Discovering Graph Temporal Association Rules

Mohammad Hossein Namaki*, Yinghui Wu*§, Qi Song*, Peng Lin*, Tingjian Ge†
Washington State University* Pacific Northwest National Laboratory§ University of Massachusetts, Lowell†
{mnamaki, yinghui, qsong, plin1}@eecs.wsu.edu* ge@cs.uml.edu†

ABSTRACT

Detecting regularities between complex events in temporal graphs is critical for emerging applications. This paper proposes graph temporal association rules (GTAR). A GTAR extends traditional association rules to discover temporal associations for complex events captured by a class of temporal pattern queries. We introduce notions of support and confidence for GTARs, and formalize the discovery problem for GTARs. We show that despite the enhanced expressive power, GTARs discovery is feasible over large temporal graphs. We develop an effective rule discovery algorithm, which integrates event mining and rule discovery as a single process, and reduces the redundant computation by leveraging their interaction. Using real-life and synthetic data, we experimentally verify the effectiveness and scalability of the algorithms. Our case study also verifies that GTARs demonstrate highly interpretable associations in real-world networks.

1 INTRODUCTION

Temporal association rules have been studied to capture the temporal regularities between item sets in transaction data [5, 11]. The temporal regularities are typically captured as association rules (ARs) [4] that hold on transactions at specified times [5, 11]. These rules state "if items X exist in transactions at a specified time, then items Y are likely to exist in the same set of transactions".

Nonetheless, temporal regularities among network events are more involved than their counterpart over item sets. (1) Network events are represented by *graph patterns*, rather than a set of items [21, 35]. (2) It is more desirable to discover temporal associations within a time window between two events P_1 and P_2 rather than associations at specific time stamps. Such rules state "*if event P_1 occurs, then event P_2 is likely to occur in Δt time.*" The need for discovering such rules is evident in social analysis [37, 38], data quality [2], software analysis [24], among others.

Example 1.1. Consider the following scenarios.

Masked Attack detection. Recent cyber security reports [1, 26] suggest that Denial of Service (DDoS) attacks coincide with "more than 45% (resp. 32%) of malware incidents (resp. network intrusions)", which are frequently used as a decoy to distract defense effort and mask the true intrusion, *e.g.,* information breaches [1]. Such masked attack can be described by a temporal association R_1 between two attack events P_1 and P_2 (as illustrated in Fig. 1), where (1) P_1 is a DDoS attack, where the attacker u controls multiple bots to create massive requests to target $\bar{u}$, and (2) P_2 describes an information exfiltration event, where a server $\bar{u}$ takes commands from a bot and browses compromised websites that lead to data breach.

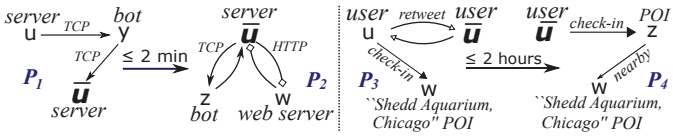


Figure 1: Temporal association rules in graphs

The rule states that *"if a host $\bar{u}$ is involved in a DDoS attack (P_1) at some time, then this host is likely to be the victim of an information exfiltration (attack P_2) within 2 minutes".*

The above rule is a departure from our familiar ARs. (1) It specifies the antecedent and consequent with graph patterns P_1 and P_2, which pose topological constraints over entities (*e.g.,* hosts, bots and Web servers). (2) It specifies a time window ($\leq$ 2 min) to describe the maximum gap between the occurrences of two events. (3) A "focus" $\bar{u}$ is designated to indicate that both events occur at the same host. Conventional (temporal) ARs are not capable to represent such regularities.

Time-aware POI recommendation. Social network users tend to behave in a short period triggered by social influences [37]. For example, a temporal social influence can be represented by two graph patterns P_3 and P_4 that capture social communities and a recommendation pattern, respectively. The rule states that *"if a user $\bar{u}$ has a friend u and they retweet each other, and u checks in at a point of interests (POI) w* (e.g., *"Shedd Aquarium, Chicago")* *at some time t (via* e.g., *Facebook Place), then it is likely he will visit a nearby POI z* (e.g., *"Field Museum, Chicago") in 2 hours.*

Such rules can be used for time-aware POI recommendation that holds for specific time and places [37], with the additional expressive power to capture the context of social structures (*e.g.,* Twitter communities). For example, z in P_2 can be recommended as a POI for $\bar{u}$, and $\bar{u}$ can be identified as a potential customer of z.

Activity prediction. Temporal associations can also help understand and predict personal behavior (*e.g.,* mobile users) [36]. One such rule is *"if a user $\bar{u}$ joins a group call of his co-workers, he usually texts another contact in 5 minutes".* Such rules (not shown) can be represented by a graph pattern P_5 that represents a small call network as a clique, and a single edge P_6 from user $\bar{u}$ to a specific contact, associated with a time window of 5 minutes.

With no clear "transaction" in a dynamic network, conventional ARs can no longer capture such temporal regularities. These call

CIKM'17 , November 6–10, 2017, Singapore, Singapore

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3133014

for ARs that incorporate *topological*, *semantic* and *temporal* constraints to capture the events and their temporal associations.

Contribution. This paper formally introduces association rules that explicitly use temporal graph patterns to capture complex events and their temporal associations.

(1) We introduce *graph temporal association rules* (GTARs, Section 3). A GTAR differs from conventional association rules in both syntax and semantics. It incorporates temporal graph pattern queries as both antecedent and consequent, constrained by a time window that specifies their temporal associations.

To balance its expressive power and computational efficiency, we introduce a class of temporal graph pattern matching to enforce topological, value and temporal conditions of the events.

(2) We formalize the discovery problem for GTARs (Section 4). Based on event matching, we formulate the support and confidence for GTARs. We define the support of GTARs in terms of "minimal occurrences" of associated event pairs, by revising a measure used by its counterpart in item streams [6]. We extend conventional confidence measures in terms of minimal occurrences of GTARs. We show that the support of GTARs is anti-monotonic, and the confidence of GTARs is "conditionally" anti-monotonic.

Based on the support and confidence, we formalize a parameterized discovery problem to strike a balance between the enhanced expressiveness of GTARs and the cost of GTAR discovery.

(3) We develop a feasible algorithm to discover GTARs (Section 4). In contrast to prior algorithms, it integrates event mining and association rule discovery in a single process. The algorithm interleaves event matching and rule validation to enable the "anytime" performance [7], *i.e.,* the discovery process can be interrupted to obtain high quality rules. We also provide effective pruning strategies to reduce the redundant computation as early as possible.

(4) Using real-world and synthetic datasets, we experimentally verify the effectiveness of GTARs, and the efficiency of GTAR discovery algorithms. We find that it is feasible to mine GTARs from large temporal graphs, and converges fast to GTARs with desirable quality. For example, it takes 22 seconds to detect GTARs in a citation network with $26M$ nodes and edges and covers 80 timestamps, and 18 seconds to find GTARs with an accuracy of 90%. Moreover, our optimization strategies improve the GTAR mining by 6.28 times over real-world graphs. We also find that GTARs capture meaningful temporal correlations that can be used for activity prediction.

Related work. We categorize the related work as follows.

Temporal association rules. Association rules (ARs) and their extensions over temporal dimension have been studied for transactional databases [4]. The usual object is to discover regularities in the occurrence of item sets and their temporal relationships [22, 31]. Cyclic ARs (ARs that occur periodically over time) is studied in [28]. Temporal features for association rules (*e.g.,* time intervals the rule holds) are studied in [11], and are used as constraints such as lifespans in the discovery process [5]. A class of patterns are studied in [22], where each composite pattern consists of a bag of events paired with temporal relations (*e.g.,* before, after, during).

Nevertheless, these work study temporal ARs over item sets. In contrast, GTARs extend conventional ARs with temporal graph patterns. It requires the integration of temporal, topological and semantic constraints. The discovery of GTARs is beyond rule mining from temporal transactions.

Graph association rules. ARs extended with graph patterns are studied in recent work [8, 13, 14]. To detect data inconsistency, (conditional) functional dependencies are extended [14] to specify value dependencies on clustered values via path patterns and graph patterns. The dependencies there are hard constraints rather than rules. Specialized for social recommendation, ARs are extended with graph patterns in [13], where the support is captured by subgraph isomorphism over static graphs, and the consequent is a single edge rather than a general graph pattern. None of these work address temporal associations.

