysis. Moreover, the memory cost of three algorithms is rational, as SYNData is large.

Exp-2: Tests of incremental algorithms. In the second set of tests, we test the running time and memory cost of incremental algorithms DFRTM, DFRTM(OPT1) and DFRTM$^+$ *w.r.t.* the number ΔT of increased timestamps, the relaxation bounds δ, the relaxation bound c, and the frequency threshold k, compared with the best performing static algorithm FRTM$^+$.

Exp-2.1. To evaluate the impacts of the number ΔT of increased timestamps, we varied ΔT from 25% to 150%, while fixed $\delta = 4\%$, $c = 3$, $k = 10$, and the original timestamps $T = (112, 10400, 800)$ for (BJData, EURData, SYNData), respectively. The results are reported in Figures 10(a)-10(c).

When varying the number of ΔT, the running time of all four algorithms increases with the increment of ΔT. Moreover, incremental algorithm DFRTM$^+$ is always the best, and is on average (3.00, 1.99, 2.15), (1.80, 2.43, 1.93) and (1.41, 1.21, 1.41) times faster than its static counterpart FRTM$^+$, algorithms DFRTM and DFRTM(OPT1) on (BJData, EURData, SYNData), respectively. These are consistent with the time complexity analysis.

Exp-2.2. To evaluate the impacts of the bound δ, we varied δ from 1% to 5%, while fixed the same settings of T, c and k as in Exp-2.1 and $\Delta T = 75\%$. The results are reported in Figures 10(d)-10(f).

When varying the bound δ, the running time of most algorithms increases with the increment of δ, except for DFRTM on EURData, for the same reason as in Exp-1.1. Moreover, incremental algorithm DFRTM$^+$ is always the best, and is on average (2.29, 1.83, 2.00), (2.04, 2.94, 1.89) and (1.53, 1.20, 1.49) times faster than its static

Table 3: Memory cost of incremental algorithms and the best static algorithm (ΔT = 75%)

Datasets	DFRTM	DFRTM(OPT1)	DFRTM+	FRTM+
BJData (457 MB)	1.21 GB	1.22 GB	1.21 GB	1.21 GB
EURData (3.15 GB)	8.06 GB	8.07 GB	7.93 GB	8.20 GB
SYNData (30.6 GB)	30.41 GB	30.48 GB	30.17 GB	29.99 GB

counterpart FRTM+, algorithms DFRTM and DFRTM(OPT1) on (BJData, EURData, SYNData), respectively.

Exp-2.3. To evaluate the impacts of the bound c, we varied c from 1 to 5, while fixed the same settings of T, δ and k as in Exp-2.1 and $\Delta T = 75\%$. The results are reported in Figures 10(d)-10(f).

When varying the bound c, the running time of all four algorithms increases with the increment of c, for the same reason as in Exp-1.2. Moreover, algorithm DFRTM+ is always the best, and is on average (2.60, 1.87, 1.99), (1.74, 2.81, 2.01) and (1.43, 1.21, 1.44) times faster than its static counterpart FRTM+, algorithms DFRTM and DFRTM(OPT1) on (BJData, EURData, SYNData), respectively.

Exp-2.4. To evaluate the impacts of the threshold k, we varied k from 10 to 50, while fixed the same settings of T, δ and c as in Exp-2.1 and $\Delta T = 75\%$. The results are reported in Figures 10(j)-10(l).

When varying the threshold k, the running time of all four algorithms decreases with the increment of k, for the same reason as in Exp-1.3. Moreover, incremental algorithm DFRTM+ is the best for $k \le 20$, and similar to the algorithm DFRTM(OPT1) for $k \ge 30$, as the strategy of short interval handling is invalid for $k \ge 30$. It is on average (2.92, 1.83, 2.03), (1.61, 2.20, 1.61) and (1.13, 1.04, 1.10) times faster than its static counterpart FRTM+, algorithms DFRTM and DFRTM(OPT1) on (BJData, EURData, SYNData), respectively.

Exp-2.5. To evaluate the space cost, we tested the memory cost of algorithms in practice, under the same settings of T, δ, c and k as in Exp-2.1, while fixed $\Delta T = 75\%$. The results are reported in Table 3.

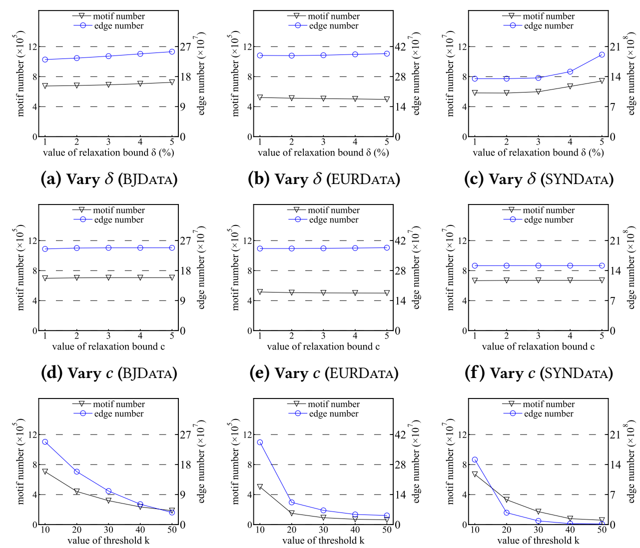
Algorithm DFRTM+ uses (0.22%, 1.66%, 0.81%) and (1.38%, 1.75%, 1.03%) less memory than DFRTM and DFRTM(OPT1) on (BJData, EURData, SYNData), respectively, for the same reason as in Exp-1.6. It uses (0.12%, 0.60%) more memory than FRTM+ on (BJData, SYNData), respectively, while uses 3.30% less memory on EURData. The reason is that, the incremental algorithm only considers intervals $[m, i]$ for $m \in [1, T-k+1]$ and $i \in [T+1, T+\Delta T]$, and the R edge sets use more memory than the sets EIntR and MIntR on EURData, but use less memory on BJData and SYNData.

Exp-3: Tests of parameter sensitivity. In the third set of tests, we test the number of RTMs generated by the algorithms, and the total number of edges in all RTMs, *w.r.t.* the relaxation bound δ, the relaxation bound c, and the frequency threshold k.

Exp-3.1. To evaluate the impacts of the bound δ, we varied δ from 1% to 5%, while fixed the same settings of c and k as in Exp-1.1. The results are reported in Figures 11(a)-11(c).

When varying the bound δ from 1% to 5%, the number of RTMs increases by (7.2%, 27.3%) on (BJData, SYNData), respectively, while it decreases by 4.8% on EURData. The reason is that there are more connected edges in the RTMs when δ is large, which reduces the number of RTMs generated on EURData. Moreover, the total number of edges in all RTMs increases by (10.2%, 2.2%, 41.5%) on (BJData, EURData, SYNData), respectively.

Exp-3.2. To evaluate the impacts of the bound c, we varied c from 1 to 5, while fixed the same settings of δ and k as in Exp-1.2. The results are reported in Figures 11(d)-11(f).



(a) Vary δ (BJData) (b) Vary δ (EURData) (c) Vary δ (SYNData)
(d) Vary c (BJData) (e) Vary c (EURData) (f) Vary c (SYNData)
(g) Vary k (BJData) (h) Vary k (EURData) (i) Vary k (SYNData)

Figure 11: Tests of parameter sensitivity

When varying the bound c from 1 to 5, the number of RTMs increases by (1.1%, 0.6%) on (BJData, SYNData), respectively, while it decreases by 2.8% on EURData, for the same reason as in Exp-4.1. Moreover, the number of edges in all RTMs increases by (1.4%, 1.0%, 0.1%) on (BJData, EURData, SYNData), respectively.

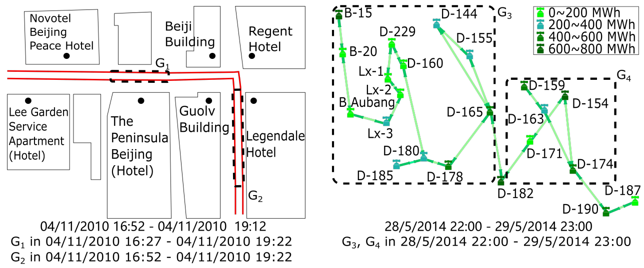
Exp-3.3. To evaluate the impacts of the threshold k, we varied k from 10 to 50, while fixed the same settings of δ and c as in Exp-1.3. The results are reported in Figures 11(g)-11(i).

