Finding Relaxed Temporal Network Motifs

Hanqing Chen Shuai Ma
NLSDE Lab, Beihang University
{chenhanqing,mashuai}@buaa.edu.cn

ABSTRACT

There is an urge need to relax temporal motifs in order to handle the data quality issues. In this study, we first propose a proper definition of relaxed temporal motifs by allowing limited label mismatches, to tolerate the data quality issues on a class of temporal networks, where the nodes and edges keep 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 analyses of the relaxed temporal motifs. 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.

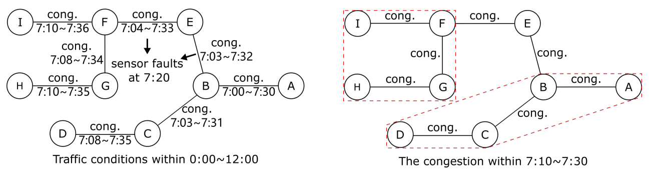
PVLDB Reference Format:
Hanqing Chen and Shuai Ma. Finding Relaxed Temporal Network Motifs. PVLDB, 14(1): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CSeCodesData/FRTM.

1 INTRODUCTION

Network (or graph) motif discovery aims at identifying recurrent and statistically significant patterns or subgraphs in networks [44, 56], which is one of the significant graph data mining problems [2, 63] (see, *e.g.,* [52, 63] for surveys). It provides valuable insights into the network functional abilities [44, 56], and has widespread applications in both industry and academia [5, 7, 12, 15, 32, 34, 40, 47, 58, 62]. To better analyze the dynamics in the data analytic systems and applications that could be modeled as graphs or networks, the temporal networks have drawn significant attention, which suggests the growing need for research into temporal motif discovery [3, 8, 11, 18, 19, 21, 24–29, 33, 37, 38, 48, 49, 65].

Existing studies on temporal motif discovery often reinterpret temporal motifs to meet the various demands of applications. Most studies propose temporal motifs, where edges are attached with beginning timestamps and durations or only timestamps [8, 18, 21, 25–29, 38, 48, 49, 65]. These typically identify all the isomorphic subgraphs, where the timestamps of edges satisfy different time orders and are limited in user-defined time windows. There are other



(a) Road network **(b) Traffic pattern**

Figure 1: Temporal motifs need relaxations

studies proposing temporal motifs as induced subgraphs, where node weights evolve in a consistent trend [11, 24], or edge labels and directions remain unchanged in a period [3]. Due to the high computational cost or the need of capturing more sensible patterns, there are studies starting to relax their topology constraints in the definitions of temporal motifs, such as utilizing dual simulation [41] instead of subgraph isomorphism to allow topology mismatches [57], and persistent k-core structures instead of fixed k-clique structures in a period to allow topology changes [33].

However, there has been little attention paid to the data quality issues, caused by *e.g.,* measurement errors or missing values [10, 16, 23, 39, 47]. Consider the example below.

Example 1: Consider a real-life example from the road network, shown in Figure 1. Figure 1(a) depicts the road network, where the edges represent roads labeled as either 'congested' (cong.) or 'fast', along with their timestamps at the minute level between 0:00 and 12:00. For simplicity, 'congested' edge labels are shown with time intervals, and 'fast' edge labels at all other timestamps are ignored. However, the sensors on two roads (BE and EF) at 7:20 encounter faults, which are mislabeled as 'fast'. A data analyst aims to identify a pattern that indicates a traffic jam lasting at least 20 minutes, and finds two separated jam areas (circled by dashed lines in Figure 1(b)). In fact, these two areas should be merged into a large area if data missing is considered. In this case, it is irrelevant to the handling of any topology constraints [33, 57], and none of the existing studies on temporal motifs aim to address this issue. To tackle this issue, a natural idea is to relax the exact road label matching constraint at certain timestamps. □

In this study, we focus on alleviating the data quality issues 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, 42, 43]. Road networks (as shown in Figure 1(a)) and energy transmission networks [22, 60] typically fall into this category, where sensor data can be noisy in urban computing [66].

However, the study of such temporal motifs has raised several challenges. (1) How to propose a proper definition of temporal motifs that tolerate the quality issues of real-world data? The existing temporal motifs only relax their topology constraints [33, 57], which cannot address these issues. (2) The efficiency of finding temporal motifs is significant. Most existing methods of finding temporal

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

motifs are computationally intractable, *i.e.,* NP-hard, as (subgraph) isomorphism tests or exponential-time subgraph enumeration are essentially unavoidable [3, 11, 18, 21, 24–28, 33, 38, 48, 49, 65], and they adopt various techniques to tackle this issue, *e.g.,* sampling [29, 36, 53, 54, 59], dual simulation [41, 57], and parallel approaches [8, 17]. (3) Dynamic algorithms are essentially needed to handle the dynamic and continuously updated temporal networks, which have not been overlooked by existing studies.

Contributions & roadmap. We propose a concept of relaxed temporal motifs (or simply RTMs) for temporal networks, where the nodes and edges are fixed, but the edge labels vary regularly with timestamps, and develop efficient both static and incremental algorithm for finding 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 timestamp) to reinterpret the recurrent and statistically significant nature of motifs in temporal networks (Section 2). Different from most existing temporal motifs, our RTMs can handle data quality issues, 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 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).

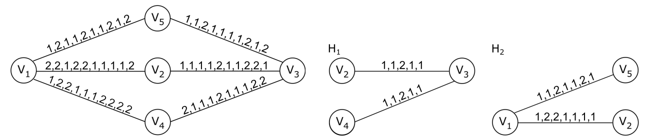
(4) Using both real-life and synthetic datasets, we conduct an extensive experimental study (Section 5). We find that (a) our static algorithm FRTM$^+$ runs fast even on the large temporal networks with 400 thousand nodes and 1.6 billion edges, (b) our incremental algorithm DFRTM$^+$ is much faster than its static counterpart FRTM$^+$, and is on average (2.70, 1.88, 2.04) times faster on datasets (BJData, EURData, SYNData), respectively, and (c) our RTMs are effective for practical applications with case studies on real-life datasets BJData and EURData.

Due to space limitations, detailed proofs and algorithm complexity analyses, and extra experimental studies are available at https://github.com/CSeCodesData/FRTM/paper-full.pdf.

2 PRELIMINARY

In this section, we introduce the basic concepts and the problem statement, and we use the terms '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 representing $(T_f - T_b + 1)$ timestamps, in which $T_b \leq T_f$ are the beginning and finishing timestamps, respectively, and (4) L is a set of label functions such that for each timestamp $t \in [T_b, T_f]$, L^t is a function that maps each edge $e \in E$ to a label (integers 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.



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

Figure 2: Running example

In temporal networks, *e.g.,* road networks, the nodes and edges are fixed, but the edge labels vary *w.r.t.* timestamps [6, 42, 43]. Intuitively, (1) a temporal network $G(V, E, L)$ essentially denotes a sequence of $T = T_f - T_b + 1$ standard networks, *i.e.,* $\langle G_1(V, E, L^{T_b}), \ldots, G_{t-T_b+1}(V, E, L^t), \ldots, G_{T_f - T_b+1}(V, E, L^{T_f})\rangle$, and (2) its edge labels $L^t(e)$ $(t \in [T_b, T_f])$ specify the states, distances or traveling duration [19, 60] 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.,* $\hat{T}_f - \hat{T}_b + 1 \geq k$ (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.,* $L^t(e) \neq L^{\hat{T}_b}(e)$ 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 $L^t(e) \neq L^{\hat{T}_b}(e)$ for all $t \in [\hat{t}_b, \hat{t}_f]$, $\hat{t}_f - \hat{t}_b + 1 \leq c$.