Evolution rules [8] detect local changes between a graph pattern and its sub-pattern occurred earlier. We show that these rules are a special case of GTARs, where the consequent is a single edge. We are not aware of temporal ARs for events as general graph patterns.

Temporal graph mining. A number of temporal queries have been studied to detect events over dynamic networks. An event is defined as a match of the query. These queries include subgraph isomorphism [21, 32], dense subgraphs [3], quasi-cliques [34], "heavy subgraph" that maximizes edge weight [9], continuous patterns [16] that incrementally find subgraph matches over evolving graphs, and durable queries [32] that isomorphism matches that last for the longest period of time. In contrast, we strike a balance between the expressiveness and evaluation cost of event model [27] by extending well established graph simulation [35].

Temporal graph mining is studied in [10], where a tree structure is used to enumerate and verify frequent patterns. A frequent graph pattern is a set of edges with valid time intervals that occur more than certain times. Mining algorithms are also introduced to discover communication motifs in dynamic networks [19, 29]. Subsequence mining has been leveraged to identify patterns over sequence representation of temporal graphs [19]. Motifs in temporal networks as induced subgraphs on sequences of temporal ordered edges are discovered over unlabeled networks [29] and fast algorithms for specific 3-nodes 3-edges patterns were proposed to mine topologically frequent motifs. These work do not consider events captured by temporal graph patterns, and use mining models that are very different from GTARs discovery.

2 TEMPORAL GRAPHS AND EVENTS

We start with the event model over temporal graphs.

Temporal graph. A temporal graph $\mathcal{G}_T$ is a tuple (V, E, L, T), where (1) T is a time window (a sequence of consecutive timestamps); (2) V is a set of nodes, $E \subseteq V \times V \times T$ is a set of directed edges associated with a timestamp from T, and (3) function L assigns a label $L(v)$ (resp. $L(e)$) to each node $v \in V$ (resp. edge $e \in E$). An edge e=(u_1, u_2, t) ($t \in T$) encodes a link with label $L(e)$ between u_1 and u_2 that exists at timestamp t.

Given a timestamp $t \in T$, a snapshot G_t of $\mathcal{G}_T$ at t is a graph induced by the set of all the edges associated with timestamp t.

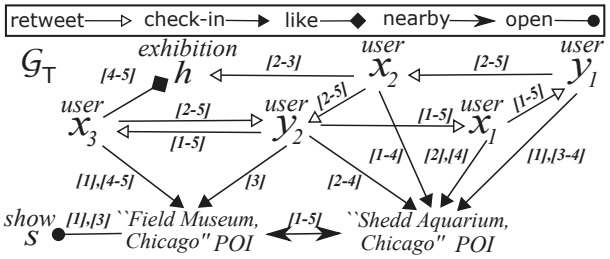


Figure 2: Temporal social graph

Example 2.1. A temporal graph $\mathcal{G}_T$ over 5 hours ($|T|$=5) is shown in Fig. 2. The graph depicts twitter users and their POI recommendation activities. It carries temporal interactions (*e.g.,*retweets and check-ins) associated with timestamps ("packed" as time intervals for the ease of presentation), as well as static locations (*e.g.,*nearby). For instance, user x_1 retweets user y_2 at every timestamp from 1 to 5, and checks in to a POI "Shedd Aquarium Chicago" at time 2 and 4 near another POI "Field Museum, Chicago". A snapshot of $\mathcal{G}_T$, G_3 at time 3 consists of all its edges except for the two edges from user x_3 to "Field Museum, Chicago" and to h, respectively, and the edge from user x_1 to "Shedd Aquarium Chicago".

Events. An *event* $P(\bar{u})$ is a connected graph pattern (V_P, E_P, L_P), in which V_P and E_P are the sets of event nodes and edges, respectively. Each node $v_p \in V_P$ (resp. $e_p \in E_P$) has a label $L_P(v_p)$ (resp. $L_P(e_p)$) specifying a predicate (*e.g.,* "Web servers" of P_2 in Example 1.1). It also carries a designated *focus* node $\bar{u} \in V_P$. The event $P(\bar{u})$ is connected if every pair of nodes in V_P is connected by an undirected path in $P(\bar{u})$.

In practice, an event P may represent *e.g.,* attack events, communication patterns [35], business models [17], among others. Accordingly, the focus $\bar{u}$ may encode the "key player" of user's interest [17, 33], such as attackers, active users, or companies.

How to characterize event occurrences? Strict pattern matching (*e.g.,* subgraph isomorphism) may miss meaningful occurrences and return an excessive number of redundant matches [35]. We thus adopt dual simulation [25], as a well established approximate matching model [12, 35], to event matching over temporal graphs.

Event matching. Given an edge e=$(u, u') \in E_P$, a *candidate function* $f(e)$ specifies a set of *edge candidates* $C(e) \subseteq E$, such that for each candidate e'=$(v, v', t) \in C(e)$, $f(L(v))$=$L_P(u)$, $f(L(v'))$=$L_P(u')$, and $f(L(e'))$=$L_P(e)$. Similarly, given a node $u \in V_P$, $f(e)$ specifies a candidate set $C(u) \subseteq V$ of u, such that for each node $v \in C(u)$, $f(L(v))$=$L_P(u)$.

Given event P and temporal graph $\mathcal{G}_T$, there exists a *event matching* R_t between the snapshot G_t of $\mathcal{G}_T$ and event P if

- for every edge $e = (u, u') \in E_P$, there exists an edge e'=$(v, v', t) \in C(e)$, such that $(e, e') \in R_t$;
- for each pair $(e, e') \in R_t$ and each edge e^p=$(u^p, u) \in E_P$, there exists an edge $e^{p'}$=$(v^p, v, t) \in C(e^p)$ that matches e^p, *i.e.,* $(e^p, e^{p'}) \in R_t$, and
- for each pair $(e, e') \in R_t$ and each edge e^c=$(u', u^{c'}) \in E_P$, there exists an edge $e^{c'}$=$(v', v^{c'}, t) \in C(e^c)$ that matches e^c, *i.e.,* $(e^c, e^{c'}) \in R_t$.

That is, snapshot G_t of $\mathcal{G}_T$ preserves the labels and both parent and child relationship of P in its match induced by R_t [25, 35].

Symbol	Description
$\mathcal{G}_T$	temporal graph with a range of timestamps [1, T]
$\bar{u}$	the focus node of user's interest
$C(e)$ (resp. $C(u)$)	candidate set of event edge e (resp. node u)
$O(\varphi, \mathcal{G}_T)$	minimal occurrences of GTAR φ in $\mathcal{G}_T$
o (P,$\bar{u}$, t)	focus occurrence of $P(\bar{u})$ at time t
φ=$(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$	a GTAR φ pertains to events P_1, P_2 and Δt
Supp(P_1, $\mathcal{G}_T$)	support of event P_1 in $\mathcal{G}_T$ (Section 4.1)
Supp(φ, $\mathcal{G}_T$)	support of GTAR φ in $\mathcal{G}_T$ (Section 4.1)
Conf(φ, $\mathcal{G}_T$)	confidence of GTAR φ over $\mathcal{G}_T$ (Section 4.1)

Table 1: Summary of notations

We use the following notations. (1) We say event P *occurs* in $\mathcal{G}_T$ at time t if there exists a matching R_t. (2) For an edge e=(u, u') and its match e'=(v, v', t) in R_t, the pair (v, t) (resp. (v', t)) is a match of u (resp. u'). Specifically, the *focus occurrence* of P at time t (denoted as o$(P, \bar{u}, t)$) refers to the nodes in V that matches $\bar{u}$ induced by R_t. (3) The matches of edge e (resp. node u) induced by R_T refers to the set of all the matches of e (resp. u) induced by R_t ($t \in T$).

Example 2.2. Event P_3 in Fig. 1 depicts a social event that two users u and $\bar{u}$ (a focus) retweet each other, and u checks in at a POI w. Event P_4 depicts the event that user $\bar{u}$ visits a POI z near w.