When varying the threshold k from 10 to 50, the number of RTMs decreases sharply, as fewer edges meet frequency thresholds k. It decreases by (73.6%, 87.2%, 91.2%) on (BJData, EURData, SYNData), respectively. Moreover, the total number of edges in all RTMs decreases by (85.4%, 88.9%, 99.0%) on (BJData, EURData, SYNData), respectively. Hence, our RTMs are sensitive to k.

Exp-4. Case studies. To justify the usefulness of our RTMs, we conducted case studies on BJData and EURData. The following shows two cases from FRTM with $\delta = 4\%$, $c = 3$ and $k = 25$.

Case 1. For BJData, we visualized our RTMs by the traffic data management system from [18] and found corresponding roads in Google Maps. We use the red color to represent roads with 'congested' traffic status labels in the RTMs. We used our RTMs to discover long-time congestion patterns, as shown in Figure 12(a). (1) The RTM corresponds to the traffic status at the *Jinyu Hu Tong* during [16:52, 19:12]. It indicates the evening peak congestion around several hotels, which are located close to the popular tourist destination *Wangfujing Street*. (2) We show the result with exact label matches (*i.e.,* $\delta = 0$ or $c = 0$), including two separate RTMs G_1 and G_2 that both correspond to a single road segment (circled by dashed lines). This is because during the period, all other adjacent road segments not in the RTMs are labeled as 'slow' or 'fast' at certain timestamps. These roads should be merged into a large area, otherwise we cannot have a comprehensive analysis of the jam.

Case 2. For EURData, we transformed the obtained RTMs to their original subgraphs, hence dynamic properties in this case are on nodes, instead of edges. We use (dark green, green, sea green, lime green) colors to represent energy demands (600 ∼ 800, 400 ∼ 600, 200



(a) An RTM on BJData **(b) An RTM on EURData**

Figure 12: Case studies on real-life datasets

$\sim 400, 0 \sim 200$) MWh, respectively. We use our RTMs to discover hybrid energy consumption patterns, as shown in Figure 12(b). (1) The RTM corresponds to transmission lines and their merging points across Belgium, Luxembourg and Germany (beginning with 'B', 'Lx' and 'D', respectively) during [28/5/2014 22:00, 29/5/2014 23:00]. It reveals the energy demands in the areas around Luxembourg and Germany differ greatly. (2) We show the result with exact label matches (*i.e.,* $\delta = 0$ or $c = 0$), including two separate RTMs G_3 and G_4 (circled by dashed lines). This is because during the period, the labels of the merging points '$D-182$', '$D-190$' and '$D-187$' change at certain timestamps. G_3 and G_4 with distinct energy demands should be merged into a large area, otherwise we cannot maintain good load balance in energy transmission of G_3 and G_4.

To the best of our knowledge, existing temporal motifs are unsuitable for finding these patterns. (1) [3, 8, 10, 16, 19, 22–27, 36, 46, 47, 63] impose restrictive constraints on temporal motifs, among which [3] imposes a strict exact label constraint that is the special case of our RTMs with $\delta = 0$ or $c = 0$. Hence, further comparisons with these methods are unnecessary. (2) [31, 55] relax the topology constraints of temporal motifs, but do not allow label mismatches.

Summary. We have the following findings. (1) Our static algorithm FRTM$^+$ runs in 699.89 seconds on large temporal networks from the dataset SYNData with 400 thousand nodes and 1.6 billion edges and $k = 10$, $c = 3$, $\delta = 1\%$. (2) Our incremental algorithm DFRTM$^+$ is faster than its static counterpart FRTM$^+$, and is on average (2.72, 1.88, 2.05) times faster on datasets (BJData, EURData, SYNData), respectively. (3) Our optimization strategies are effective, as FRTM$^+$ and DFRTM$^+$ are on average (2.09, 2.76, 2.03) and (1.80, 2.59, 1.86) times faster than baselines FRTM and DFRTM on datasets (BJData, EURData, SYNData), respectively. (4) Our incremental algorithm DFRTM$^+$ uses at most 0.60% more memory than its static counterpart with $\Delta T = 75\%$. (5) Our RTMs are insensitive to the relaxation bounds $\delta \leq 5\%$ and $c \leq 5$, but sensitive to the frequency threshold k. (6) Our RTMs are effective for practical applications.

6 Related Work

Network motifs in static networks. Network motifs refer to the recurrent and statistically significant subgraphs or patterns within a graph [42, 54], which have widespread applications in both industry and academia [5, 7, 11, 13, 30, 32, 38, 42, 45, 54, 56, 60] (see, *e.g.,* [50, 61] for surveys). There are studies starting to relax the topology constraints in network motifs, such as allowing for several connected components [5], and inexact subgraph isomorphism matching [11, 13, 45]. These focus on network motifs in static networks, while we focus on network motifs in temporal networks.

Network motifs in temporal networks. The studies on network motifs in temporal networks typically redefine network motifs to meet various application demands (see, *e.g.,* [35] for surveys). Most motifs are defined as isomorphic subgraphs, where the edges are attached with timestamps satisfying partial time orders [16, 24, 25, 63] or total time orders [19, 23, 46], as well as local time windows [16, 19, 24, 25, 36, 63] or global time windows [23, 36, 46]. There are also motifs defined as induced subgraphs, where node weights evolving in a consistent trend [10, 22], and edge labels and directions keeping unchanged in a period [3]. To speed up, there are methods exploring sampling [34, 51, 52, 57], parallel computing [15], and for motifs with specific topologies [8, 26, 27, 47]. There are also studies relaxing the topology constraints, *e.g.,* [55] proposes temporal motifs based on dual simulation [39] instead of subgraph isomorphism, while [31] proposes temporal motifs with persistent k-core structures instead of fixed clique structures in a period.

However, existing studies on temporal motif discovery are mostly computationally intractable, and pay little attention to data errors. Instead, our temporal motifs can be computed in low polynomial time, and allow label mismatches to tolerate data errors. Further, incremental methods are developed to handle with the dynamic nature of temporal networks, which has not been considered before.

Patterns with constraint relaxations. There are several studies on patterns with constraint relaxations. [33, 37, 49, 59] relax the support constraint in frequent itemset mining, allowing for missed transactions or items. [4, 14, 21, 62] relax the subgraph isomorphism constraint in frequent subgraph mining, allowing for label or topology mismatches. [29] relax the label constraint in graph simulation with taxonomy, allowing for more label matching relations. Different from these studies, we focus on tolerating data errors for temporal motif discovery.

Similar but different concepts. There are also studies on graphs that bear similarities but are different from motifs, such as lasting dense subgraphs [53], dense temporal subgraphs [6, 40, 41], frequent graph patterns [1] and frequent patterns[1].

Our temporal networks essentially follow from [40, 41], but use edge labels instead of edge weights, to represent the dynamics of temporal networks. Further, our maximal and non-expandable properties are similar to maximal or closed frequent patterns [1, 43], which both serve to reduce redundant patterns.

7 Conclusions

We have proposed a proper notion of relaxed temporal motifs, *i.e.,* RTMs, by allowing limited label mismatches to tolerate data errors. We have developed a static algorithm FRTM$^+$, equipped with two optimization strategies, to efficiently find all the maximal and non-expandable RTMs. We have also developed an incremental algorithm DFRTM$^+$ to handle the continuous updating scenario in temporal networks. Finally, we have experimentally verified that our static algorithm FRTM$^+$ runs fast on large temporal networks, and that our incremental algorithm DFRTM$^+$ significantly outperforms its static counterpart. With a case study on real-life datasets, we have further verified the effectiveness of our RTMs.

A couple of interesting topics are necessary for a further study, and we are planning to investigate the periodicity and distributed settings of temporal motifs.

References

[1] Charu C. Aggarwal and Jiawei Han. 2014. *Frequent Pattern Mining*. Springer, Cham.

[2] Charu C Aggarwal and Haixun Wang. 2010. *Managing and mining graph data*. Springer.

[3] Rezwan Ahmed and George Karypis. 2012. Algorithms for mining the evolution of conserved relational states in dynamic networks. *KAIS* 33, 3 (2012), 603–630.

[4] Pranay Anchuri, Mohammed J. Zaki, Omer Barkol, Shahar Golan, and Moshe Shamy. 2013. Approximate graph mining with label costs. In *KDD*.