Here, $0 \leq c < k$ are integers, and $0 \leq \delta \leq 1$. We next discuss the rationale to introduce the two relaxation bounds.

Example 2: Recall the road network in Example 1.

(1) The global relaxation bound δ assures 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, 7:30] in Figure 1(b) can be identified, instead of two separate ones. In this way, the mislabeled roads (BE and EF) at timestamp 7:20 are handled.

(2) The local relaxation bound c assures that each occurrence of edge label mismatches only continues 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' with interval [0:00, 12:00]. However, this pattern essentially treats all the edge labels within [0:00, 12:00] as 'fast', and it does not make sense for the edges labeled as 'cong.' within [7:10, 7:30].

However, with any local relaxation bound $1 \leq c < k$, one can verify that this unreasonable pattern cannot be identified any more. That is, the local relaxation bound remedies that side effects of the global relaxation bound for handling data quality issues. □

Maximal RTMs. A 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. A 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 $\hat{T}_b \leq \hat{t}_b$ and $\hat{t}_f \leq \hat{T}_f$, and for each edge $e \in E_s$, $L^{\hat{t}_b}(e) = L^{\hat{T}_b}(e)$.

We also say that a 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 RTM $G_s(V_s, E_s, \hat{T}_b, \hat{T}_f)$ in G.
Example 4: Given the temporal network G in Figure 2(a), the 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. Our proposed RTMs have certain potential applications, *e.g.,* discover traffic patterns with long-time congested or bottleneck roads in road networks [30, 46], and energy consumption patterns with continuous high energy demand signals in transmission networks [22, 50]. In addition, our relaxation bounds are suitable for handling data quality issues in these applications, as short-time traffic mitigation or sudden sensor faults are common.
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, 42, 43]. However, our motifs can be extended to handling varying edges, by treating missing edges with virtual labels. (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. Existing studies on temporal motifs [3, 11, 24, 33] are computationally intractable, and do not consider the data quality issues, while our RTMs can be computed in low polynomial time. (3) Different from existing studies on relaxing topology constraints in [33, 57], our RTMs focus on relaxing label constraints to handle data quality issues, which have not 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 is an algorithm that finds all the maximal and non-expandable* RTMs *in $O(\max I \cdot T^2|E|)$ time. Here, $\max I$ is the maximal number of intervals to be checked when determining whether a* RTM *is expandable or not.* □

3.1 Analyses of RTMs

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

Let $\mathsf{S}[m, i]$ ($1 \leq m \leq T - k + 1$ and $m + k - 1 \leq i \leq T$) be the set of edges $e \in E$ such that $L^m(e) = L^i(e)$ in the interval $[m, i]$, and the label mismatches of edge e satisfy the relaxation bounds δ and c. When $\delta = 0$ or $c = 0$, one can verify that $\mathsf{S}[m, h] \subseteq \mathsf{S}[m, i]$ for $m + k - 1 \leq i < h \leq T$. 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 $e \notin \mathsf{S}[m, i]$ ($m + k - 1 \leq i < h \leq T$).* □

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

Let $\mathsf{R}[m, i] = \mathsf{S}[m, i] \setminus \bigcup_{i<h\leq T} \mathsf{S}[m, h]$ for $m+k-1 \leq i < T$, 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 obtain, the first challenge is transformed into obtaining R edge sets. As will be seen shortly, we design a data structure DEL-Table to facilitate the computation.
Challenge 2: Once we obtain the R edge sets, the next challenge is how to efficiently generate maximal RTMs by these R edge sets.

For convenience, we denote $\bigcup_{i\leq h\leq T} \mathsf{R}[m, h]$ as $\mathsf{S}^*[m, i]$ for $m + k - 1 \leq i \leq T$. 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 $j \leq i$.* □

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

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

For the edge set $\mathsf{S}^*[m, i]$, we can derive the temporal subgraph $G_s(V(\mathsf{S}^*[m, i]), \mathsf{S}^*[m, i], m, i)$ with the interval $[m, i]$, 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 $G_s(\mathsf{S}^*[m, i])$ ($1 \leq m \leq T - k + 1$ and $m + k - 1 \leq i \leq T$) such that $L^m(e) \neq L^i(e)$, or the 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. For convenience, we use $\mathsf{S}^*[m, T + 1]$ to denote $\emptyset$. We

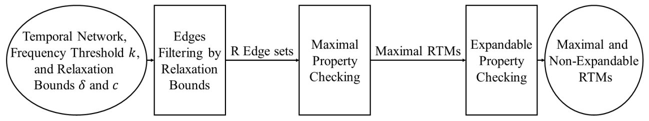

Temporal Network, Frequency Threshold k, and Relaxation Bounds δ and c
Edges Filtering by Relaxation Bounds
R Edge sets
Maximal Property Checking
Maximal RTMs
Expandable Property Checking
Maximal and Non-Expandable RTMs

Figure 3: The overall algorithm framework

adopt the right-to-left (R2L) scheme, *i.e.,* generate maximal RTMs from intervals $[m,T]$ to $[m,m+k-1]$. For each interval $[m,i]$, we can add edges in the set $\mathsf{R}[m,i]$ to the ccs of the subgraph $G_s(\mathsf{S}^*[m,i+1])$, 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. After that, we can obtain the maximal RTMs by newly generated ccs. Note that there is also a left-to-right (L2R) scheme, *i.e.,* generate maximal RTMs from intervals $[m,m+k-1]$ to $[m,T]$, which adds the same number of edges as the R2L scheme, but needs to remove 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 the interval $[m,i]$ is expandable, the label mismatches of each edge in the maximal RTM may satisfy bounds δ and c in any other interval containing $[m,i]$. Hence, the final challenge is how to check fewer intervals, when determining whether a maximal RTM is expandable or not.

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

Proposition 4 reveals that it is unnecessary to check all the intervals containing $[m,i]$, 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. 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 to which e belongs. We use $\mathsf{scope}[e]$ to denote the interval $[min(p_1,\ldots,p_x), max(j_1,\ldots,j_x)]$, and we have the following observation.

Proposition 5: *Let* $[l,r]$ *be the intersection of* $\mathsf{scope}[e]$ *for edges* e *in the* RTM. *We can identify an expandable* RTM *with interval* $[m,i]$ *by checking if it appears in maximal* RTM*s with intervals* $[n,h]$, *for* $i \leq h \leq r$ *when* $l \leq n \leq m-1$, *and for* $i+1 \leq h \leq r$ *when* $n=m$. □

Proposition 5 reveals that we can check fewer intervals by computing the intersection of $\mathsf{scope}[e]$ for each edge e in the RTM. Note that the label mismatches of each edge e in the RTM satisfy the relaxation bound c in any intervals $[n,h]$ by the definition of R edge sets. Hence, for each edge e in the RTM, we only need to check whether $L^n(e) = L^h(e)$ holds, and whether the label mismatches of e satisfy the relaxation bound δ. As will be seen shortly, we use DEL-Table to facilitate the computation. Moreover, since these intervals have the left endpoints $n \leq m$, a top-to-bottom (T2B) scheme, *i.e.,* computing RTMs for intervals $[m,m+k-1]$, $\cdots$, $[m,T]$ for each m from 1 to $T-k+1$, is a suitable choice.