For event P_3 and $\mathcal{G}_T$ in Fig. 2, (1) the matches of $\bar{u}$ induced by a matching R_3 in the snapshot G_3 of $\mathcal{G}_T$ contains $\{(x_1, 3), (x_2, 3), (x_3, 3)\}$; (2) the focus occurrence o$(P, \bar{u}, 3)$ of P_3 at time 3 is $\{x_1, x_2, x_3\}$; (3) the matches of the edge $(u, \bar{u})$ in P_3 contains $\{(x_1, y_1, 3), (x_2, y_2, 3), (x_3, y_2, 3)\}$; (4) the matches of $\bar{u}$ in $\mathcal{G}_T$ include $\{(x_1, 3), (x_1, 4), (x_2, 3), (x_2, 4), (x_3, 3)\}$.

Similarly, for event P_4 in Fig. 2, the matches of $\bar{u}$ is $\{(x_1, 2), (x_1, 4), (x_2, 1), (x_2, 2), (x_2, 3), (x_2, 4), (x_3, 1), (x_3, 4), (x_3, 5)\}$.

3 GRAPH TEMPORAL ASSOCIATION RULES

In this section, we introduce graph temporal association rules.

GTARs. A *graph temporal association rule* (GTAR) φ is defined as $(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$, where (1) P_1=$(V_1, E_1, L_1, \bar{u})$ and P_2=$(V_2, E_2, L_2, \bar{u})$ are two events that share a common focus node $\bar{u}$, and (2) Δt is a constant that specifies a time interval. We refer to P_1 and P_2 as the antecedent (or *left-hand side* (LHS) event) and consequent (or *right-hand side* (RHS) event) of φ, respectively.

The GTAR rule states that "*if there exists an occurrence of event P_1 at an entity specified by $\bar{u}$ at some time t, then it is likely that an event P_2 occurs at the same entity, within a time window $[t, t+\Delta t]$*". A GTAR φ incorporates three constraints: (1) topological constraint specified by graph patterns; (2) common "focus" of the antecedent and consequent specified by $\bar{u}$; and (3) a temporal constraint Δt.

Example 3.1. Recall the temporal association for masked attacks in Example 1.1. This can be expressed as a GTAR $\varphi_1 = (P_1 \Rightarrow P_2, \bar{u}, \Delta t = 2min)$ with LHS event (antecedent) P_1 and RHS event (consequent) P_2 given in Fig. 1. Similarly, the temporal association for POI recommendation in Example 1.1 can be expressed as a GTAR $\varphi_2 = (P_3 \Rightarrow P_4, \bar{u}, \Delta t = 2)$ with P_3 and P_4 in Fig. 1.

We consider nontrivial GTARs. A GTAR $\varphi = (P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ is trivial if (a) P_2 is a sub-pattern of P_1, or (b) P_1=$\emptyset$. Trivial GTARs are not interesting in practice.

Expressiveness. A GTAR subsumes a number of existing (temporal) rules. (1) *Evolution rules* [8]. An evolution rule is a special case of GTAR, where P_2 is a single-edge event between two nodes in P_1. (2) Graph association rules [13] are a special case of GTARs where P_2 is a single-edge pattern, and Δt=0. Observe that the graph association rules already subsume conventional association rules defined over item sets [13].

The notations used in this paper is shown in Table 1.

4 THE DISCOVERY PROBLEM

We want to detect temporal event associations by explicitly using GTARs. To this end, we introduce support and correlation measures for GTARs (Section 4.1), followed by the formulation of the GTAR discovery problem (Section 4.2).

4.1 Interestingness Measures

To characterize interesting GTARs, below we first define support and confidence for GTARs. These notions have their counterparts in conventional interestingness measures for temporal ARs [18, 23], and are extended for GTARs over temporal graphs.

Rule occurrence. Given $\mathcal{G}_T$ and a GTAR $\varphi = (P_1 \Rightarrow P_2, \bar{u}, \Delta t)$, a time window $O(v)$=$[t_1, t_2]$ ($t_1 \in T, t_2 \in T$) is an *occurrence* of φ in $\mathcal{G}_T$ supported by a node $v \in V$, if the following holds:

- $v \in \mathrm{o}(P_1(\bar{u}), t_1) \cap \mathrm{o}(P_2(\bar{u}), t_2)$, *i.e.,* node v is a match of $\bar{u}$ in P_1 at t_1, and v is a match of $\bar{u}$ in P_2 at t_2, and
- $0 \leq (t_2 - t_1) \leq \Delta t$.

That is, φ occurs if at least a node matches the focus of both P_1 and P_2 at t_1 and t_2, respectively, within Δt time. This enforces a strong association between P_1 and P_2 in terms of common matches. Indeed, without common focus matches, P_2 and P_1 can be irrelevant even one occurred soon after the other.

A time window may contain multiple occurrences of a GTAR. We want to further find "minimal" ones that suffice to support GTARs. This can be characterized by a variant of the *minimal occurrence*, a well established notion for events in item streams [6].

Minimal occurrence. A *minimal occurrence* of a GTAR φ supported by node $v \in V$ is a time window $O(v)$=$[t_1, t_2]$, such that (1) $O(v)$ is an occurrence of φ in $\mathcal{G}_T$ supported by v, and (2) there exists no time window $O'(v) \subset O(v)$ (*i.e.,* $O'(v) = [t'_1, t'_2]$ and $t_1 < t'_1 \leq t'_2 \leq t_2$, or $t_1 \leq t'_1 \leq t'_2 < t_2$), such that $O'(v)$ is an occurrence of φ in $\mathcal{G}_T$ supported by v.

Example 4.1. Recall the GTAR $\varphi_2 = (P_3 \Rightarrow P_4, \bar{u}, \Delta t = 2)$ in Example 3.1. The focus occurrences of P_3 and P_4 in T are shown in Fig. 3. (1) The time window $O_1(x_3)$=$[3, 5]$ is an occurrence of φ_2 supported by x_3. Indeed, x_3 matches $\bar{u}$ of P_1 and $\bar{u}$ of P_2 at time 3 and 5, respectively. Intuitively, it suggests that user x_3 who checks in at a POI at time 5 is potentially influenced by his friend y_2 who checks in at a POI earlier at time 3 (as required by P_3). (2) $O_2(x_3)$=$[3, 4]$ is a minimal occurrence of φ_2 supported by x_3. This suggests that an earlier check-in at time 4 of x_3 at a nearby POI is more likely to be influenced by the check-in of y_2 at time 3.

Similarly, x_1 and x_2 support minimal occurrences $O_3(x_1)$=$[4, 4]$, and $\{O_4(x_2) = [3, 3], O_5(x_2) = [4, 4]\}$, respectively. Observe that O_3 and O_5 refer to the same time window $[4, 4]$, but are supported by different matches. We consider such cases as different occurrences rather than a single one. Indeed, it suggests that both users x_1 and x_2 are influenced by the temporal association at $[4, 4]$, which should be considered as a "stronger" evidence that φ_2 holds.

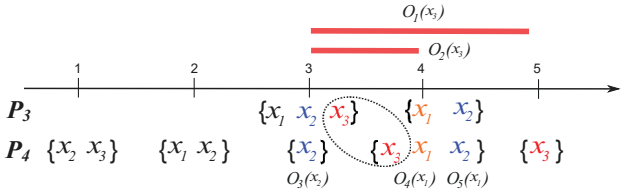


Figure 3: Counting minimal occurrences

Support. Based on minimal occurrences, the support of a GTAR φ in temporal graph $\mathcal{G}_T$, denoted as $\mathrm{Supp}(\varphi, \mathcal{G}_T)$, is defined as

$$\mathrm{Supp}(\varphi, \mathcal{G}_T) = \frac{|O(\varphi, \mathcal{G}_T)|}{|C(\bar{u})||T|}$$

where $O(\varphi, \mathcal{G}_T)$ refers to all the minimal occurrences of φ in $\mathcal{G}_T$. We normalize its size with $|C(\bar{u})|$ (candidates of $\bar{u}$, Section 2) and $|T|$, as there are up to $|C(\bar{u})|$ nodes that support a minimal occurrence of φ, and each node support up to $|T|$ minimal occurrences.