[5] Nadja Betzler, René van Bevern, Michael R. Fellows, Christian Komusiewicz, and Rolf Niedermeier. 2011. Parameterized Algorithmics for Finding Connected Motifs in Biological Networks. *IEEE ACM Trans. Comput. Biol. Bioinform.* 8, 5 (2011), 1296–1308.

[6] Petko Bogdanov, Misael Mongiovì, and Ambuj K Singh. 2011. Mining heavy subgraphs in time-evolving networks. In *ICDM*.

[7] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proc. VLDB Endow.* 12, 11 (2019), 1651–1663.

[8] Xinwei Cai, Xiangyu Ke, Kai Wang, Lu Chen, Tianming Zhang, Qing Liu, and Yunjun Gao. 2024. Efficient Temporal Butterfly Counting and Enumeration on Temporal Bipartite Graphs. *Proc. VLDB Endow.* 17, 4 (2024), 657–670.

[9] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*.

[10] Elise Desmier, Marc Plantevit, Céline Robardet, and Jean-François Boulicaut. 2013. Trend mining in dynamic attributed graphs. In *ECML/PKDD*.

[11] Riccardo Dondi, Guillaume Fertin, and Stéphane Vialette. 2013. Finding approximate and constrained motifs in graphs. *Theor. Comput. Sci.* 483 (2013), 10–21.

[12] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *Proc. VLDB Endow.* 15, 9 (2022), 1861–1874.

[13] Michael R. Fellows, Guillaume Fertin, Danny Hermelin, and Stéphane Vialette. 2011. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.* 77, 4 (2011), 799–811.

[14] Marisol Flores-Garrido, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez Trinidad. 2015. AGraP: an algorithm for mining frequent patterns in a single graph using inexact matching. *KAIS* 44, 2 (2015), 385–406.

[15] Zhongqiang Gao, Chuanqi Cheng, Yanwei Yu, Lei Cao, Chao Huang, and Junyu Dong. 2022. Scalable Motif Counting for Large-scale Temporal Graphs. In *ICDE*.

[16] Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. 2015. Commit: A scalable approach to mining communication motifs from dynamic networks. In *SIGMOD*.

[17] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

[18] Haixing Huang, Jinghe Song, Xuelian Lin, Shuai Ma, and Jinpeng Huai. 2016. TGraph: A Temporal Graph Data Management System. In *CIKM*.

[19] Yuriy Hulovatyy, Huili Chen, and Tijana Milenković. 2015. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinform.* 31, 12 (2015), i171–i180.

[20] Tue V Jensen and Pierre Pinson. 2017. RE-Europe, a large-scale dataset for modeling a highly renewable European electricity system. *Scientific data* 4 (2017), 170175:1–170175:18.

[21] Yi Jia, Jintao Zhang, and Jun Huan. 2011. An efficient graph-mining method for complicated and noisy data with real-world applications. *KAIS* 28, 2 (2011), 423–447.

[22] Ruoming Jin, Scott McCallen, and Eivind Almaas. 2007. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *ICDM*.

[23] Chrysanthi Kosyfaki, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2019. Flow Motifs in Interaction Networks. In *EDBT*.

[24] 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), P11005:1–P11005:18.

[25] Lauri Kovanen, Kimmo Kaski, János Kertész, and Jari Saramäki. 2013. Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences. *PNAS* 110, 45 (2013), 18070–18075.

[26] Rohit Kumar and Toon Calders. 2018. 2scent: An efficient algorithm to enumerate all simple temporal cycles. *Proc. VLDB Endow.* 11, 11 (2018), 1441–1453.

[27] Geon Lee and Kijung Shin. 2023. Temporal hypergraph motifs. *KAIS* 65, 4 (2023), 1549–1586.

[28] Changle Li, Wenwei Yue, Guoqiang Mao, and Zhigang Xu. 2020. Congestion propagation based bottleneck identification in urban road networks. *IEEE Trans. Veh. Technol.* 69, 5 (2020), 4827–4841.

[29] Jia Li, Yang Cao, and Shuai Ma. 2017. Relaxing Graph Pattern Matching With Explanations. In *CIKM*.

[30] Pan Li, Hoang Dau, Gregory Puleo, and Olgica Milenkovic. 2017. Motif clustering and overlapping clustering for social network analysis. In *INFOCOM*.

[31] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent Community Search in Temporal Networks. In *ICDE*.

[32] Xiaodong Li, Reynold Cheng, Kevin Chen-Chuan Chang, Caihua Shan, Chenhao Ma, and Hongtai Cao. 2021. On analyzing graphs with motif-paths. *Proc. VLDB Endow.* 14, 6 (2021), 1111–1123.

[33] Jinze Liu, Susan Paulsen, Xing Sun, Wei Wang, Andrew B. Nobel, and Jan F. Prins. 2006. Mining Approximate Frequent Itemsets In the Presence of Noise: Algorithm and Analysis. In *SDM*.

[34] Paul Liu, Austin R Benson, and Moses Charikar. 2019. Sampling methods for counting temporal motifs. In *WSDM*.

[35] Penghang Liu, Valerio Guarrasi, and A Erdem Sariyuce. 2023. Temporal network motifs: Models, limitations, evaluation. *TKDE* 35, 1 (2023), 945–957.

[36] Penghang Liu, Naoki Masuda, Tomomi Kito, and Ahmet Erdem Sarıyüce. 2022. Temporal motifs in patent opposition and collaboration networks. *Scientific reports* 12, 1 (2022), 1917:1–1917:11.

[37] Shengxin Liu and Chung Keung Poon. 2018. On mining approximate and exact fault-tolerant frequent itemsets. *KAIS* 55 (2018), 361–391.

[38] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. 2019. Linc: a motif counting algorithm for uncertain graphs. *Proc. VLDB Endow.* 13, 2 (2019), 155–168.

[39] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. 2014. Strong simulation: Capturing topology in graph pattern matching. *TODS* 39, 1 (2014), 4:1–4:46.

[40] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. 2017. Fast Computation of Dense Temporal Subgraphs. In *ICDE*.

[41] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. 2020. An efficient approach to finding dense temporal subgraphs. *TKDE* 32, 4 (2020), 645–658.

[42] 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.

[43] Mohammad Hossein Namaki, Yinghui Wu, Qi Song, Peng Lin, and Tingjian Ge. 2017. Discovering graph temporal association rules. In *CIKM*.

[44] Hoang Nguyen, Wei Liu, and Fang Chen. 2016. Discovering congestion propagation patterns in spatio-temporal traffic data. *IEEE Trans. Big Data* 3, 2 (2016), 169–180.

[45] Carlos G. Oliver, Vincent Mallet, Pericles Philippopoulos, William L. Hamilton, and Jérôme Waldispühl. 2022. Vernal: a tool for mining fuzzy network motifs in RNA. *Bioinform.* 38, 4 (2022), 970–976.

[46] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *WSDM*.

[47] Noujan Pashanasangi and C Seshadhri. 2021. Faster and generalized temporal triangle counting, via degeneracy ordering. In *KDD*.

[48] Stefan Pfenninger and Iain Staffell. 2016. Long-term patterns of European PV output using 30 years of validated hourly reanalysis and satellite data. *Energy* 114 (2016), 1251–1265.

[49] Ardian Kristanto Poernomo and Vivekanand Gopalkrishnan. 2009. Towards efficient mining of proportional fault-tolerant frequent itemsets. In *KDD*.

[50] 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 Comput. Surv.* 54, 2 (2021), 1–36.

[51] Ilie Sarpe and Fabio Vandin. 2021. odeN: Simultaneous Approximation of Multiple Motif Counts in Large Temporal Networks. In *CIKM*.

[52] Ilie Sarpe and Fabio Vandin. 2021. PRESTO: Simple and Scalable Sampling Techniques for the Rigorous Approximation of Temporal Motif Counts. In *SDM*.

[53] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2019. Finding lasting dense subgraphs. *DMKD* 33, 5 (2019), 1417–1445.

[54] Shai S. Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. 2002. Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetics* 31 (2002), 64–68.

[55] Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. 2014. Event pattern matching over graph streams. *Proc. VLDB Endow.* 8, 4 (2014), 413–424.

[56] Ngoc Hieu Tran, Kwok Pui Choi, and Louxin Zhang. 2013. Counting motifs in the human interactome. *Nature communications* 4, 1 (2013), 2241:1–2241:8.