By Propositions 3 & 5, we know 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

By Proposition 1, we know that it is difficult to compute the S edge sets directly. Hence, we choose to compute the R edge sets. Moreover, following Propositions 2 & 3, we can generate maximal RTMs for an interval $[m,i]$ by removing edges e from $G_s(\mathsf{S}^*[m,i])$ such that $L^m(e) \neq L^i(e)$, or the label mismatches of e violate the relaxation bound δ, and then recomputing connected components. By comparing the R2L and L2R schemes, we choose to generate maximal RTMs in the R2L scheme, *i.e.,* from intervals $[m,T]$ to $[m,m+k-1]$. Finally, following Propositions 4 & 5, we can check fewer intervals when determining if a maximal RTM is expandable by computing the interval $\mathsf{scope}[e]$ for each edge 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 $L^m(e) \neq L^i(e)$, or the label mismatches of e violate the relaxation bound δ. (3) Once it generates maximal RTMs for an interval $[m,i]$, it immediately checks the expandable property of maximal 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 each edge $e \in E$ such that $L^m(e) = L^i(e)$, and the label mismatches of e satisfy the relaxation bounds δ and c to obtain the edge sets $\mathsf{R}[m,i]$ for $i \in [m+k-1,T]$. By the definition of the R edge sets, for each edge $e \in \mathsf{R}[m,i]$, $L^m(e) \neq L^h(e)$ holds, or the label mismatches of e violate relaxation bounds δ or c in any intervals $[m,h]$ for $i < h$. Hence, we can find such an interval $[m,i]$ (referred to as *maximum valid intervals*), and know $e \in \mathsf{R}[m,i]$ if $[m,i]$ exists, and $e \notin \mathsf{R}[m,h]$ for $h \in [m+k-1,T]$, otherwise. To efficiently identify the maximum valid intervals, we design a data structure called DEL-Table, which can be constructed in $O((T+|L|)|E|)$ time and irrelevant to the specific δ, c and k.

DEL-Table. DEL-Table is a $(T+|L|)\times|E|$ table, which stores the label perseveration information of each edge *w.r.t.* timestamps in different time orders (referred to as DEL-Table). 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, which are different from $L^t(e)$ in interval $[1,t]$, and (3) the numbers bef_t and aft_t, which can take both positive and negative values. The positive number bef_t (resp. aft_t) indicates the number of timestamps that edge e keeps the same label lab_t until the timestamp t in the time order (resp. in the reverse order). The negative number bef_t (resp. aft_t) indicates the time difference between the 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 the negative numbers bef_t (resp. aft_t) at each timestamp when edge e changes its label in the time order (resp. in the reverse order). Moreover, it stores the last timestamps $tail_{lab}$ when each edge has the label lab, which is used for dynamic updating.

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) the timestamp t. Given a timestamp t, we can identify the interval containing t such that the edge keeps the same label as lab_t by $[n,h] = [t - max(bef_t,1)+1, t+max(aft_t,1)-1]$, and identify the most recent

	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)	$lab_1=2$	$lab_2=2$	$lab_3=1$	$lab_4=2$	$lab_5=2$	$lab_6=1$	$lab_7=1$	$lab_8=1$	$lab_9=1$	$lab_{10}=2$	$tail_1=9$	$tail_2=10$
	$bef_1=-\infty$	$bef_2=2$	$bef_3=-\infty$	$bef_4=-2$	$bef_5=2$	$bef_6=-3$	$bef_7=2$	$bef_8=3$	$bef_9=4$	$bef_{10}=-5$		
	$aft_1=2$	$aft_2=-2$	$aft_3=-3$	$aft_4=2$	$aft_5=-5$	$aft_6=4$	$aft_7=3$	$aft_8=2$	$aft_9=-\infty$	$aft_{10}=-\infty$		
	$dif_1=0$	$dif_2=0$	$dif_3=2$	$dif_4=1$	$dif_5=1$	$dif_6=4$	$dif_7=4$	$dif_8=4$	$dif_9=4$	$dif_{10}=5$		
(v_1,v_5)	$lab_1=1$	$lab_2=2$	$lab_3=1$	$lab_4=1$	$lab_5=2$	$lab_6=1$	$lab_7=1$	$lab_8=2$	$lab_9=1$	$lab_{10}=2$	$tail_1=9$	$tail_2=10$
	$bef_1=-\infty$	$bef_2=-\infty$	$bef_3=-2$	$bef_4=2$	$bef_5=-3$	$bef_6=-2$	$bef_7=2$	$bef_8=-3$	$bef_9=-2$	$bef_{10}=-2$		
	$aft_1=-2$	$aft_2=-3$	$aft_3=2$	$aft_4=-2$	$aft_5=-3$	$aft_6=2$	$aft_7=-2$	$aft_8=-2$	$aft_9=-\infty$	$aft_{10}=-\infty$		
	$dif_1=0$	$dif_2=1$	$dif_3=1$	$dif_4=1$	$dif_5=3$	$dif_6=2$	$dif_7=2$	$dif_8=5$	$dif_9=3$	$dif_{10}=6$		

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

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, 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 the edge (v_1, v_5) as an example. Given the timestamp $t = 7$, 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 the DEL-Table for edge e. We alternately check the last timestamp such that the labels of e either continue matching $L^m(e)$ or continue mismatching from the currently checked timestamp t, while dynamically maintaining the following information in $[m, t]$: (1) the maximum number $intvL$ of continuous label mismatches for edge e, or (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 $[m, lastT]$. The process stops if one of the following conditions is met: (1) $intvL > c$ (*i.e.,* violate the relaxation bound c), and (2) $t = T$ (*i.e.,* the last timestamp). If $lastT \geq m + k - 1$, then $e \in \mathsf{R}[m, lastT]$, otherwise, $e \notin \mathsf{R}[m, h]$ for $h \geq m + k - 1$. Hence, using DEL-Table, we do not need to check all timestamps to identify the R edge set for edge e.

We next use an example to show the above process.

Example 6: We consider the threshold $k = 5$, the relaxation bounds $\delta = 0.3$ and $c = 2$, and the starting timestamp $t = 1$. Take the edge (v_1, v_2) in Figure 4 as an example.

(1) For the first timestamp $t = 1$, the next checked timestamp, where the label of (v_1, v_2) matches lab_1, is $1 + aft_1 - 1 = 2$. (2) For the timestamp $t = 2$, we store $intvL = 0$ and $lastT = 2$. The next checked timestamp, where the label of (v_1, v_2) mismatches, is $2 - aft_2 - 1 = 3$. (3) For the timestamp 3, $intvL$ is updated to 1. The next checked timestamp is 5. (4) For the timestamp 5, $lastT$ is updated to 5. The next checked timestamp is 9. (5) For the timestamp 9, $intvL$ is updated to 4 ($4 > c$). The process stops. As $lastT = m + k - 1 = 5$, we have $(v_1, v_2) \in \mathsf{R}[1, 5]$. □

Procedure edgeFilter. We next present the details of the procedure edgeFilter to compute the edge sets $\mathsf{R}[m, i]$ ($i \in [m + k - 1, T]$) 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 threshold k, relaxation bounds δ and c, arrays maxIntv, checkT and vioT, and the timestamp m, and returns the edge sets $\mathsf{R}[m, i]$ 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 checked timestamp when the checking process of 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 the temporal network $G(V, E, 1, T, L)$, the threshold $k$, relaxation bounds $\delta$ and $c$, arrays maxIntv, checkT and vioT, and the 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 the 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, $[n, h]$ 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 $e \in \mathsf{R}[m, lastT]$ and $lastT > h$, we need to add $[m, lastT]$ to maxIntv$[e, L^m(e)]$.