Anti-monotonicity. The support of GTARs is well defined with anti-monotonicity property with the following refinement relation among GTARs. Given two GTARs $\varphi = (P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ and $\varphi' = (P'_1 \Rightarrow P'_2, \bar{u}, \Delta t)$, (1) P'_1 *refines* P_1, denoted by $P_1 \preceq P'_1$, if P'_1=P_1, or P'_1 is obtained by adding nodes or edges to P_1; $P_2 \preceq P'_2$ is similarly defined; (2) φ' *refines* φ, denoted by $\varphi \preceq \varphi'$, if $P_1 \preceq P'_1$ and $P_2 \preceq P'_2$.

We show that GTAR support is anti-monotonic.

Lemma 4.2. *For any temporal graph* $\mathcal{G}_T$ *and any* GTARs φ *and* φ', $\mathrm{Supp}(\varphi, \mathcal{G}_T) \geq \mathrm{Supp}(\varphi', \mathcal{G}_T)$ *if* $\varphi \preceq \varphi'$.

Proof sketch: Consider GTARs $\varphi = (P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ and $\varphi' = (P'_1 \Rightarrow P'_2, \bar{u}, \Delta t)$ such that $\varphi \preceq \varphi'$. By definition of GTAR support, it suffices to show that at any time $t \in T$, $\mathrm{o}(P'_1, \bar{u}, t) \subseteq \mathrm{o}(P_1, \bar{u}, t)$ and $\mathrm{o}(P'_2, \bar{u}, t) \subseteq \mathrm{o}(P_2, \bar{u}, t)$ As $P_1 \preceq P'_1$, for any node $v \in \mathrm{o}(P'_1, u'_1, t)$, v is also a match of $\bar{u}$ in P_1 at time t. Otherwise, either v is not a match of $\bar{u}$ in P_1, or P_1 contains at least an edge that is not in P'_1. Both leads to contradiction. Hence, $\mathrm{o}(P'_1, \bar{u}, t) \subseteq \mathrm{o}(P_1, \bar{u}, t)$. Similarly, $\mathrm{o}(P'_2, \bar{u}, t) \subseteq \mathrm{o}(P_2, \bar{u}, t)$. The anti-monotonicity thus follows. □

Example 4.3. For GTAR $\varphi_2 = (P_3 \Rightarrow P_4, \bar{u}, \Delta t = 2)$ in Example 3.1 and temporal graph $\mathcal{G}_T$ of Fig. 2, as there are in total 4 minimal occurrences contributed by common focus matches x_1, x_2 and x_3, respectively, $\mathrm{Supp}(\varphi, \mathcal{G}_T) = \frac{1+2+1}{3*5} = 0.26$.

Confidence. The confidence of a GTAR φ in $\mathcal{G}_T$, denoted as $\mathrm{Conf}(\varphi, \mathcal{G}_T)$, measures how likely P_2 occurs within Δt time at the focus occurrence of P_1. We define $\mathrm{Conf}(\varphi, \mathcal{G}_T)$ as

$$\mathrm{Conf}(\varphi, \mathcal{G}_T) = \frac{\mathrm{Supp}(\varphi, \mathcal{G}_T)}{\mathrm{Supp}(P_1, \mathcal{G}_T)}$$

where $\mathrm{Supp}(P_1, \mathcal{G}_T)$ is defined as

$$\mathrm{Supp}(P_1, \mathcal{G}_T) = \frac{\sum_{i \in [1,T]} |\mathrm{o}(P_1, \bar{u}, i)|}{|C(\bar{u})||T|}$$

That is, $\mathrm{Conf}(\varphi, \mathcal{G}_T)$ quantifies the probability that a focus match in the occurrences of P_1 also support a minimal occurrence of φ.

Example 4.4. Consider GTAR $\varphi_2 = (P_3 \Rightarrow P_4, \bar{u}, \Delta t = 2)$ and GTAR in Example 4.3. We can verify that $\text{Supp}(P_6, \mathcal{G}_T) = \frac{2+2+1}{3*5} = 0.33$. Hence, $\text{Conf}(\varphi, \mathcal{G}_T) = \frac{0.26}{0.33} = 0.78$.

The anti-monotonicity does not hold for GTAR confidence: the confidence of a GTAR may be larger than those it refines. However, it has a "conditional" anti-monotonicity, as shown below.

Lemma 4.5. *Given two* GTARs φ=$(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ *and* φ'=$(P_1 \Rightarrow P_2', \bar{u}, \Delta t)$, $\text{Conf}(\varphi', \mathcal{G}_T) \leq \text{Conf}(\varphi, \mathcal{G}_T)$ *if* $P_2 \preceq P_2'$.

Proof sketch: As both φ and φ' pertain to the same antecedent event P_1, $\varphi \preceq \varphi'$ if $P_2 \preceq P_2'$. Given Lemma 4.2 and the definition of $\text{Conf}(\varphi, \mathcal{G}_T)$, we can verify the following.

$$\text{Conf}(\varphi', \mathcal{G}_T) = \frac{\text{Supp}(\varphi', \mathcal{G}_T)}{\text{Supp}(P_1, \mathcal{G}_T)} \leq \frac{\text{Supp}(\varphi, \mathcal{G}_T)}{\text{Supp}(P_1, \mathcal{G}_T)} = \text{Conf}(\varphi, \mathcal{G}_T)$$

The lemma thus follows. □

Informative GTARs. We prefer GTARs with high support and confidence. Moreover, the rules should also be "informative" [30] and concise. That is, they bridge complex events rather than trivial (though more frequent) events (*e.g.,* routine calls between two persons). Meanwhile, they should be easily interpreted. To strike a balance, we introduce *maximal* GTARs with size bound.

Given a support threshold θ, a GTAR φ=$(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ is a maximal GTAR, if (1) $\text{Supp}(\varphi, \mathcal{G}_T) \geq \theta$, and (2) there exists no GTAR φ', such that $\varphi \preceq \varphi'$, and $\text{Supp}(\varphi', \mathcal{G}_T) \geq \theta$. It is a *b-maximal* GTAR, if both P_1 and P_2 have at most b edges.

Example 4.6. Given graph $\mathcal{G}_T$ in Fig. 2, focus $\bar{u}$, $\Delta t = 2$, support threshold $\sigma = 0.1$, confidence threshold $\theta = 0.7$, and size bound $b = 3$, the GTAR $\varphi_2 = (P_3 \Rightarrow P_4, \bar{u}, \Delta t = 2)$ is a 3-maximal GTAR. Indeed, we can verify that no φ' such that $\varphi \preceq \varphi'$ has satisfiable support and confidence.

4.2 The Discovery Problem

We now formalize the GTAR discovery problem. For practical applications, the problem is parameterized with user specified focus $\bar{u}$, time bound Δt, support and confidence threshold. In addition, a size bound b is posed on the events to strike a balance between informativeness and interpretability.

- Input: Temporal graph $\mathcal{G}_T$, focus $\bar{u}$, time interval Δt, size bound b, support threshold σ and confidence threshold θ;
- Output: The set of b-maximal GTARs Σ pertaining to $\bar{u}$ and Δt such that for each GTAR $\varphi \in \Sigma$, $\text{Supp}(\varphi, \mathcal{G}_T) \geq \sigma$, and $\text{Conf}(\varphi, \mathcal{G}_T) \geq \theta$.

Usually, we have $\Delta t \ll T$, and $|V| \ll b$. That is, we consider discovering GTARs that associate two reasonably large events in a short period, in a relatively much longer time period.

5 GTAR DISCOVERY

A brute-force algorithm first enumerates all the events in $\mathcal{G}_T$, and then enumerates and validates GTARs by pairing the events and adding the temporal constraints. Enumeration of all events is already expensive when $\mathcal{G}_T$ is large. Moreover, one has to wait until all the events are discovered before rule generation; both are intractable processes themselves.

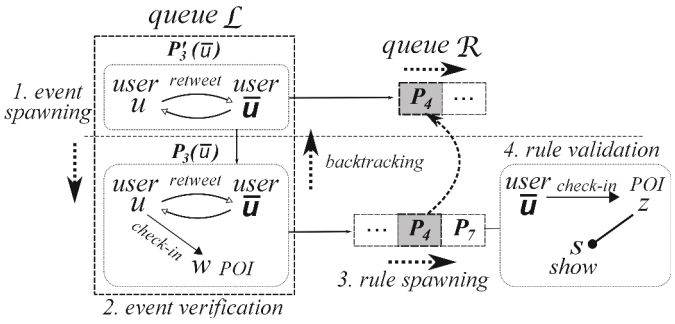


Figure 4: GTAR **discovery**

We can perform GTARs discovery more efficiently. The idea is to integrate event mining and rule discovery as a *single* process, and prune non-interesting events and GTARs as early as possible.