[57] Jingjing Wang, Yanhao Wang, Wenjun Jiang, Yuchen Li, and Kian-Lee Tan. 2020. Efficient sampling algorithms for approximate temporal motif counting. In *CIKM*.

[58] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path Problems in Temporal Graphs. *Proc. VLDB Endow.* 7, 9 (2014), 721–732.

[59] Cheng Yang, Usama M. Fayyad, and Paul S. Bradley. 2001. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD*.

[60] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *KDD*.

[61] Shuo Yu, Yufan Feng, Da Zhang, Hayat Dino Bedru, Bo Xu, and Feng Xia. 2020. Motif discovery in networks: A survey. *Comput. Sci. Rev.* 37 (2020), 100267.

[62] Shijie Zhang, Jiong Yang, and VenuMadhav Cheedella. 2007. Monkey: Approximate Graph Mining Based on Spanning Trees. In *ICDE*.

[63] Qiankun Zhao, Yuan Tian, Qi He, Nuria Oliver, Ruoming Jin, and Wang-Chien Lee. 2010. Communication motifs: a tool to characterize social communications. In *CIKM*.

[64] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban Computing: Concepts, Methodologies, and Applications. *ACM Trans. Intell. Syst. Technol.* 5, 3 (2014), 38:1–38:55.

Supplementary Material for 'Mining Relaxed Temporal Network Motifs'

Paper ID: 147

1 Details of Existing Temporal Network Motifs

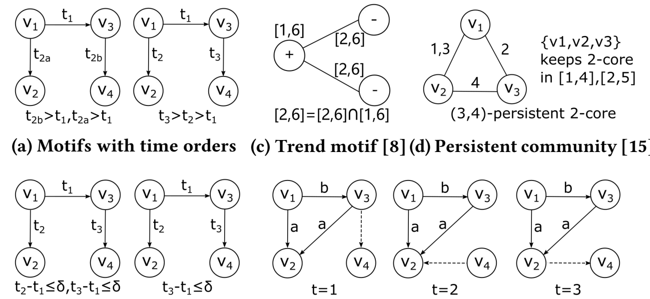
We first introduce details of existing temporal network motifs.

Most existing studies of temporal network motifs focus on time-dependent edges in temporal networks, *i.e.,* the dynamics come from the evolution/change of edges [2, 5, 7, 10–14, 16–19, 21]. They re-define motifs, where edges are attached with beginning timestamps and durations or only timestamps. The timestamps of the edges satisfy different time orders and time windows, *e.g.,* partial or total time orders, and local or global time windows. The difference between the total and partial time orders lies in whether the time order of all the edges is defined or not. The local time window is the time difference bound on adjacent edges, while the global time window is defined on all the edges. We next illustrate these concepts with examples.

Example 1: Figures 1(a) and 1(b) depict temporal motifs with different time orders and time windows, respectively, where each edge is attached with the timestamp. Assume that $t_3 > t_2, t_{2a}, t_{2b} > t_1$.
(1) For the temporal motif with the total time order, the time order of all the edges is defined. As the right figure in Figure 1(a) shows, the time order of edges (v_1, v_3), (v_1, v_2) and (v_3, v_4) is defined. All the isomorphic temporal subgraphs, where the timestamps on all the edges have the same time order, can be found in the network.
(2) For the temporal motif with the partial time order, the time order of a part of the edges is not defined. As the left figure in Figure 1(a) shows, the time order of edges (v_1, v_2) and (v_3, v_4) is not defined, due to the unknown relation between t_{2a} and t_{2b}. All the isomorphic temporal subgraphs, where the timestamps on a part of the edges have the same time order, can be found in the network.
(3) For the temporal motif with the global time window, the time difference bound is defined on all the edges. As the right figure in Figure 1(b) shows, the time difference of any two edges is bounded by the global time window δ. All the isomorphic temporal subgraphs, where the timestamps on all the edges satisfy the global time window, can be found in the network.
(4) For the temporal motif with the local time window, the time difference bound is defined on adjacent edges. As the left figure in Figure 1(b) shows, the time difference of edges (v_1, v_3) and (v_1, v_2), and the time difference of edges (v_1, v_3) and (v_3, v_4) are bounded by the local time window δ, respectively. All the isomorphic temporal subgraphs, where the timestamps on adjacent edges satisfy the local time window, can be found in the network. □

SIGMOD '26, May 31–June 05, 2026, Bangalore, India

ACM ISBN 978-1-4503-XXXX-X/25/06
https://doi.org/XXXXXXX.XXXXXXX



(a) Motifs with time orders (c) Trend motif [8] (d) Persistent community [15]

(b) Motifs with time windows (e) Induced relational state [1]

Figure 1: Existing temporal network motifs

There are studies focusing more on persistent nodes or edges, or their time-dependent weights (labels) in temporal networks. Each of nodes, edges or their weights (labels) in their temporal motifs displays statics or similar dynamics over a user-defined period. [4, 8] impose an increasing or decreasing trend of weights on the nodes in a motif, *i.e.,* induced subgraphs with node weights evolving in a consistent trend over a period. [15] imposes time-persistent relations on the subgraph structure in a pattern called persistent community, *i.e.,* induced subgraphs of the node set preserving the k-core structures over a period. [1] imposes time-persistent relations on the edges in a pattern called induced relational state, *i.e.,* induced subgraphs with edge labels and directions keeping unchanged over a period. We next illustrate these concepts with examples.

Example 2: (1) Figure 1(c) depicts a trend motif, where the weights of one node have an increasing trend (denoted as '+') in $[1, 6]$, while the weights of other two nodes have decreasing trends (denoted as '-') in $[2, 6]$. The interval of the trend motif is the intersection of all the intervals of these trends. All the trend motifs with sufficiently large time intervals can be found in the temporal network.
(2) Figure 1(d) depicts a $(3, 4)$-persistent 2-core with thresholds $\theta = 3, \tau = 4$ and $\widetilde{k} = 2$ (referred to as a persistent community), where induced subgraphs of the node set $\{v_1, v_2, v_3\}$ preserve 2-cores in any 3-length subintervals of the interval $[1, 5]$ (referred to as a maximal $(3, 2)$-persistent-core interval). The node set $\{v_1, v_2, v_3\}$ is a persistent community when its core persistence (aggregate the length of all the maximal $(3, 2)$-persistent-core intervals) is not less than the threshold $\tau = 4$ (here, the length of the only one maximal $(3, 2)$-persistent-core interval is not less than 4, please refer to [15] for more details). Given three thresholds θ, τ and $\widetilde{k}$, all the (θ, τ)-persistent $\widetilde{k}$-cores can be found in the temporal network.
(3) Figure 1(e) depicts an induced relational state, which is the induced subgraph with nodes v_1, v_2 and v_3, where the labels and directions of induced edges keep unchanged in $[1, 3]$. All the induced relational states with sufficiently large time intervals can be found in the temporal network. □

Among these studies are [19] and [15], which both relax their topology constraints in the definitions of temporal motifs, due to the high computational cost or the need of capturing more sensible patterns. However, there has been little attention paid to the data errors. Our RTMs are defined as subgraphs appearing continuously in sufficiently large time intervals with edge label mismatches occurring at a limited number of timestamps, which are somehow similar to temporal motifs proposed in [1, 4, 8, 15], as their nodes or edges of motifs appear at all the timestamps of the motif interval. Different from these studies, our RTMs allow label mismatches to handle data errors, and can be computed in low polynomial time.

2 Proofs in Relaxed Temporal Motif Analyses

Proof of Proposition 1. There are two cases.

(1) Assume that there exists an edge e such that $L^t(e) = L^m(e)$ for $t \in [m, i-1] \cup [i+1, h]$, *i.e.,* $L^i(e) \neq L^m(e)$. If $\delta(h-m+1) \geq 1$, we have $e \in \mathsf{S}[m, h]$ and $e \notin \mathsf{S}[m, i]$.

(2) Assume that there exists an edge e such that $L^t(e) = L^m(e)$ for $t \in [m, i-2] \cup [i, h]$, *i.e.,* $L^{i-1}(e) \neq L^m(e)$. If $\delta(h-m+1) \geq 1 > \delta(i-m+1)$, we have $e \in \mathsf{S}[m, h]$ and $e \notin \mathsf{S}[m, i]$.