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

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 have scanned DEL-Table and stopped because the label mismatches of e violate the relaxation bound c in $[m,$ checkT$[e, L^m(e)]]$, or checkT$[e, L^m(e)] = T$. We can obtain the R edge set directly, along the same lines as Case 2. (2) If checkT$[e, L^m(e)] < m$, we need to obtain the R edge set by scanning DEL-Table from the timestamp m, along the same lines as Case 1.

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 maintains arrays maxIntv, checkT and vioT (lines 3-4). For Case 2, it updates array vioT, and obtains the R edge set for e directly by arrays maxIntv and vioT (lines 5-7). For Case 3, it updates array vioT (line 8), and decides how to obtain the R edge set. If checkT$[m, L^m(e)] < m$, it executes along the same line as Case 1 (lines 9-10). Otherwise, it executes 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), the threshold $k = 5$, and the 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 $G_s($S$^*[m, i])$. By Propositions 2 & 3 in Section 3.1, we need to maintain connected components (or simply ccs) of the subgraph $G_s($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 achieve these efficiently, we dynamically maintain the cc set CC$[i, T]$ for $G_s($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, then 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.

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 $\emptyset$ (line 1). For convenience, we use CC$[T + 1, T]$ and checkCC$[T + 1, T]$ to denote $\emptyset$. 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

```
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 := $\emptyset$;
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 $G_s \notin$ 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 $e \in G_s$ if changes;
14.   let $E_{rm}$ := edges $e \in G_s$, which need to be removed;
15.   if $E_{rm} = \emptyset$ then
16.     Move $G_s$ from checkCC[$i, T$] to maxCC;
17.   else Add each recomputed cc in $G_s \setminus E_{rm}$ with ccScope to maxCC;
18. return maxCC, CC[$i, T$], checkCC[$i, T$], and the updated array scope.
```

Figure 6: Procedure maxCheck

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 then 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 each edge $e \in G_s$, and identifies the set E_{rm} of edges to be removed by G_s.vioTS (lines 12-14). If E_{rm} is $\emptyset$, 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 $G_s \setminus E_{rm}$ with ccScope 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).

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 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 R[m, i], the cc set maxCC, the relaxation bound δ, DEL-Table,
and the interval [m, i].
Output: the set 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 e in $G_s$, e ∈ R[m, i] and l = m then
5.     Add the non-expandable $G_s$ to TF[m, i];
6.   else Check if each edge e ∈ $G_s$ satisfies $L^n(e) = L^h(e)$ and the mismatches
       of e satisfy the bound δ in any intervals [n, h] containing [m, i];
7.     Add $G_s$ to TF[m, i] if it 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 the intervals within the intersection of the interval scope$[e]$ for each edge $e \in E_s$, which is dynamically maintained in procedure maxCheck, *i.e.,* G_s.ccScope.

Assume that G_s.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 conditions by verifying DEL-Table for edge e whether the conditions $lab_n = lab_h$ and $dif_h - dif_n \leq \delta(h - n + 1)$ both hold.

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 be $\emptyset$ (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 if there is an interval $[n, h]$ ($n \geq l$ and $h \leq r$) containing $[m, i]$ 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 $[n, h]$. 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}.ccScope $= [1, 7]$ and G_{s5}.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 $O(\max I \cdot |E_{m,T}|)$ 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, $\max I$ is the maximum number of intervals to be checked when determining whether a 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 $\emptyset$, 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 cc sets $\mathsf{CC}[i, T]$ and $\mathsf{checkCC}[i, T]$ for $i \in [m + k - 1, T + 1]$ to $\emptyset$. Note that we use $\mathsf{CC}[T + 1, T]$ and $\mathsf{checkCC}[T + 1, T]$ to denote $\emptyset$ for convenience. (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). 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 $O(\max I \cdot T^2 \max E_{m,T} + T^2|E|) = O(\max I \cdot T^2|E|)$ 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

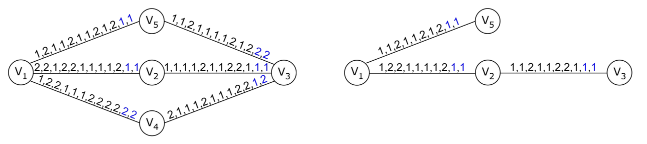
In this section, we introduce two optimization strategies.

Strategy 1: common edge identifying. In algorithm FRTM, procedure maxCheck generates maximal but expandable RTMs for interval $[m, i]$ if the mismatches of their 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: *For each* cc *in* $G_s(\mathsf{S}^*[m, i])$, *if each edge* e *in the* cc *satisfies* $L^{m-1}(e) = L^m(e)$, *any edges in the* cc *cannot form a maximal and non-expandable* RTM *with the 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 edgeFilter (lines 10-11 in Figure 5). (2) We can initialize a mark (set to '*True*') for each generated cc in maxCheck, and change the mark to '*False*' if it contains an edge



(a) Updated temporal network G' with the interval $[1, 12]$

(b) A RTM H_3 **of** G' **with the interval** $[3, 12]$

Figure 8: Running example for the incremental algorithm

$e \in \mathsf{R}^+[m]$. 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 new ccs in maxCheck (line 17 in Figure 6), we do not add the ccs that contain no 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 any edge 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 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 into 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*), along with 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* RTM*s*). 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* RTM*s for $G(V, E, 1, T, L)$, and two intermediate result sets* EIntR *and* MIntR *for G, there is an algorithm that finds the set* TF$^+$ *of updated maximal and non-expandable* RTM*s in $O(\max I \cdot (T\Delta T|E_{\mathsf{EIntR}}| + (\Delta T)^2|E| + |E_{\mathsf{MIntR}}|))$ 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 affected edges easily when computing R edge sets for G (we check whether the label mismatches of edge e violate the relaxation bound c in Section 3.3).

Proposition 10: *Unaffected* RTM*s: if any edge of a* 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 a* RTM *in set* TF *are affected, the* RTM *is affected,* i.e., *might be expandable for G'.* □

Proposition 11 reveals that there exist RTMs that might be expandable, and that require checking for the expandable property.