5.1 Discovery Algorithm

We present the GTAR discovery algorithm, denoted as DisGTAR. In a nutshell, the algorithm DisGTAR interleaves event mining and rule discovery as a single mining process: it generates promising GTAR candidates by spawning and verifying the LHS events and dynamically "append" promising RHS events, and validates the generated GTAR candidates only when necessary. It eliminates non-interesting GTARs whenever possible.

We start with the auxiliary structures used by DisGTAR.

Auxiliary structures. Algorithm DisGTAR uses structures below.

(1) It dynamically maintains an event lattice $\mathcal{P}$. Each lattice node in $\mathcal{P}$ at level i records an event P with i edges, associated with (a) its anchored matches $\text{o}(P, \bar{u}, \mathcal{G}_T)$ and (b) its occurrences $O(P, \mathcal{G}_T)$, both encoded as bit vectors to save space. The auxiliary information will be used to validate GTAR candidates.

(2) It uses two priority queues below to control the generation of $\mathcal{P}$: (a) a LHS *queue* $\mathcal{L}$, where each entry $\mathcal{L}[P_1]$ points to a LHS event P_1 in $\mathcal{P}$, and (b) each entry $\mathcal{L}[P_1]$ is associated with a *RHS queue* $\mathcal{L}[P_1].\mathcal{R}$ that points to a set of RHS events in $\mathcal{P}$. Both priority queues maintain two keys for an event P: its level (edge number) in $\mathcal{P}$, and $\text{Supp}(P, \mathcal{G}_T)$. A GTAR φ=$(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ is encoded by the LHS event $\mathcal{L}[P_1]$ and its associated RHS event $\mathcal{L}[P_1].\mathcal{R}[P_2]$.

Example 5.1. A fraction of an LHS queue $\mathcal{L}$ and RHS queues $\mathcal{R}$ is shown in Fig. 4. The corresponding fraction of event lattice $\mathcal{P}$ (not shown) contains LHS events P_3' and P_3, and RHS events P_4 and P_7. LHS queue $\mathcal{L}$ contains P_3' and P_3. The event P_3 is associated with a RHS queue $P_3.\mathcal{R}$, which contains RHS events P_4 and P_7. Events P_3 and P_4 together encode the GTAR φ_2 in Example 4.6.

Overview. Algorithm DisGTAR, illustrated in Fig. 5, discovers and stores maximal GTARs in a set Σ. It "cold-starts" GTAR discovery by initializing the event lattice $\mathcal{P}$ and the LHS stack $\mathcal{L}$ with an event P_u as a single-node event that contains the focus $\bar{u}$ (line 1). DisGTAR then expands $\mathcal{P}$ by interleaving two processes, both guided by *best-first* strategies: a prioritized *LHS event generation* (lines 2-8) to generate and verify best new LHS events; and a *rule validation* (lines 10-18) to generate and validate new GTAR candidates by appending best RHS events to verified LHS events.

Algorithm DisGTAR

Input: temporal graph $\mathcal{G}_T$, focus $\bar{u}$, integer b, bound b, time window Δt support threshold σ and confidence threshold θ.
Output: a set Σ of b-maximal GTARs *w.r.t.* $\sigma$and θ.

1. set Σ:=$\emptyset$; event lattice $\mathcal{P}$:=$\{P_u\}$; priority queue $\mathcal{L}$:=$\{P_u\}$;
2. **while** $\mathcal{L}$ is not empty **do**
3. /* LHS *event generation* */
4. event P_1:= $\mathcal{L}$.poll (); /* *retrieves the "best"* LHS *event* */
5. **if** P_1 is "spawnable" **then** set $S(P_1)$:=**lSpawn** (P_1, $\mathcal{P}$);
6. **for each** $P_1' \in S(P_1)$ **do**
7. **eMatch** (P_1', $\mathcal{G}_T$, $\mathcal{P}$); /* LHS *event verification* */
8. $\mathcal{L}$.push ($S(P_1) \cup P_1$); continue; /* *prioritizes* LHS *events* */
9. /* GTARs *rule generation* */
10. initializes queue $P_1.\mathcal{R}$;
11. **while** $P_1.\mathcal{R}$ is not empty **do**
12. event P_2 := $P_1.\mathcal{R}$.poll (); /**retrieves the "best"* RHS *event* */
13. generates GTAR φ=($P_1 \Rightarrow P_2$, $\bar{u}$, Δt);
14. **valGTAR** (φ, $\mathcal{G}_T$); /**rule validation* */
15. Identify maximal GTARs Σ'; $\Sigma := \Sigma \cup \Sigma'$;
16. **if** P_2 is "spawnable" **then** set $S(P_2)$:=**rSpawn** (P_1, P_2, $\mathcal{P}$);
17. **for each** $P_2' \in S(P_2)$ **do eMatch** (P_2, $\mathcal{G}_T$, $\mathcal{P}$);
18. $P_1.\mathcal{R}$.push ($S(P_2)$); /* *prioritizes* RHS *events* */
19. **return** Σ;

Figure 5: Algorithm DisGTAR

LHS *event generation* (lines 2-8). The algorithm DisGTAR uses the LHS queue $\mathcal{L}$ to generate LHS events. It applies a *best-first* strategy, which gives priority to the generation of LHS events with larger size and smaller support (lines 5-8; to be discussed shortly). To this end, it (1) uses an atomic operation lspawn (line 5) to spawn a set of LHS events $S(P_1)$ from the "best" LHS event P_1 verified so far, and (2) invokes a procedure eMatch (line 7) to perform event matching, which finds Supp$(P_1', \mathcal{G}_T)$ and its focus occurrence $o(P_1', \bar{u}, \mathcal{G}_T)$ for each new event P_1'. The verified LHS events are prioritized based on their size and support in $\mathcal{L}$ (line 8).

Rule generation (lines 10-18). Given a verified LHS event P_1, DisGTAR generates GTARs that pertain to P_1, by appending RHS events to $P_1.\mathcal{R}$. Similar to LHS event generation, it also uses a *best-first* approach to prioritize the process of RHS events. The difference is that (1) it prefers RHS events with higher support (lines 16-18, to be discussed), and (2) it uses a different operator rSpawn to reduce RHS events verification based on both the best LHS and RHS events so far (line 16, to be discussed).

Each RHS event produces a new GTARs (line 13). For each GTAR, it invokes a procedure valGTAR to perform GTAR validation and computes its support and confidence (line 14). It then finds maximal GTARs in $\mathcal{P}$ and updates Σ accordingly (line 15). The rule generation terminates if all the GTARs pertaining to P_1 under event size bound b are checked ($P_1.\mathcal{R}$ is empty).

Algorithm DisGTAR "switches" between the two processes and reduces redundant verification and validation by checking two sets of "spawnable" conditions for LHS and RHS events, respectively. (1) When a LHS event P_1 is no longer spawnable, it starts GTAR generation at P_1 (line 10); (2) When no RHS events can be spawned with a LHS event P_1 (line), it "backtracks" to the parent of P_1 (retrieved from L, line 4) and starts LHS event generation. We introduce the details of the spawnable conditions in Section 5.2.

The above process is illustrated in Fig. 4. We next introduce the key components of algorithm DisGTAR.

GTAR **Generation**. Algorithm DisGTAR generates GTAR candidates by performing two core operations below. Both *share* the access to $\mathcal{P}$ to avoid redundant generation and verification; an event in $\mathcal{P}$ can be either an LHS or RHS events in different GTARs.

Operators lSpawn. Given a LHS event P_1, operator **lSpawn** $(P_1, \mathcal{P})$ spawns a set of LHS event $S(P_1)$ from P_1. Each event $P_1' \in S(P_1)$ is obtained by adding an event edge e=(u, u') to P_1, such that (a) either u or u' is in P, and (b) e has at least a candidate $e' = (v, v', e.I) \in C(e)$. If P_1' is new ($P_1' \notin \mathcal{P}$), it inserts P_1' to $\mathcal{P}$. The events are then verified by procedure eMatch.