Putting these together, we have the conclusion.

Proof of Proposition 2. By the definition of the R set, we know that the number of continuous label mismatches for each edge $e \in \mathsf{R}[m, i]$ does not exceed the relaxation bound c in the interval $[m, i]$ for each $i \in [m+k-1, T]$. For any subintervals $[m, j]$ $(j \leq i)$, the number of continuous label mismatches for edge e also does not exceed the relaxation bound c. For each edge e in $\mathsf{S}^*[m, i] = \bigcup_{i \leq h \leq T} \mathsf{R}[m, h]$, the number of continuous label mismatches of edge e in the interval $[m, j]$ $(j \leq i)$, does not exceed the relaxation bound c, as all the intervals $[m, h]$ for $i \leq h \leq T$ contain $[m, j]$. Hence, we have the conclusion.

Proof of Proposition 3. By Proposition 2 and the definition of S edge sets, after removing edges e from the subgraph $G_s(\mathsf{S}^*[m, i])$, such that $L^m(e) \neq L^i(e)$, or the label mismatches of the edge e violate the relaxation bound δ, we can obtain the subgraph $G_{s'}(\mathsf{S}[m, i])$. Assume that there is a connected component of $G_{s'}$ that is not a maximal RTM. It implies that there is an RTM G' falling into the interval $[m, i]$ such that $G_{s'}$ is a subgraph of G' and G' has edges not in $G_{s'}$, which contradicts with the fact that connected components are maximal [3]. Hence, we have the conclusion.

Proof of Proposition 4. Assume that we are generating the maximal RTMs for the interval $[m, i]$ with the set $\mathsf{R}[m, i]$ $(m+k-1 \leq i \leq T)$. It is obvious that any edge e in the set $\mathsf{R}[m, i]$ does not belong to sets $\mathsf{R}[m, h]$ for $i+1 \leq h \leq T$, and hence $e \notin \mathsf{S}^*[m, h]$. As all the maximal RTMs are generated by $G_s(\mathsf{S}^*[m, h])$, they do not contain the edge e. That is, if there exists an edge $e \in \mathsf{R}[m, i]$ in the generated maximal RTM, edge e does not belong to any RTMs in intervals $[m, h]$ for $i+1 \leq h \leq T$. Hence, we have the conclusion.

Proof of Proposition 5. (1) Assume that there exists a generated maximal RTM with interval $[m, i]$ which can appear in an RTM with interval $[n, h]$ $(1 \leq n \leq m$ and $r+1 \leq h \leq T)$. By the definition of scope, we can verify that there is an edge e in the RTM such that $e \in \mathsf{R}[p_1, j_1], \ldots, \mathsf{R}[p_x, j_x]$ $(p_1, \ldots, p_x \in [1, m]$ and $\max(j_1, \ldots, j_x) = r)$. By the definition of the R set, we have $e \notin \mathsf{S}[p_1, h], \ldots, \mathsf{S}[p_x, h]$. Hence, the RTM with the edge e cannot appear in RTMs with intervals $[n, h]$ for $1 \leq n \leq m$, as $e \notin \mathsf{R}[q, h]$ $(1 \leq q \leq m$ and $q \neq p_1, \ldots, p_x)$. It is a contradiction.

(2) Assume that there is a generated maximal RTM with interval $[m, i]$ which can appear in an RTM with interval $[n, h]$ $(1 \leq n \leq l-1$ and $i \leq h \leq r)$. As defined by scope, one can verify that there is an edge e in the RTM such that $e \in \mathsf{R}[p_1, j_1], \ldots, \mathsf{R}[p_x, j_x]$ $(\min(p_1, \ldots, p_x) = l$ and $j_1, \ldots, j_x \in [i, T])$. Hence, $e \notin \mathsf{S}[1, h], \ldots, \mathsf{S}[l-1, h]$, and the RTM with the edge e cannot appear in RTMs with intervals $[n, h]$ for $1 \leq n \leq l-1$. It is a contradiction.

Putting these together, we have the conclusion.

3 Proofs in the Static Algorithm

Proof of Proposition 6. We show this by a loop invariant.

Loop invariant: before the start of each timestamp m, algorithm FRTM finds all the maximal and non-expandable RTMs for intervals $[1, i]$, $[2, i]$, $\cdots$, $[m, i]$ $(i \in [m+k-1, T])$.

For $m = 1$, it is easy to verify that the loop invariant holds. Assume that the loop invariant holds for $m > 1$, and we show that the loop invariant holds for $m+1$. (1) By Proposition 2 & 3, the procedure maxCheck correctly generates all the maximal RTMs for intervals $[m+1, i]$ $(i \in [m+k, T])$. The frequency threshold k is assured from the time length of the intervals. (2) By Propositions 4 & 5, the procedure expCheck correctly checks whether a maximal RTM is expandable or not, and generates maximal and non-expandable RTMs. This guarantees that FRTM correctly finds the maximal and non-expandable RTMs for all the intervals $[m+1, i]$ $(i \in [m+k, T])$. This shows that the loop invariant holds for $m+1$.

Putting these together, we have the conclusion.

4 Proofs in Optimization Strategies

Proof of Proposition 7. There are two cases of subgraph $G_{s'}$ generated by edges of a connected component in $G_s(\mathsf{S}^*[m, i])$.

(1) There exist some edges e in $G_{s'}$ such that $L^m(e) \neq L^i(e)$ or the label mismatches of e violate the relaxation bound c in $[m, i]$. By Proposition 3, we know that $G_{s'}$ cannot correspond to a maximal RTM with interval $[m, i]$, because these edges need to be removed.

(2) All edges e in $G_{s'}$ satisfy $L^m(e) = L^i(e)$ and the label mismatches of e satisfy the relaxation bound c in $[m, i]$, *i.e.,* all edges e belong to $\mathsf{S}[m, i]$. As $L^{m-1}(e) = L^m(e)$, all edges e belong to $\mathsf{S}[m-1, i]$. That is, $G_{s'}$ corresponds to an expandable RTM which can appear in the RTM with interval $[m-1, i]$.

Putting these together, we have the conclusion.

Proof of Proposition 8. (1) As $\delta \cdot \max L < 1$, edges in $\mathsf{S}^*[m, m+\max L-1]$, $\mathsf{S}^*[m, m+\max L-2]$, $\cdots$, $\mathsf{S}^*[m, m+k-1]$ keep label unchanged in intervals $[m, m+\max L-1]$, $[m, m+\max L-2]$, $\cdots$, $[m, m+k-1]$, respectively. That is, for any edges e in the edge set $\mathsf{S}^*[m, i]$ $(m+k-1 \leq i \leq m+\max L-1)$, the condition $L^m(e) = L^i(e)$ holds and the label mismatches of e satisfy the relaxation bounds δ and c, *i.e.,* $e \in \mathsf{S}[m, i]$. Hence, $\mathsf{S}^*[m, i] \subseteq \mathsf{S}[m, i]$ for $m+k-1 \leq i \leq m+\max L-1$. As $\mathsf{S}[m, j] \subseteq \mathsf{S}^*[m, j]$ for $m+k-1 \leq j \leq T$, $\mathsf{S}^*[m, i] = \mathsf{S}[m, i]$ for $m+k-1 \leq i \leq m+\max L-1$. Hence, we have the conclusion.

5 Proofs in the Incremental Algorithm

Proof of Proposition 9. Let N be the number of label mismatches for edge e in intervals $[m, T]$. If the label mismatches of edge e satisfy the relaxation bound c in interval $[m, T]$, there exists a timestamp $i \in [T+1, T+\Delta T]$ such that $L^m(e) = L^i(e)$ and the label mismatches of edge e satisfy relaxation bounds δ and c in interval $[m, i]$, as long as $N \leq \delta(i - m + 1)$ and the number of continuous label mismatches satisfy the relaxation bound c. That is, there exists a timestamp $i \in [T + 1, T + \Delta T]$ such that $e \in \mathsf{R}[m, i]$.

Proof of Proposition 10. Assume that an edge e in an RTM is an unaffected edge, *i.e.,* $e \notin \mathsf{R}[m, i]$ for $m \in [1, T - k + 1]$ and $i \in [T+1, T+\Delta T]$. It is obvious that $e \notin \mathsf{S}[m, i]$ for $m \in [1, T-k+1]$ and $i \in [T + 1, T + \Delta T]$. Hence, any RTMs containing the edge e in the set TF cannot appear in the RTMs falling into intervals $[m, T + 1], \cdots, [m, T + \Delta T]$ for $m \in [1, T - k + 1]$, *i.e.,* they are non-expandable for G'.