We next show how to obtain affected edges and affected RTMs in algorithm FRTM. Assume that the algorithm executes for intervals $[m, m + k - 1], \cdots, [m, T]$. We denote the sets EIntR and MIntR obtained for these intervals as EIntR$[m]$ and 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 edge e with the label $L^m(e)$ and the arrays maxIntv$[e, L^m(e)]$, vioT$[e, L^m(e)]$ and checkT$[e, L^m(e)]$ into EIntR$[m]$, when edge e with its label $L^m(e)$ is not in EIntR. Moreover, we store each non-expandable affected RTM into MIntR$[m]$ in procedure expCheck, after verifying that all of its edges 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 $e \in$ EIntR$[m]$. After that, it invokes the procedure edgeFilter, but only considers the edges in $\bigcup_{n \leq m}$ EIntR$[n]$ instead of E when computing R edge sets, by Proposition 9. Specifically, for each edge $e \in$ EIntR$[m]$, procedure edgeFilter obtains the R set for edge e by scanning DEL-Table from the timestamp $T + 1$, while for each edge $e \in \bigcup_{n < m}$ EIntR$[n]$, it executes 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 executes 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 $\bigcup_{1\le n\le 3}\mathsf{EIntR}[n]$, 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 $O(\max I\cdot(T\Delta T|E_{\mathsf{EIntR}}|+(\Delta T)^2|E|+|E_{\mathsf{MIntR}}|))$ time, while static algorithm FRTM takes $O(\max I \cdot (T+\Delta T)^2|E|)$ 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 is 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, as 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 $|\bigcup_{m+k-1\le h\le T+\Delta T}\mathsf{R}[m,h]|$ 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 gathered three datasets, as shown in Table 1.

(1) BJData [43] is a real-life dataset that records three road traffic conditions (*i.e.,* congested, slow and fast) in Beijing. We extracted the day-level data (November 4th, 2010) with 5 minutes a snapshot.

(2) EURData [22] 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 [13].

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. All algorithms were implemented with C++, and tested by function GetProcessMemoryInfo for the memory cost. Experiments were conducted 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 algorithm FRTM, FRTM(OPT1) and FRTM^+ *w.r.t.* the relaxation bound δ, the relaxation bound c, the frequency threshold k, the number $|E|$ of edges, and the number T of snapshots.

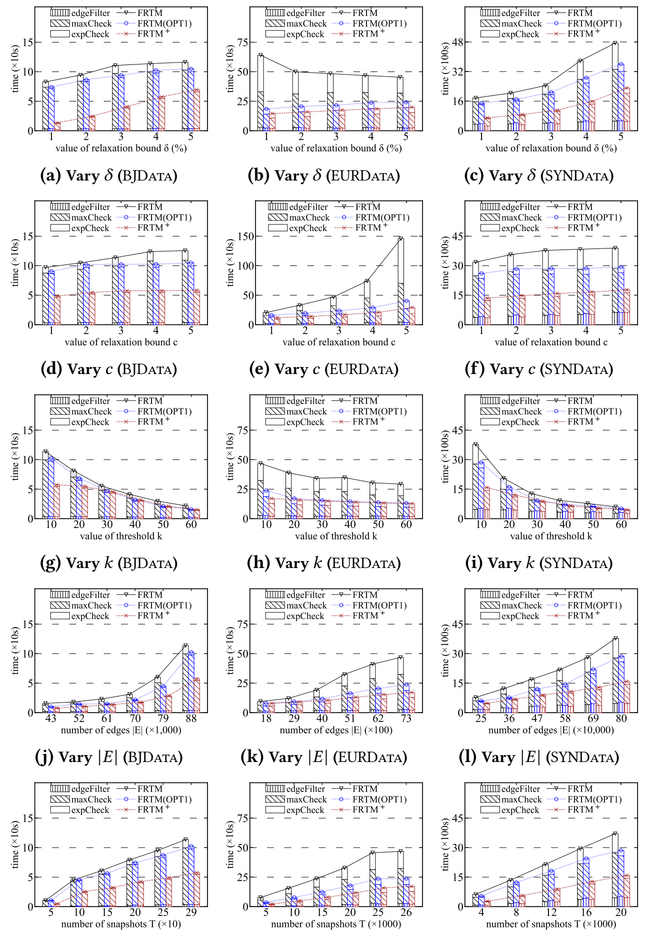
Exp-1.1. To evaluate the impacts of the bound δ, we varied δ from 1% to 5%, fixed $c = 3$ and $k = 10$, and used the entire networks for all the datasets. The results are reported in Figures 9(a)-9(c).

When varying the bound δ, the running time of three algorithms all increases with the increment of δ, except for FRTM on EURData. The reason is that the number of checked intervals in expCheck decreases for larger δ on EURData, which has more impact 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 three procedures all 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 three algorithms all increases with the increment of c, as there 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.



(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)**

(j) Vary $|E|$ (BJDATA) **(k) Vary $|E|$ (EURDATA)** **(l) Vary $|E|$ (SYNDATA)**

(m) Vary T (BJDATA) **(n) Vary T (EURDATA)** **(o) Vary T (SYNDATA)**

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

When varying the bound c, the running time of three procedures all increases with the increment of c. Procedure maxCheck 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 60, 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 three algorithms all 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 (1.63, 2.60, 1.62) and (1.11, 1.03, 1.11) times faster than FRTM and FRTM(OPT1) on (BJDATA, EURDATA, SYNDATA), respectively. Note that the strategy of short interval handling is invalid for $k \geq 30$ in FRTM$^+$.

When varying the threshold k, the running time of three procedures all decreases with the increment of k. Procedure maxCheck takes the most time, except for large k on SYNDATA (FRTM for $k = 60$, and FRTM(OPT1) and FRTM$^+$ for $k \geq 40$), in which case procedure edgeFilter takes the most time. The reason is that, fewer edges in SYNDATA meet bounds δ and c in large intervals. Procedure maxCheck is on average (78%, 57%, 43%) for FRTM, (87%, 60%, 48%) for FRTM(OPT1) and (86%, 58%, 46%) for FRTM$^+$ on (BJDATA, EURDATA, SYNDATA), respectively.

Exp-1.4. 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 $\delta = 4\%$, $c = 3$ and $k = 10$ and randomly selected edges by fixing $|V| = (69,416, 2,707, 400K)$ for (BJDATA, EURDATA, SYNDATA), respectively. The results are reported in Figures 9(j)-9(l).

When varying the number $|E|$ of edges, the running time of three algorithms all 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 three procedures all 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-1.5. To evaluate the impacts of the number T of snapshots, 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 9(m)-9(o).

When varying the number T of snapshots, the running time of three algorithms all 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 number T of snapshots, the running time of three procedures all 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-1.6. To evaluate the space cost, we tested the memory cost of static algorithms in practice, along the same setting as Exp-1.3, while fixed $k = 10$. The results are reported in Table 2.