Operators rSpawn. By default, operator **rSpawn** works similarly as its counterpart for LHS events. The difference is that it optimizes the spawning of RHS events for LHS event P_1 when DisGTAR "backtracks" from its children. We defer the details of optimized rSpawn in Section 5.2.

Example 5.2. Consider the LHS queue $\mathcal{L}$ in Fig. 4. When algorithm DisGTAR executes **lSpawn** $(P_3', \mathcal{P})$, it spawns LHS event P_3 (among others) by adding an edge (u, w) to P_3', and inserts P_3 to $\mathcal{P}$. Similarly, when **rSpawn** $(P_3, P_4', \mathcal{P})$ is executed ($P_4'(\bar{u})$ is a single-edge pattern $(\bar{u}, z)$; not shown), it spawns P_4 (in Fig. 1) by adding edge (z, s) to P_4', and inserts P_4 to $\mathcal{P}$.

Prioritization. Recall the anti-monotonicity properties of GTAR support and confidence. (1) GTAR candidates with larger LHS events are "closer" to maximal GTARs in P (Lemma 4.2). (2) GTARs with LHS events P_1 having relatively smaller support Supp$(P_1, \mathcal{G}_T)$ and RHS events P_2 with relatively larger support Supp$(P_2, \mathcal{G}_T)$ tend to have higher confidence (Lemma 4.5).

Based on these intuitions, DisGTAR prioritizes the spawning of events as follows. (1) DisGTAR inserts all the events in newly spawned LHS events $S(P_1)$ to priority queue $\mathcal{L}$, following a *descending order* of their support. It also inserts P_1 to $\mathcal{L}$ for backtracking. The LHS event with the lowest support is picked to spawn new LHS events. (2) Given a "best" LHS event P_1 in (1), $\mathcal{R}$ pushes newly spawned RHS events $S(P_2)$ by *ascending order* of their support. Fixing LHS event, the RHS event with highest support is selected to construct the next GTAR candidate. The prioritization in (1) and (2) lead to b-maximal GTARs with high confidence faster.

Example 5.3. Continuing the example 5.2, once a set of LHS events $S(P_3')$ is spawned by **lSpawn**$(P_3', \mathcal{P})$ (step 1), DisGTAR verifies each event (step 2) and pushes them into LHS queue with descending order of their support. It selects P_3 as the top event to be spawned with relatively low support. As P_3 already has 3 edges (with size bound b=3), it is no longer "spawnable". Algorithm DisGTAR then enters rule spawning (step 3).

In the rule spawning with LHS event P_3, once a set of RHS events $S(P_4')$ is spawned by **rSpawn**$(P_3, P_{4'}, \mathcal{P})$, DisGTAR verifies and selects P_4 with relatively higher support, and constructs the GTAR φ_2 for further validation.

We now introduce procedures eMatch and valGTAR, used in event verification and rule validation, respectively.

Procedure eMatch

Input: An event $P(V_P, E_P, L_P)$, temporal graph $\mathcal{G}_T$
Output: focus occurrence $o(P, \bar{u}, \mathcal{G}_T)$;

1. **for each** $e = (u, u') \in E_P$ **do**
2. set $C(e) := f(e)$; /* initializes edge candidates*/
3. **while** there are changes in edge candidate sets **do**
4. **for each** pair of edges $e^P = (u^P, u)$ and $e=(u, u')$ in E_P **do**
/* refines candidates with "join" $C(e^P) \bowtie C(e)$*/
5. **for each** edge $e^{P'}=(v^{P'}, v, e^{P'}.t) \in C(e^P)$ **do**
6. **if** there exists no edge $e'=(v, v', t) \in C(e)$ such that $e^{P'}.t=e'.t$ **then** $C(e^P) := C(e^P) \setminus \{e^{P'}\}$;
7. **if** $C(e^P) = \emptyset$ **then return** $\emptyset$;
8. **for each** edge $e'=(v, v', t) \in C(e^P)$ **do**
9. **if** there exists no edge $e^{P'}=(v^{P'}, v, t) \in C(e^P)$ such that $e^{P'}.t=e'.t$ **then** $C(e) := C(e) \setminus \{e'\}$;
10. **if** $C(e) = \emptyset$ **then return** $\emptyset$;
11. derives $R^{T'}$ and $o(P, \bar{u}, \mathcal{G}_T)$ from edge candidates;
12. **return** $o(P, \bar{u}, \mathcal{G}_T)$;

Figure 6: Procedure eMatch

Event verification. A straightforward algorithm first induces a set of T "snapshots" of $\mathcal{G}_T$, and then computes matching relation R_t between P and each snapshot G_t ($t \in T$) one by one. This incurs redundant verification, and is expensive when $\mathcal{G}_T$ and T are large.

We introduce a more efficient procedure eMatch (illustrated in Fig. 6). The procedure performs a "one batch" matching to directly compute R^T between P and $\mathcal{G}_T$, without one-by-one matching. It extends the algorithm in [20] by using a fixed-point join over candidates as tables. More specifically, it works as follows.

(1) For each edge $e = (u, u') \in E_P$, eMatch initializes its candidate set $C(e)$ specified by the candidate function f (lines 1-2), as a "table" of triples $< v, v', t >$, each encodes an edge $e=(v, v', t)$ in $\mathcal{G}_T$.

(2) It iteratively refines the candidate sets via *fixed-point* "join" operations as follows. For each pair of event edges $e^P = (u^P, u)$ and $e = (u, u^c)$ in P, it performs a join operation $C(e^P) \bowtie C(e)$ with condition "$C(e^P).v' = C(e).v \wedge C(e^P).t = C(e).t$"which encodes the semantics of temporal matching (lines 4-10). eMatch iteratively removes the edge candidates from $C(e^P)$ (line 6) and $C(e)$ (line 9) which fails the join condition, until no edge candidate can be removed from the candidate sets (line 3).

Procedure eMatch then extracts the temporal matching relation R^T by pairing the edges with their candidates, and extracts the focus occurrences $o(P, \mathcal{G}_T)$, and $\text{Supp}(P, \mathcal{G}_T)$ (line 11) from R^T. The information is associated to P in the event lattice $\mathcal{P}$.

Remarks. We optimize the join operation in eMatch by "grouping" consecutive timestamps as time intervals and perform range intersections.

GTAR **Validation**. Given a GTAR candidate $\varphi=(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$, procedure valGTAR (not shown) validates φ as follows.

(1) As P_1 is verified in LHS event verification, it invokes eMatch to verify P_2 only when necessary, and update $\mathcal{P}$ accordingly.

(2) Procedure valGTAR then finds $O(\varphi, \mathcal{G}_T)$, the minimal occurrences of φ, by a linear scan over T. It finds the common focus matches of P_1 and P_2. It initializes a pair of pointers (t_1, t_2) for P_1 to indicate the first time interval in which a common focus match of P_1 occurs, and similarly set pointers (t'_1, t'_2) for P_2. It then iteratively counts the intersection of the pointed time intervals of P_1 and P_2 as a minimal occurrence and moves the pointers, until the last interval in T is visited. This takes a linear scan of T.

Procedure valGTAR then computes and returns $\text{Supp}(\varphi, \mathcal{G}_T)$ and $\text{Conf}(\varphi, \mathcal{G}_T)$ with $O(\varphi, \mathcal{G}_T)$ and $\text{Supp}(P_1, \mathcal{G}_T)$ by definition.

5.2 Optimization

Algorithm DisGTAR further reduces cost with a set of pruning rules, verified by the result below.

Lemma 5.4. *For a* GTAR $\varphi=(P_1 \Rightarrow P_2, \bar{u}, \Delta t)$ *with* $\text{Supp}(\varphi, \mathcal{G}_T) \geq \sigma$ *and* $\text{Conf}(\varphi, \mathcal{G}_T) \geq \theta$,

(a) $\text{Supp}(P_1, \mathcal{G}_T) \geq \sigma$, *and* $\text{Supp}(P_2, \mathcal{G}_T) \geq \sigma$;
(b) for any GTAR φ' *that* $\varphi' \preceq \varphi$, $\text{Supp}(\varphi, \mathcal{G}_T) \geq \sigma$; *and if* φ' *pertains to* P_1, *then* $\text{Conf}(\varphi, \mathcal{G}_T) \geq \theta$;
(c) If φ *is a* b*-maximal* GTAR, *then any* GTAR φ' *is not a* b*-maximal* GTAR *if* $\varphi' \preceq \varphi$.