Proof of Proposition 11. Assume that all edges e of an RTM are affected, *i.e.,* they might belong to $\mathsf{R}[m, i]$ for $m \in [1, T - k + 1]$ and $i \in [T + 1, T + \Delta T]$. As long as the label mismatches of all these edges e satisfy the relaxation bound δ and $L^m(e) = L^j(e)$ for certain $j \in [T + 1, T + \Delta T]$, *i.e.,* $e \in \mathsf{S}[m, j]$, the RTM consisting of these edges is expandable for G'.

6 The Incremental Maintenance of DEL-Table

We next illustrate how to incrementally maintain DEL-Table, which can be incrementally maintained in $O((\Delta T + |L|)|E|)$ time with the lazy update strategy. We need to maintain DEL-Table for each edge e as follows. For each timestamp t, we store the label lab_t. (1) When the label lab_t changes at timestamp t (*i.e.,* $lab_t \neq lab_{t-1}$), we first obtain the interval $[tail_{lab_{t-1}}, t - 1]$ where the label of e is lab_{t-1}, and update the number $aft_q = t - q$ for timestamps $q \geq T + 1$ and $q \in [tail_{lab_{t-1}}, t - 1]$. Note that the numbers aft_q for timestamp $q \leq T$ are not updated for the lazy update strategy. Let p be the previous timestamp where the label of e is lab_t, *i.e.,* $p = tail_{lab_t}$. We next update both the numbers bef_t and aft_p to $p - t$ (negative values), store the number $dis_t = dis_p + t - p - 1$, and update the number $tail_{lab_{t-1}} = t - 1$ (the last timestamp with the label lab_{t-1}). (2) When the label lab_t does not change at timestamp t (*i.e.,* $lab_t = lab_{t-1}$), we need to store the number bef_t with $(\max(bef_{t-1}, 1) + 1)$ for timestamp t, and copy the number dis_{t-1} to dis_t.

We store the labels lab_t, and the numbers bef_t, dis_t and aft_t for timestamps t in interval $[T + 1, T + \Delta T]$, and update the number aft_t for timestamps $t \leq T$ at most $(|L| + 1)$ times with the lazy update strategy. Hence, DEL-Table is incrementally maintained in $O((\Delta T + |L|)|E|)$ time. Note that the labels lab_t, and the numbers bef_t and dis_t for timestamps $t \leq T$ do not need to be updated. When the number aft_t for each timestamp $t \leq T$ is used, we need to update the number aft_t to $aft_p - t + p$ at first if $aft_t \neq aft_p - t + p$ in $O(1)$ time, where $p = t - \max(bef_t, 1) + 1$.

7 Detailed Complexity Analyses

(1) Time complexity of the procedure edgeFilter. The process of computing the R edge sets by scanning DEL-Table takes $O(|E|)$ amortized time, as all the timestamps are checked once for each edge. However, the process of updating the array vioT needs $O(T|E|)$ worst time (all the timestamps might be updated). Hence, the procedure edgeFilter takes $O(T|E|)$ worst time.

(2) Time complexity of the procedure maxCheck. The time complexity of the procedure maxCheck is dominated by the process of computing connected components in the set checkCC. Different from the dynamic connectivity problem [6, 9], which is focused on determining the connectivity between two nodes, we need to identify which connected component each edge in the RTM belongs to. Hence, computing connected components for one interval needs at most $O(|E_{m,T}|)$ time, which is the same as computing connected components from scratch by utilizing disjoint sets [20]. Here, $|E_{m,T}|$ is the number of edges in $\mathsf{S}^*[m, m + k - 1]$, *i.e.,* the maximal edge number of all the connected components for one interval. Hence, the procedure maxCheck takes $O(|E_{m,T}|)$ worst time.

(3) Time complexity of the procedure expCheck. Let $avgI$ be the average number of intervals to be checked when determining whether an RTM is expandable. For one interval, the number of edges in all the maximal RTMs to be checked is at most $|E_{m,T}|$. Hence, the procedure expCheck takes $O(avgI \cdot |E_{m,T}|)$ worst time.

We next prove that $O(avgI) < O((T - k)^2)$ for an RTM. Assume that the interval length of an RTM is I. For any I-length intervals of RTMs, the number of intervals to be checked is $\Sigma_{z=1}^{T-I+1}(z \times (T - I + 1 - z) + (z - 1))$. As $k \leq I \leq T$, the number of all the intervals to be checked for any length intervals of RTMs is $\Sigma_{I=k}^{T}\Sigma_{z=1}^{T-I+1}(z \times (T - I + 1 - z) + (z - 1)) = \Sigma_{I=k}^{T}[(T - I)(T - I + 1)(T - I + 5)/6] < O((T - k)^4)$. It is easy to prove that The number of intervals of RTMs is $O((T - k)^2)$. Hence, $O(avgI) < O((T - k)^4/(T - k)^2) = O((T - k)^2)$.

(4) Time complexity of the algorithm FRTM. Let $\max E_{m,T}$ be the maximum number of $|\mathsf{S}^*[m, m + k - 1]|$ for each m.

For each m, (a) the procedure edgeFilter takes $O(T|E|)$ time, (b) the procedure maxCheck takes $O(T|E_{m,T}|)$ time, and (c) the procedure expCheck takes $O(avgI \cdot T|E_{m,T}|)$ time. There are in total $O(T - k + 1) = O(T)$ execution times. Hence, algorithm FRTM takes $O(avgI \cdot T^2 \max E_{m,T} + T^2|E|) = O(avgI \cdot T^2|E|)$ time.

(5) Space complexity of the algorithm FRTM. The space cost of FRTM is dominated by its key data structures.

(a) For DEL-Table, it costs $O((T + |L|)|E|) = O(T|E|)$ space.

(b) For arrays maxIntv and vioT, they cost $O(\delta \cdot T|E|)$ space.

(c) For array checkT, it costs $O(|L||E|)$ space for all the edges.

(d) For connected components, they cost $O(\max E_{m,T})$ space in total, as there is only one copy is maintained for all $\mathsf{CC}[i, T]$ and $\mathsf{checkCC}[i, T]$ ($i \in [m + k - 1, T]$).

(e) For the set TF^+, it cost $O(T^2 \max E_{m,T})$ space.

Therefore, algorithm FRTM takes $O(T|E| + T^2 \max E_{m,T})$ space.

(6) Time complexity of the algorithm DFRTM. Let $|E_{\mathsf{EIntR}}|$ and $|E_{\mathsf{MIntR}}|$ are the numbers of edges in intermediate result sets EIntR and MIntR, respectively.

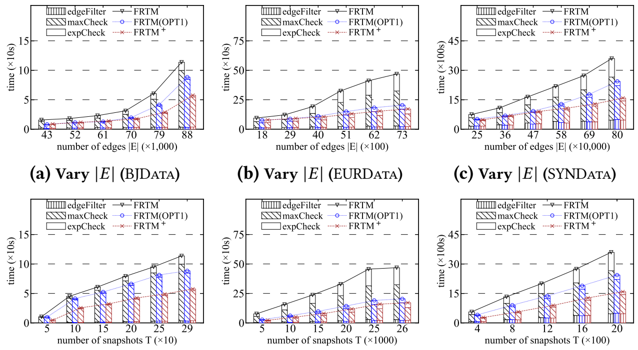
(a) For each $m \in [1, T + k - 1]$, procedures edgeFilter and maxCheck take $O(\Delta T|E_{\mathsf{EIntR}}|)$ time, and the procedure expCheck takes $O(avgI \cdot \Delta T|E_{\mathsf{EIntR}}|)$ time. The process of checking RTMs in sets MIntR whether they are expandable or not takes $O(avgI \cdot |E_{\mathsf{MIntR}}|)$ time for all $m \in [1, T + k - 1]$. Hence, this part in total takes $O(avgI \cdot T\Delta T|E_{\mathsf{EIntR}}| + avgI \cdot |E_{\mathsf{MIntR}}|)$ time.

(b) For each $m \geq T - k + 2$, the time complexity is the same as the static algorithm, which in total takes $O(avgI \cdot (\Delta T)^2|E|)$ time.