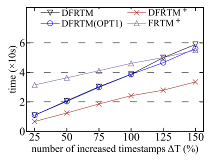
Table 2: Memory cost of static algorithms

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

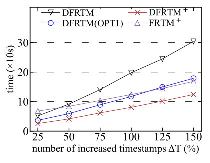
Algorithm FRTM(OPT1) uses (0.77%, 0.03%, 0.15%) more memory than FRTM on all the datasets for more space of the set R$^+$ and the marks for connected components, while FRTM$^+$ uses (1.66%, 1.46%, 1.16%) less memory than FRTM on (BJDATA, EURDATA, SYNDATA), respectively, for less space of connected components.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

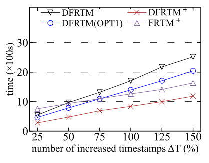
Exp-2.1. To evaluate the impacts of the number ΔT of increased timestamps, we varied ΔT from 25% to 150%, while fixed $\delta = 4\%$,



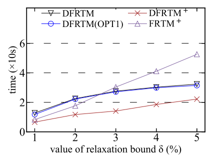
(a) Vary ΔT (BJDATA)



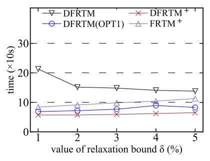
(b) Vary ΔT (EURDATA)



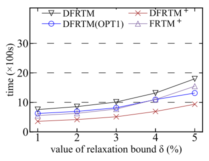
(c) Vary ΔT (SYNDATA)



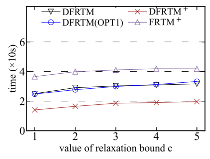
(d) Vary δ (BJDATA)



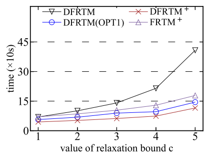
(e) Vary δ (EURDATA)



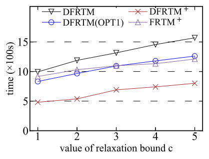
(f) Vary δ (SYNDATA)



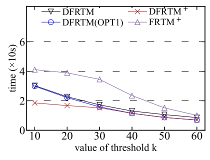
(g) Vary c (BJDATA)



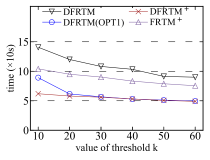
(h) Vary c (EURDATA)



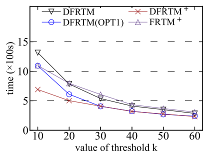
(i) Vary c (SYNDATA)



(j) Vary k (BJDATA)



(k) Vary k (EURDATA)



(l) Vary k (SYNDATA)

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

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

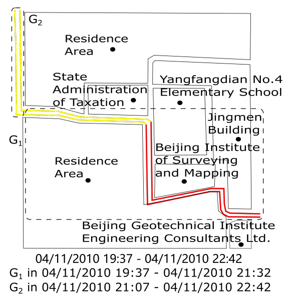
c = 3, k = 10, and the original timestamps T = (112, 10400, 800) for (BJDATA, EURDATA, SYNDATA), respectively. The results are reported in Figures 10(a)-10(c).

When varying the number of ΔT, the running time of all the 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.

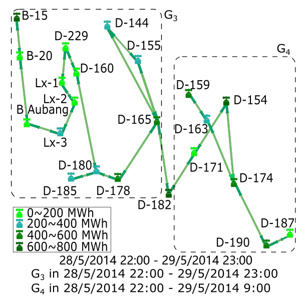
Exp-2.2. To evaluate the impacts of the bound δ, we varied δ from 1% to 5%, while fixed the same settings of T, c and k as Exp-2.1 and ΔT = 75%. The results are reported in Figures 10(d)-10(f).

When varying the bound δ, the running time of all the algorithms increases with the increment of δ, except for DFRTM on EURDATA, along the same reason as 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 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 Exp-2.1 and ΔT = 75%. The results are reported in Figures 10(d)-10(f).


G₂
Residence Area
State Administration of Taxation
Yangfangdian No.4 Elementary School
Jingmen Building
Beijing Institute of Surveying and Mapping
G₁
Residence Area
Beijing Geotechnical Institute Engineering Consultants Ltd.
04/11/2010 19:37 - 04/11/2010 22:42
G₁ in 04/11/2010 19:37 - 04/11/2010 21:32
G₂ in 04/11/2010 21:07 - 04/11/2010 22:42

(a) A RTM on BJDATA



(b) A RTM on EURDATA

Figure 11: Case studies on real-life datasets

When varying the bound c, the running time of all the algorithms increases with the increment of c, along the same reason as 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 60, while fixed the same settings of T, δ and c as Exp-2.1 and ΔT = 75%. The results are reported in Figures 10(j)-10(l).

When varying the threshold k, the running time of all the algorithms decreases with the increment of k, along the same reason as Exp-1.3. Moreover, incremental algorithm DFRTM$^+$ is the best for $k \leq 20$, and similar to the algorithm DFRTM(OPT1) for $k \geq 30$, as the strategy of short interval handling is invalid for $k \geq 30$. It is on average (2.82, 1.82, 2.01), (1.63, 2.17, 1.57) and (1.10, 1.04, 1.08) 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, along the same settings of T, δ, c and k as Exp-2.1, while fixed ΔT = 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, along the same reason as 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 sets EIntR and MIntR on EURDATA, but use less memory on BJDATA and SYNDATA.

Exp-3. 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 δ = 4%, c = 3 and k = 10.

Case 1. For BJDATA, we visualized our RTMs by the traffic data management system from [20] and found corresponding roads in Google Maps. We use red and yellow colors to represent roads with 'congested' and 'slow' traffic condition labels, respectively. We used our RTMs to discover long-time congestion patterns, as shown in Figure 11(a). (1) The RTM corresponds to the traffic conditions in the *Yangfangdian community* during [19:37, 22:42]. It indicates the congestion may be caused by roads shown in red colors, which are narrow side streets in practice. (2) We also show the result with exact label matches (*i.e.*, δ = 0 or c = 0), including two RTMs G_1 and G_2 (circled by dashed lines). G_1 and G_2 should be merged into a large area, otherwise we cannot find the bottleneck roads for G_2.

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 demand signals (600 ∼ 800, 400 ∼ 600, 200 ∼ 400, 0 ∼ 200) MWh, respectively. We use our RTMs to discover hybrid energy consumption patterns, as shown in Figure 11(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. The merging point named '$D-182$' with high energy demand may need more transmission lines. (2) We also show the result with exact label matches (*i.e.,* $\delta = 0$ or $c = 0$), including two RTMs G_3 and G_4 (circled by dashed lines). G_3 and G_4 should be merged into a large area, we cannot maintain good load balance in energy transmission of G_3 and G_4.
Existing temporal motifs. To the best of our knowledge, existing temporal motifs are unsuitable for finding these patterns. (1) Existing studies on temporal motifs mostly impose restrictive constraints [3, 8, 11, 18, 21, 24–29, 38, 48, 49, 65]. Among these is [3], which imposes strict unchanged label constraint that is somewhat similar to our RTMs with $\delta = 0$ or $c = 0$. Hence, further comparison with this study is unnecessary. (2) The studies on temporal motifs in [33, 57] relax the topology constraints but cannot alleviate the data quality issues to tolerate label mismatches.
Summary. We have 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 (2.70, 1.88, 2.04) times faster on datasets (BJData, EURData, SYNData), respectively. (3) Our optimization strategies are effective, as FRTM$^+$ and DFRTM$^+$ are on average (2.08, 2.75, 2.00) and (1.79, 2.56, 1.84) times faster than baselines FRTM and DFRTM on datasets (BJData, EURData, SYNData), respectively. (4) Our incremental algorithm DFRTM$^+$ use at most 0.60% more memory than its static counterpart with $\Delta T = 75\%$. (5) 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 [44, 56], which have widespread applications in both industry and academia [5, 7, 12, 15, 32, 34, 40, 44, 47, 56, 58, 62] (see, *e.g.,* [52, 63] 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 [12, 15, 47]. 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.,* [37] for surveys). Most motifs are defined as isomorphic subgraphs, where the edges are attached with timestamps satisfying partial time orders [18, 26, 27, 65] or total time orders [21, 25, 48], as well as local time windows [18, 21, 26, 27, 38, 65] or global time windows [25, 38, 48]. 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 and global time windows refer to the time bounds on adjacent edges and on all the edges, respectively. There are other motifs defined as induced subgraphs, where node weights evolving in a consistent trend [11, 24], and edge labels and directions keeping unchanged in a period [3]. To achieve speed up, there are studies exploring sampling [36, 53, 54, 59], parallel approaches [17], and approaches for motifs with specific topologies [8, 28, 29, 49]. There are also studies relaxing the topology constraints, *e.g.,* [57] proposes temporal motifs based on dual simulation [41] instead of subgraph isomorphism, while [33] 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 the data quality issues. Instead, we propose a concept of temporal motifs, which can be computed in low polynomial time, and allows label mismatches to handle data quality issues. We further develop incremental methods to handle with the dynamic nature of temporal networks. These have not been considered by existing studies.
Patterns with constraint relaxations. There are several studies on patterns with constraint relaxations. [35, 39, 51, 61] relax the support constraint in frequent itemset mining, allowing for missed transactions or items. [4, 16, 23, 64] relax the subgraph isomorphism constraint in frequent subgraph mining, allowing for label or topology mismatches. [31] relax the label constraint in graph simulation with taxonomy, allowing for more label matching relations. Different from these studies, we focus on alleviating the data quality issues 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 [55], graph association rules [14], graph temporal association rules [45], dense temporal graphs [6, 42, 43], frequent graph patterns [9] and frequent patterns[1].