Proof sketch: (a) We can verify that the event support $\text{Supp}(P, \mathcal{G}_T)$ is a valid upper bound of any GTARs with P as either LHS or RHS event. (b) The result can be verified by the (conditional) anti-monotonicity of GTAR support (Lemma 4.2) and confidence (Lemma 4.5). (c) If $\varphi' \preceq \varphi$ and φ is a b-maximal GTAR, then one can add at least an edge to either LHS or RHS event of φ' that leads to φ. Thus φ' is not a b-maximal GTAR and can be pruned. □

In accordance, DisGTAR uses the pruning rules below.

(a) DisGTAR spawns events only from "spawnable" P, *i.e., P* has no more than b edges, and $\text{Supp}(P, \mathcal{G}_T) \geq \sigma$ (line 5 and 16, Fig. 5).

(b) For any validated GTAR φ with RHS event P_2, if either $\text{Supp}(\varphi, \mathcal{G}_T) < \sigma$, or $\text{Conf}(\varphi, \mathcal{G}_T) < \theta$, DisGTAR stops spawning the RHS events from P_2 (line 16, Fig. 5).

(c) Optimized rSpawn (line 16, Fig. 5): whenever DisGTAR "backtracks" from a LHS event P_1 to its parent P'_1, rSpawn (i) bookkeeps the set of RHS events $\mathcal{P}_2$ from $P_1.\mathcal{R}$ that are in a maximal GTAR pertaining to P_1, and (ii) skips all GTARs that pertains to P'_1 and P'_2 with $P'_2 \preceq P_2$ ($P_2 \in \mathcal{P}_2$).

Example 5.5. We continue with Example 5.3. (1) Once RHS event P_7 is generated by **rSpawn** (step 3, rule spawning), DisGTAR finds its support $\text{Supp}(P_7, \mathcal{G}_T) = 0.06 < \sigma$. By pruning rule (c) (Lemma 5.4(c)), DisGTAR stops spawning new RHS events from P_7 and skips all its descendants in $\mathcal{P}$. (2) When DisGTAR backtracks from P_3 to P'_3, since RHS event P_4 is in a validated 3-maximal GTAR φ_2, rSpawn starts spawning LHS events from P_4 for P'_3, skipping all the LHS events refined by P_4 (*e.g.,* P'_4 in Example 5.2).

5.3 Performance analysis

The algorithm DisGTAR correctly discovers the maximal GTARs when terminates, and is feasible over large temporal graphs.

Correctness. Algorithm DisGTAR enumerates GTAR candidates. It enumerates all the LHS events via depth-first pattern generation. For each LHS event P, it enumerates RHS events that contributes to maximal GTARs pertaining to P. The pruning rules only remove redundant events in pattern and rule generation. Moreover, the procedure eMatch and valGTAR correctly verify and validate the events and GTARs, respectively.

Complexity. Algorithm DisGTAR incurs the following cost.

Event verification cost. Denote by $N(b)$ the number of events with size bounded by b. For each event, procedure eMatch takes $O(|T|(b+|V|)(b+|E|))$ time to compute $\text{Supp}(P, \mathcal{G}_T)$, following the analysis in [20], with an additional cost on the timestamps posed on the edges. Thus DisGTAR takes $O(|T|N(b)(b + |V|)(b + |E|))$ time.

GTAR *validation cost.* There are at most $N(b)^2$ GTARs for $N(b)$ events. For each GTAR φ, valGTAR verifies RHS event only when necessary, and $O(|T|)$ time to scan T and identify minimal occurrences of φ. Thus, it takes in total $O(|T|N(b)(b + |V|)(b + |E|) + N(b)^2|T|)$ time to validate and identify all maximal GTARs.

Taken together, the overall cost of DisGTAR is in $O(|T|N(b)(b+|V|)(b + |E|) + N(b)^2|T|)$ time. In practice, b is small, and $N(b)$ is significantly reduced by the optimization strategies.

Space cost. Algorithm DisGTAR takes in total $O(N(b)|C(\bar{u})||T|)$ space. Indeed, (1) it maintains up to $|C(\bar{u})|$ focus matches with associated timestamps up to $|T|$ for each event, and (2) the LHS and RHS queues maintain up to $N(b)$ pointers to the shared $\mathcal{P}$.

Anytime performance. With integrated event mining and rule validation, DisGTAR enables "anytime" performance that can return GTARs as the events are discovered. This is desirable for applications that require ad-hoc detection of temporal associations. As verified by our experimental study (Section 6), DisGTAR converges 73 times faster and report GTARs with accuracy 90% compared with its "enumerate-and-validate" counterpart.

Remarks. Algorithm DisGTAR can be easily extended for GTARs with multiple focus nodes, with extended support and confidence that aggregate weighted sum of minimal occurrences. To this end, it only needs to extend valGTAR to validate GTARs with multiple focus. The time and space complexity remains intact.

6 EXPERIMENTAL EVALUATION

Using real-life and synthetic graphs, we conducted four sets of experiments to evaluate (1) the efficiency and scalability of GTARs discovery algorithms, (2) the impact of the parameters (*e.g.,* support, event size) to their efficiency, (3) the "anytime" performance of GTAR discovery, and (4) the effectiveness of GTARs model.

Experimental setting. We used the following setting.

Datasets. We use the following datasets. (1) Citation[1] is a citation network of 4.3M entities (*e.g.,* papers, authors, publication venues), 21.7M edges (*e.g.,* citation, published at), and 273 labels (*e.g.,* keywords, research domains), with timestamps corresponding to publication date. The active timestamps span from the year 1936 to the year 2015 ($|T|$=80). (2) Panama[2] contains in total $839K$ offshore entities (*e.g.,* companies, countries, jurisdiction), $3.6M$ relationships (*e.g.,* establish, close) and 433 labels covering 40 years of offshore entities and financial activities including $12K$ active days ($|T|$=$12K$). (3) MovieLens[3] records $10M$ ratings of $71.5K$ users given to $10K$ movies in an online movie recommendation system. In addition to User and Movie, the temporal graph includes 20 other labels demonstrating the category of each movie (*e.g.,* Romance, Drama) and it has 1439 hours of interactions ($|T|$=1439).

Besides real-world temporal graphs, we also use a generator that simulates a synthetic dynamic traffic network using VISSIM [15]. The generator is controlled by the number of nodes (vehicles, locations, traffic signs), edges (roads), and the size of snapshots. We generated synthetic traffic networks of up to $91K$ snapshots, each with $1.2M$ nodes and $15M$ edges on average.

Algorithms. We implemented the following algorithms, all in Java: (1) Algorithm DisGTAR (including procedures eMatch and valGTAR); (2) Algorithm DisGTARn, a variant of DisGTAR without the pruning strategies (Section 5.2); (3) Algorithm IsoGTAR, a variant of DisGTAR that uses an event matching procedure different with eMatch, which "isolates" the snapshots and computes event matching over each snapshots one by one (Section 5.1). For a fair comparison, we apply all the applicable optimizations of DisGTAR to IsoGTAR. (4) Algorithm SeqGTAR is a variant of DisGTAR that separates event mining and rule discovery to two independent processes. It discovers all the events using the event spawning and procedure eMatch in DisGTAR, and then enumerates and validates all GTARs candidates.

We ran all of our experiments on a machine powered by an Intel 2.3GHz CPU with 64GB memory. Each experiment was run 3 times and the average is reported here.

Experimental results. We next report our findings.

Exp-1: Performance of DisGTAR. We first evaluate the efficiency of DisGTAR over real-world and synthetic datasets, compared with DisGTARn, SeqGTAR and IsoGTAR. For all the datasets, we sampled focus nodes specified with common interests (*e.g.,* paper with popular topics in Citation, active companies and jurisdictions in Panama) with proper candidate size (100-300 nodes). We fix Δt=5, b=4 and draw support (resp. confidence) threshold from $[0.1 - 0.2]$ (resp. $[0.6 - 0.9]$) for all datasets. For a fair comparison, we tuned the thresholds to ensure (1) the mining process runs to completion for most baselines, and (2) over all datasets, they return reasonably large number ($\geq$ 100) of maximal GTARs.