Therefore, incremental algorithm DFRTM takes $O(avgI \cdot (T\Delta T |E_{\mathsf{EIntR}}| + (\Delta T)^2|E| + |E_{\mathsf{MIntR}}|))$ time.

8 Extra Experimental Tests

Exp-1: Extra tests of static algorithms. We further test the running time of static algorithm FRTM, FRTM(OPT1) and FRTM^+ *w.r.t.*



(a) Vary $|E|$ (BJData) (b) Vary $|E|$ (EURData) (c) Vary $|E|$ (SYNData)

(d) Vary T (BJData) (e) Vary T (EURData) (f) Vary T (SYNData)

Figure 2: Algorithms FRTM, FRTM(OPT1) and FRTM⁺

Table 1: Parameters of static algorithms

Datasets	Average $\lvert E_{m,T}\rvert$	max$E_{m,T}$	*avgI*		
			FRTM	FRTM(OPT1)	FRTM⁺
BJData ($\lvert E\rvert = 88,396$)	57,676	75,146	34.67	1.52	1.14
EURData ($\lvert E\rvert = 7,334$)	5,048	6,695	203,739	791.42	475.54
SYNData ($\lvert E\rvert = 800K$)	218,954	301,473	415.82	21.67	10.20

the number $|E|$ of edges and the number T of snapshots, as well as the parameters used in the time complexity analysis.

Exp-1.5. To evaluate the impacts of the number $|E|$ of edges, we varied $|E|$ from 43,000 to 88,396 for BJData, from 1,800 to 7,334 for EURData, and from $250K$ to $800K$ for SYNData, respectively. We fixed $\delta = 4\%$, $c = 3$ and $k = 10$ and randomly selected edges by fixing $|V|= (69416, 2707, 400K)$ for (BJData, EURData, SYNData), respectively. The results are reported in Figures 2(a)-2(c).

When varying the number $|E|$ of edges, the running time of all three algorithms increases with the increment of $|E|$. This is consistent with the complexity analysis. Moreover, FRTM⁺ is on average (1.80, 2.05, 1.90) and (1.16, 1.09, 1.20) times faster than FRTM and FRTM(OPT1) on (BJData, EURData, SYNData), respectively.

When varying the number $|E|$ of edges, the running time of all three procedures increases with the increment of $|E|$. Procedure maxCheck takes the most time, and is on average (72%, 55%, 56%) for FRTM, (81%, 59%, 64%) for FRTM(OPT1) and (79%, 53%, 60%) for FRTM⁺ on (BJData, EURData, SYNData), respectively.

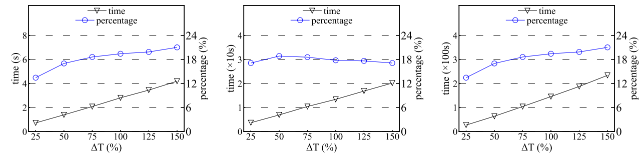
Exp-1.6. To evaluate the impacts of the snapshot number T, we varied T from 50 to 288 for BJData, from 5,000 to 26,304 for EURData, and from 400 to 2,000 for SYNData, respectively. We fixed $\delta = 4\%$, $c = 3$ and $k = 10$, and used the networks for all the datasets. The results are reported in Figures 2(d)-2(f).

When varying the snapshot number T, the running time of all three algorithms increases with the increment of T. This is consistent with the complexity analysis. Moreover, FRTM⁺ is on average (1.94, 3.00, 2.21) and (1.69, 1.22, 1.53) times faster than FRTM and FRTM(OPT1) on (BJData, EURData, SYNData), respectively.

When varying the snapshot number T, the running time of all three procedures increases with the increment of T. Procedure maxCheck takes the most time, and is on average (88%, 63%, 61%) for FRTM, (93%, 74%, 72%) for FRTM(OPT1) and (90%, 62%, 61%) for FRTM⁺ on (BJData, EURData, SYNData), respectively.

Exp-1.7. We tested the maximum and average values of $|E_{m,T}|$ and $|I|$, fixed $\delta = 4\%$, $c = 3$ and $k = 10$, and used the entire temporal networks for all the datasets. The results are reported in Table 1. Note that $|E_{m,T}|$ is the number of edges in $S^*[m, m+k-1]$, and $|I|$ is the maximal number of intervals to be checked when checking the expandable property of an RTM; max$E_{m,T}$ is the largest $|E_{m,T}|$, while *avgI* is the average of $|I|$.

$|E_{m,T}|$ is less than $|E|$ to varying degrees, as edge labels in different datasets change with different frequencies. $|E|$ is (1.18, 1.10, 2.65) times of max$E_{m,T}$ on (BJData, EURData, SYNData), respectively. Moreover, the values of *avgI* vary significantly in different datasets for the same reason. The value of *avgI* for FRTM is (22.81, 257.43, 19.19) and (30.41, 428.44, 40.77) times more than FRTM(OPT1) and FRTM⁺ on (BJData, EURData, SYNData), respectively. It verifies the effectiveness of optimization strategies.



(a) Vary ΔT (BJData) (b) Vary ΔT (EURData) (c) Vary ΔT (SYNData)

Figure 3: The incremental maintenance of DEL-Table

Exp-2: Extra tests of incremental algorithms. We further evaluate the space cost of intermediate results for incremental algorithms, the parameters used in the time complexity analysis, and test the incremental maintenance time of DEL-Table *w.r.t.* the number ΔT of increased timestamps.

Exp-2.6. We tested the value of max$E_{m,(T+\Delta T)}$, the value of $|E_{\mathsf{EIntR}}|$ and $|E_{\mathsf{MIntR}}|$ and the memory cost of sets EIntR and MIntR in practice (using function GetProcessMemoryInfo). We fixed $\delta = 4\%$, $c = 3$, $k = 10$, $\Delta T = 75\%$ and the original timestamps $T = (112, 10400, 800)$ for (BJData, EURData, SYNData), respectively. The results are reported in Table 2. Note that max$E_{m,(T+\Delta T)}$ is the maximal number of edges in $\bigcup_{m+k-1\leq h\leq T+\Delta T} \mathsf{R}[m,h]$ for each m, sets EIntR and MIntR are necessary intermediate results for incremental algorithms, and $|E_{\mathsf{EIntR}}|$ and $|E_{\mathsf{MIntR}}|$ are the numbers of edges in sets EIntR and MIntR, respectively.

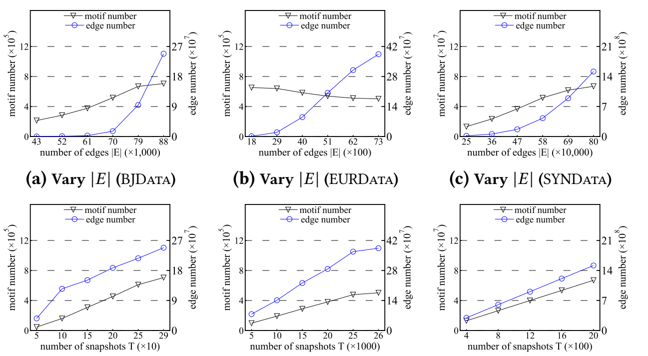
Parameter $|E_{m,(T+\Delta T)}|$ is less than $|E|$ to various degrees, and the $|E|$ is (1.18, 1.10, 2.65) times larger than max$E_{m,(T+\Delta T)}$ on (BJData, EURData, SYNData), respectively. In addition, E_{EIntR} is less than $|E|$, and E_{MIntR} is much less than T^2 max$E_{m,(T+\Delta T)}$. It justifies the usage of EIntR and MIntR. Moreover, the memory cost of sets EIntR and MIntR is rational.

Exp-2.7. To evaluate the impacts of the number ΔT of increased timestamps on the incremental maintenance time of DEL-Table, we varied ΔT from 25% to 150% for all the datasets, while fixed the original timestamps $T = (112, 10400, 800)$ for (BJData, EURData, SYNData), respectively. Note that the incremental maintenance time of DEL-Table is irrelevant to δ, c and k. The results are reported in Figures 3(a)-3(c).