We essentially follow the concepts of temporal networks from [42, 43], but use edge labels instead of edge weights, to represent the dynamics of temporal networks. Moreover, our maximal and non-expandable properties are related to maximal or closed frequent patterns [1, 9, 45], 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 the data quality issues on a class of temporal networks, where the nodes and edges are fixed, but the edge labels vary regularly with timestamps. 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 (Eds.). 2014. *Frequent Pattern Mining*. Springer, Cham.

[2] Charu C Aggarwal, Haixun Wang, et al. 2010. *Managing and mining graph data*. Vol. 40. 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*. ACM, 518–526.

[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. *PVLDB* 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. https://doi.org/10.14778/3636218.3636223

[9] Hong Cheng, Xifeng Yan, and Jiawei Han. 2014. Mining Graph Patterns. In *Frequent Pattern Mining*, Charu C. Aggarwal and Jiawei Han (Eds.). Springer, Cham, 307–338.

[10] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. ACM, 315–326.

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

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

[13] 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.

[14] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.

[15] 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.

[16] 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. *Knowl. Inf. Syst.* 44, 2 (2015), 385–406.

[17] Zhongqiang Gao, Chuanqi Cheng, Yanwei Yu, Lei Cao, Chao Huang, and Junyu Dong. 2022. Scalable Motif Counting for Large-scale Temporal Graphs. In *ICDE*. https://doi.org/10.1109/ICDE53745.2022.00244

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

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

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

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

[22] 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.

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

[30] 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.

[31] Jia Li, Yang Cao, and Shuai Ma. 2017. Relaxing Graph Pattern Matching With Explanations. In *CIKM*. ACM, 1677–1686.

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

[33] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent Community Search in Temporal Networks. In *ICDE*. https://doi.org/10.1109/ICDE.2018.00077

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

[35] 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*. 407–418.

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

[37] Penghang Liu, Valerio Guarrasi, and A Erdem Sariyuce. 2023. Temporal network motifs: Models, limitations, evaluation. IEEE. *TKDE* 35, 1 (2023), 945–957. https://doi.org/10.1109/TKDE.2021.3077495

[38] 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.

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

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

[41] 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.

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

[43] 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.

[44] 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.

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

[46] 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.

[47] 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.

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

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

[50] 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.

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

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

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

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

[55] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2019. Finding lasting dense subgraphs. *Data Mining and Knowledge Discovery* 33, 5 (2019), 1417–1445.

[56] 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.

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

[58] 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.

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

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

[61] Cheng Yang, Usama M. Fayyad, and Paul S. Bradley. 2001. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD*, Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant (Eds.). ACM, 194–203. https://doi.org/10.1145/502512.502539

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

[63] Shuo Yu, Yufan Feng, Da Zhang, Hayat Dino Bedru, Bo Xu, and Feng Xia. 2020. Motif discovery in networks: A survey. *Computer Science Review* 37 (2020), 100267.

[64] Shijie Zhang, Jiong Yang, and VenuMadhav Cheedella. 2007. Monkey: Approximate Graph Mining Based on Spanning Trees. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.). IEEE Computer Society, 1247–1249.

[65] 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*.

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

Supplementary Material for 'Mining Relaxed Temporal Network Motifs'

Hanqing Chen Shuai Ma
NLSDE Lab, Beihang University
{chenhanqing,mashuai}@buaa.edu.cn

A CORRECTNESS 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, and 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, 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 a 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 [1]. 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 cannot 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 a 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 $e \in \mathsf{R}[p_1, j_1], \ldots, \mathsf{R}[p_x, j_x]$ ($p_1, \ldots, p_x \in [1, m]$ 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 $[p_1, h], \ldots, [p_x, h]$, and further 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) Similarly, by the definition of scope and the R set, we can verify that there are no generated maximal RTMs with interval $[m, i]$ that can appear in RTMs with intervals $[n, h]$ for $1 \leq n \leq l-1$ and $i \leq h \leq r$.

Putting these together, and we have the conclusion.

B CORRECTNESS 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, 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, 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, and we have the conclusion.

C CORRECTNESS PROOFS IN OPTIMIZATION STRATEGIES

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

(1) There exists some edge $e \notin \mathsf{S}[m, i]$ in G_s. By Proposition 3, we know that the subgraph can not correspond to a maximal RTM with interval $[m, i]$.

(2) All the edges e in G_s belong to $\mathsf{S}[m, i]$. As $L^{m-1}(e) = L^m(e)$, all the edges $e \in \mathsf{S}[m-1, i]$. That is, G_s corresponds to an expandable RTM which can appear in the RTM with interval $[m-1, i]$.

Putting these together, and 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 $\mathsf{S}^*[m, i]$ ($m+k-1 \leq i \leq m+\max L-1$), $L^m(e) = L^i(e)$ and label mismatches of e satisfy the relaxation bounds δ and c, *i.e.,* $e \in \mathsf{S}[m, i]$. Hence, $\mathsf{S}^*[m, i] \subseteq \mathsf{S}[m, i]$. 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.

Table 1: Parameters of static algorithms

Datasets	Average $\lvert E_{m,T}\rvert$	max$E_{m,T}$	Average $\lvert I\rvert$			max$I$		
			FRTM	FRTM(OPT1)	FRTM$^+$	FRTM	FRTM(OPT1)	FRTM$^+$
BJData ($\lvert E\rvert = 88,396$)	57,676	75,146	34.67	1.52	1.14	19,596	19,321	19,043
EURData ($\lvert E\rvert = 7,334$)	5,048	6,695	203,739	791.42	475.54	1.73×10^8	1.40×10^8	1.40×10^8
SYNData ($\lvert E\rvert = 800$K)	218,954	301,473	415.82	21.67	10.20	9.91×10^5	9.90×10^5	9.89×10^5

D CORRECTNESS PROOFS IN THE INCREMENTAL ALGORITHM

Proof of Proposition 9. Assume that label mismatches of an edge e satisfy the relaxation bound c in the interval $[m, T]$. Let N be the number of label mismatches in the intervals $[m, T]$. There exists timestamp $t \in [T+1, T+\Delta T]$ that label mismatches of edge e satisfy relaxation bounds δ and c in the interval $[m, t]$, as long as ΔT is large enough, labels of e in $[T+1, T+\Delta T]$ are the same as $L^m(e)$, and $N \leq \delta(t - m + 1)$.

Proof of Proposition 10. Assume that an edge e in a RTM is not an affected edge, *i.e.,* label mismatches of e violate the bound c in interval $[m, T]$ ($m \in [1, T-k+1]$). It is obvious that label mismatches of e violate the bound c in intervals $[n, h]$ for $n < m$ and $h \in [T+1, T+\Delta T]$. Hence, any RTMs containing the edge e in TF can not appear in these intervals, *i.e.,* they are non-expandable.

Proof of Proposition 11. Assume that all the edges e of is a RTM are affected, *i.e.,* they might belong to R$[m, i]$ for $m \in [1, T-k+1]$ and $i \in [T+1, T+\Delta T]$. That is, they might belong to S$^*[m, j]$ for certain $j \in [T+1, T+\Delta T]$. As long as label mismatches of e satisfy the relaxation bound δ and $L^m(e) = L^j(e)$ holds, $e \in$ S$[m, j]$, *i.e.,* the RTM consisting of these edges is expandable.

E DETAILED COMPLEXITY ANALYSES

(1) Time complexity of the procedure edgeFilter. The process of identifying 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 need to be updated). Hence, 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 [2, 3], we need to know the connectivity among all the edges instead of two nodes. Hence, the process of computing connected components needs $O(E_s)$ time to identify the connected components to which each edge belongs, which has the same time complexity as computing connected components from scratch by utilizing disjoint sets [4]. Let $|E_{m,T}|$ be the number of edges in S$^*[m, m+k-1]$. For one interval, the number of edges in all the connected components is at most $|E_{m,T}|$. Hence, procedure maxCheck takes $O(|E_{m,T}|)$ worst time.

(3) Time complexity of the procedure expCheck. Let max$I$ be the maximum number of intervals to be checked when checking the expandable property of a RTM. For one interval, the number of edges in all the maximal RTMs to be checked is also at most $|E_{m,T}|$. Hence, procedure expCheck takes $O(\max I \cdot |E_{m,T}|)$ worst time.

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

For each m, (1) the procedure edgeFilter takes $O(T|E|)$ time, (2) the procedure maxCheck takes $O(T|E_{m,T}|)$ time, and (3) the procedure expCheck takes $O(\max I \cdot T|E_{m,T}|)$ time. There are in total $O(T-k+1) = O(T)$ execution times. Hence, algorithm FRTM takes $O(\max I \cdot T^2 \max E_{m,T} + T^2|E|) = O(\max I \cdot T^2|E|)$ 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 for all the edges.

(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 CC$[i, T]$ and checkCC$[i, T]$ ($i \in [m+k-1, T]$).

(e) For the set TF$^+$ of maximal and non-expandable RTMs, it cost $O(T^2 \max E_{m,T})$ space.

From the above mentioned, 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. (I) For each $m \in [1, T+k-1]$, procedures edgeFilter and maxCheck take $O(\Delta T|E_{\mathsf{EIntR}}|)$ time, and procedure expCheck takes $O(\max I \cdot \Delta T|E_{\mathsf{EIntR}}|)$ time. The process of checking RTMs in sets MIntR whether they are expandable or not takes $O(\max I \cdot |E_{\mathsf{MIntR}}|)$ time for all $m \in [1, T+k-1]$. Hence, this part in total takes $O(\max I \cdot T\Delta T|E_{\mathsf{EIntR}}| + \max I \cdot |E_{\mathsf{MIntR}}|)$ time. (II) For each $m \geq T-k+2$, the time complexity is the same as the static algorithm. Hence, this part in total takes $O(\max I \cdot (\Delta T)^2|E|)$ time.

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

F EXTRA EXPERIMENTAL TESTS

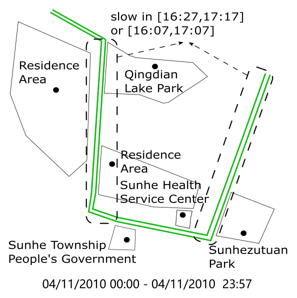
Exp-1.7. To evaluate the parameters used in the time complexity analysis, we tested the maximum and average value of $|E_{m,T}|$ and $|I|$, fixed $\delta = 4\%$, $c = 3$ and $k = 10$, and used the entire temporal networks for all the datasets. The results are reported in Table 1. Note that $|E_{m,T}|$ 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 a RTM; max$E_{m,T}$ and max$I$ are the largest $|E_{m,T}|$ and $|I|$, respectively.

$|E_{m,T}|$ is less than $|E|$ to various 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 max$I$ vary significantly in different datasets for the same reason. The average value of $|I|$ 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.

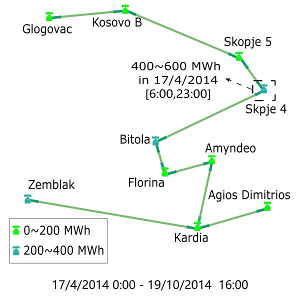
Exp-2.6. To evaluate the space cost and the parameters used in the time complexity analysis, we tested the value of max$E_{m,(T+\Delta T)}$, the

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

Datasets	$\max E_{m,(T+\Delta T)}$	$\lvert E_{\mathsf{EIntR}}\rvert$	$\lvert E_{\mathsf{MIntR}}\rvert$	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) RTM **with no settings of the relaxation bound** c **on** BJDATA



(b) RTM **with no settings of the relaxation bound** c **on** EURDATA

Figure 1: Extra case studies on real-life datasets

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 δ = 4%, c = 3, k = 10, ΔT = 75% and the original timestamps T = (112, 10, 400, 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 $\bigcup_{m+k-1\le h\le T+\Delta T}\mathsf{R}[m,h]$ for each m, sets EIntR and MIntR are necessary intermediate results for incremental algorithms, and E_{EIntR} and E_{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 far 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-3. We conducted case studies on BJDATA and EURDATA by the result of FRTM with δ = 4% and k = 10, *i.e.,* no settings of c.

Case 1. For BJDATA, we use green color to represent roads with 'fast' traffic condition labels. As Figure 1(a) depicts, we identify the RTM corresponding to the roads in the *Sunhe community*. However, this pattern essentially 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] (circled by dashed lines).

Case 2. For EURDATA, we use lime green and light 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 1(b) depicts, we identify the RTM corresponding to the area in Italy. However, this pattern essentially 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] (circled by dashed lines).

This verifies the effectiveness of the local relaxation bound c.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, Mass: MIT press, 2009.

[2] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *SODA*, S. Khanna, Ed. SIAM, 2013, pp. 1131–1142.

[3] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *JACM*, vol. 48, no. 4, pp. 723–760, 2001.

[4] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *JACM*, vol. 22, no. 2, pp. 215–225, 1975.