When varying the number ΔT from 25% to 150%, the incremental maintenance time of DEL-Table increases linearly, and grows from on average (0.72s, 3.67s, 26.76s) to (4.19s, 20.19s, 234.07s) on (BJData, EURData, SYNData), respectively. These are consistent with the time complexity analysis. Moreover, the percentage of the incremental maintenance time of DEL-Table in algorithm DFRTM⁺

Table 2: Memory cost and parameters of incremental algorithms (ΔT = 75%)

Datasets	$\max E_{m,(T+\Delta T)}$	$\lvert E_{\text{EIntR}}\rvert$	$\lvert E_{\text{MIntR}}\rvert$	Memory cost EIntR and MIntR
BJDATA ($\lvert E\rvert$ = 88, 396)	75, 145	75, 666	2, 118, 999	0.02 GB
EURDATA ($\lvert E\rvert$ = 7, 334)	6, 695	5, 616	167, 711	0.18 GB
SYNDATA ($\lvert E\rvert$ = 800K)	301, 473	697, 100	5, 392, 620	0.51 GB



(a) **Vary** $|E|$ (BJDATA) (b) **Vary** $|E|$ (EURDATA) (c) **Vary** $|E|$ (SYNDATA)

(d) **Vary** T (BJDATA) (e) **Vary** T (EURDATA) (f) **Vary** T (SYNDATA)

Figure 4: Extra tests of parameter sensitivity

is on average (12.57%∼14.49%, 17.12%∼18.86%, 13.46%∼21.03%) on (BJDATA, EURDATA, SYNDATA), respectively.

Exp-3: Extra tests of parameter sensitivity. We further test the number of RTMs generated by the algorithms, and the total number of edges in all RTMs, *w.r.t.* the number $|E|$ of edges and the number T of snapshots.

Exp-3.4. To evaluate the impacts of the number $|E|$ of edges, we varied $|E|$ from 43, 000 to 88, 396 for BJDATA, from 1, 800 to 7, 334 for EURDATA, and from 250K to 800K for SYNDATA, respectively. We fixed δ = 4%, c = 3 and k = 10. The results are reported in Figures 4(a)-4(c).

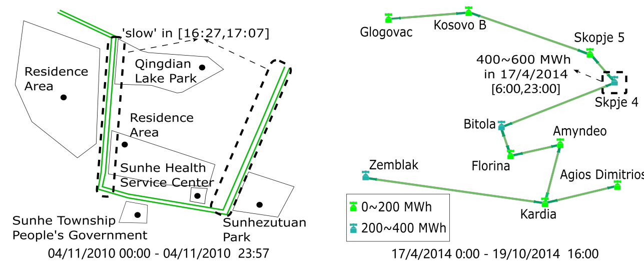
When varying the number $|E|$ of edges, the number of RTMs increases sharply on most datasets with the increment of $|E|$, except for EURDATA. It increases by (230.3%, 407.2%) on (BJDATA, SYNDATA), respectively, while it decreases by 22.9% on EURDATA. The reason is that there are more connected edges in the RTMs when $|E|$ is large, which reduces the number of RTMs generated on EURDATA. Moreover, the total number of edges in all RTMs increases sharply by (68918.9%, 28970.9%, 8352.5%) on (BJDATA, EURDATA, SYNDATA), respectively.

Exp-3.5. To evaluate the impacts of the snapshot number T, we varied T from 50 to 288 for BJDATA, from 5, 000 to 26, 304 for EURDATA, and from 400 to 2, 000 for SYNDATA, respectively. We fixed δ = 4%, c = 3 and k = 10. The results are reported in Figures 4(d)-4(f).

When varying the snapshot number T, the number of RTMs increases sharply with the increment of T, as there are more edges forming the RTMs when T is large. It increases by (1422.8%, 414.7%, 417.9%) on (BJDATA, EURDATA, SYNDATA), respectively. Moreover, the total number of edges in all RTMs increases by (583.2%, 402.7%, 412.0%) on (BJDATA, EURDATA, SYNDATA), respectively.

Exp-4. Extra case studies. We further conducted case studies on BJDATA and EURDATA by the result of FRTM with δ = 4% and k = 10, *i.e.,* no settings for c.

Case 1. For BJDATA, we use the green color to represent roads with 'fast' traffic status labels. As Figure 5(a) depicts, we identify the



(a) An RTM with no settings for the relaxation bound c on BJDATA **(b) An RTM with no settings for the relaxation bound c on EURDATA**

Figure 5: Extra case studies on real-life datasets

RTM corresponding to the roads in the *Sunhe community*. However, the RTM treats all these road labels within [0:00, 23:57] as 'fast', and it does not make sense for the roads labeled as 'slow' within [16:27, 17:07] in another RTM (circled by dashed lines).

Case 2. For EURDATA, we use sea green and lime green colors to represent merging points of transmission lines with energy demand signals 200 ∼ 400 MWh and 0 ∼ 200 MWh in the RTM, respectively. As Figure 5(b) depicts, we identify the RTM corresponding to the area in Italy. However, the RTM treats the labels of all these merging points of transmission lines within [17/4/2014 0:00, 19/10/2014 16:00] as low energy demand signals, *i.e.,* not more than 400 MWh, and it does not make sense for the merging point named '$D-182$' labeled as '400 ∼ 600 MWh' energy demand signals within [17/4/2014 6:00, 17/4/2014 23:00] in another RTM (circled by dashed lines).

These verify the effectiveness of the local relaxation bound c.

References

[1] Rezwan Ahmed and George Karypis. 2012. Algorithms for mining the evolution of conserved relational states in dynamic networks. *KAIS* 33, 3 (2012), 603–630.
[2] Xinwei Cai, Xiangyu Ke, Kai Wang, Lu Chen, Tianming Zhang, Qing Liu, and Yunjun Gao. 2024. Efficient Temporal Butterfly Counting and Enumeration on Temporal Bipartite Graphs. *Proc. VLDB Endow.* 17, 4 (2024), 657–670.
[3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms.* MIT press, Cambridge, Mass.
[4] Elise Desmier, Marc Plantevit, Céline Robardet, and Jean-François Boulicaut. 2013. Trend mining in dynamic attributed graphs. In *ECML/PKDD.*
[5] Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. 2015. Commit: A scalable approach to mining communication motifs from dynamic networks. In *SIGMOD.*
[6] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *JACM* 48, 4 (2001), 723–760.
[7] Yuriy Hulovatyy, Huili Chen, and Tijana Milenković. 2015. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinform.* 31, 12 (2015), i171–i180.
[8] Ruoming Jin, Scott McCallen, and Eivind Almaas. 2007. Trend motif: A graph mining approach for analysis of dynamic complex networks. In *ICDM.*
[9] Bruce M. Kapron, Valerie King, and Ben Mountjoy. 2013. Dynamic graph connectivity in polylogarithmic worst case time. In *SODA*, Sanjeev Khanna (Ed.). SIAM, 1131–1142.
[10] Chrysanthi Kosyfaki, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2019. Flow Motifs in Interaction Networks. In *EDBT.*
[11] 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), P11005:1–P11005:18.
[12] Lauri Kovanen, Kimmo Kaski, János Kertész, and Jari Saramäki. 2013. Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences. *PNAS* 110, 45 (2013), 18070–18075.
[13] Rohit Kumar and Toon Calders. 2018. 2scent: An efficient algorithm to enumerate all simple temporal cycles. *Proc. VLDB Endow.* 11, 11 (2018), 1441–1453.
[14] Geon Lee and Kijung Shin. 2023. Temporal hypergraph motifs. *KAIS* 65, 4 (2023), 1549–1586.
[15] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent Community Search in Temporal Networks. In *ICDE.*

[16] Penghang Liu, Naoki Masuda, Tomomi Kito, and Ahmet Erdem Sarıyüce. 2022. Temporal motifs in patent opposition and collaboration networks. *Scientific reports* 12, 1 (2022), 1917:1–1917:11.
[17] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *WSDM*.
[18] Noujan Pashanasangi and C Seshadhri. 2021. Faster and generalized temporal triangle counting, via degeneracy ordering. In *KDD*.
[19] Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. 2014. Event pattern matching over graph streams. *Proc. VLDB Endow.* 8, 4 (2014), 413–424.
[20] Robert Endre Tarjan. 1975. Efficiency of a good but not linear set union algorithm. *JACM* 22, 2 (1975), 215–225.
[21] Qiankun Zhao, Yuan Tian, Qi He, Nuria Oliver, Ruoming Jin, and Wang-Chien Lee. 2010. Communication motifs: a tool to characterize social communications. In *CIKM*.