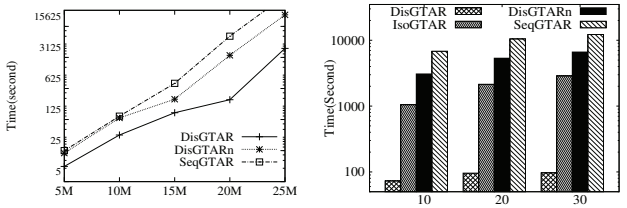
Efficiency. The performance of GTAR discovery is shown in Table 2. The result tells us the following. (1) DisGTAR performs best among all the algorithms. It outperforms DisGTARn, SeqGTAR and IsoGTAR by 6.28, 7.85 and 64.79 times on average over real-world graphs. In particular, it outperforms the three baselines by 43, 71 and 561 times respectively over Citation. (2) In general, DisGTAR is feasible over large graphs. For example, it takes 22 seconds over Citation with $26M$ nodes and edges and 80 snapshots, and identifies 20 4-maximal GTARs. (3) The pruning strategy significantly reduces the verification cost. DisGTAR incurs at most 90.3% number of verifications over all real-world datasets. In particular, it reduces 98% verifications over Citation.

Algorithm DisGTAR takes less time over Panama with more verifications than over Citation. This is because eMatch takes much less time over Panama which is more sparse and heterogeneous, despite of more verifications. Algorithm IsoGTAR takes more than 3.5 hours over Citation with 80 snapshots, and does not run to completion in other datasets in 10 hours.

[1] https://aminer.org/citation
[2] https://offshoreleaks.icij.org/pages/database
[3] https://grouplens.org/datasets/movielens/

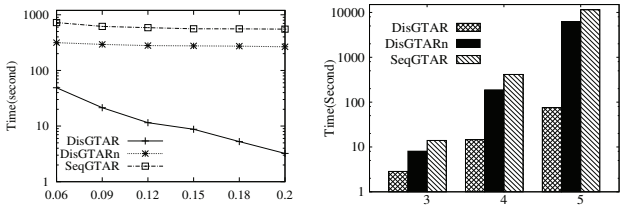
	DisGTAR		DisGTARn		SeqGTAR		IsoGTAR	
	Time (s)	# of verification	Time (s)	# of verification	Time (s)	# of verification	Time (s)	# of verification
Panama	9	1,194	276	8,393	560	8,393	N/A	
Citation	22	157	994	12,507	1,621	12,507	12,721	11,461
MovieLens	558	191	2,432	1,423	2,445	1,423	N/A	
Synthetic	42	303	358	20,230	816	20,230	N/A	

Table 2: Performance of GTAR discovery



(a) Varying $|G|$ (Synthetic) (b) Varying $|T|$ (Synthetic)

Figure 7: Scalability of DisGTAR



(a) Varying σ (Panama) (b) Varying b (Panama)

Figure 8: Impact of parameters

We generate a synthetic graph with $91K$ snapshots, each with size $(1.2M, 15M)$ (*i.e.,* contains $1.2M$ nodes and $15M$ edges). The results over synthetic graphs (Table 2) is consistent with its counterparts over real-world graphs. In particular, DisGTAR outperforms DisGTARn and SeqGTAR by 8.5 and 19.5 times.

Scalability. We also evaluated the scalability of DisGTAR vs. the baselines over synthetic graphs.

(1) Fixing $|T|$=$91K$, σ=0.2, θ=0.15, b=4, and node size $|V|$=$1.2M$, we varied $|E|$ from $5M$ to $25M$. Fig. 7(a) shows that all algorithms take more time, as expected. Algorithm DisGTAR is less sensitive to $|G|$ due to the pruning strategies. While SeqGTAR does not terminate after 10 hours over $|G|$ with size $(1.2M, 25M)$, DisGTAR successfully discovered all the maximal GTARs in 1.38 hours.

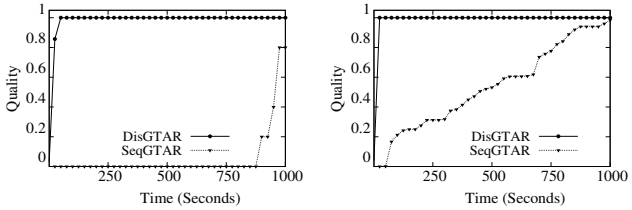
(2) Fixing $|G|$ as $(1.2M, 15M)$, we varied $|T|$ from 10 to 30. As shown in Fig. 7(b), all the algorithms take more time with larger $|T|$. DisGTAR is much less sensitive than IsoGTAR due to the "packing" of consecutive timestamps to time intervals for more efficient fixed-point joins (Section 5.1), reducing the impact of $|T|$. Among the algorithms, IsoGTAR is most sensitive due to its one-by-one matching. This also verifies the results that IsoGTAR fails to complete over real-world graphs with more snapshots (Table 2).

Exp-2: Impact of parameters. We evaluated the impact of the support threshold σ, θ and event size bound b. We report the result over Panama; the results over other real-world datasets Citation and MovieLens are consistent, hence omitted.

Varying σ. Fixing b=4, confidence θ=0.9, and $\Delta t = 5$, we vary σ from 0.06 to 0.2. As shown in Fig. 8(a), all the algorithms take less time for larger σ. DisGTAR is more sensitive to larger σ compared with DisGTARn and SeqGTAR, due to its optimization strategies.

Varying b. Fixing confidence $\theta = 0.9$, σ=0.12 and $\Delta t = 5$, we vary b from 3 to 5. As shown in Fig. 8(b), all the algorithms take longer time when b is larger due to more events need to be verified. However, DisGTAR is less sensitive due to the pruning strategy.

We also evaluated the impact of θ (results not shown). We found that the algorithms are insensitive to the change of θ except for DisGTAR: it takes less time with larger θ. This is also due to that its pruning strategy takes advantage of the conditional anti-monotonicity of GTAR confidence.



(a) Time vs. Accuracy (Citation) (b) Time vs. Accuracy (Panama)

Figure 9: Anytime performance

Exp-3: Anytime performance. In this experiment, we evaluate the impact of time constraint on the anytime performance of DisGTAR, compared with SeqGTAR. To this end, we define a metric of *anytime quality* of the GTARs as follows. Denote the set of maximal GTARs discovered by DisGTAR (resp. SeqGTAR) at time t as Σ^t (Σ^{*t}), and the complete maximal GTAR as Σ^*. The anytime quality at time t of DisGTAR (resp. SeqGTAR) is computed as $\frac{\sum_{\varphi\in\Sigma^t} \mathsf{Conf}(\varphi,\mathcal{G}_T)}{\sum_{\varphi\in\Sigma^*} \mathsf{Conf}(\varphi,\mathcal{G}_T)}$ (resp. $\frac{\sum_{\varphi\in\Sigma^{*t}} \mathsf{Conf}(\varphi,\mathcal{G}_T)}{\sum_{\varphi\in\Sigma^*} \mathsf{Conf}(\varphi,\mathcal{G}_T)}$). We set a time bound t=1000 seconds for both algorithms.

As shown in Fig. 9(b) and Fig. 9(a), DisGTAR converges with high quality GTARs much faster than SeqGTAR. It produces the results with a quality above 0.9 within 8 seconds and 18 seconds over Panama and Citation, respectively. In contrast, SeqGTAR takes 850 seconds to reach the same quality in Panama, and waits for 900 seconds over Citation before it starts to produce GTARs.

Exp-4: Case study. To demonstrate the effectiveness of GTARs and its application, we perform two case studies below.

Real-world GTARs. We manually examined top GTARs returned by DisGTAR over real-world datasets. Two real-world GTARs are shown in Fig. 10, from Panama and Citation, respectively.

(1) GTAR φ_3 =$(P_8 \Rightarrow P_9, \bar{u}, \Delta t = 1.5K)$: The rule identifies a temporal association among offshore entities that indicate a business shifting operation. It states that *companies with jurisdiction area "BVI" (British Virgin Islands) that were once inactive become active again with a changed jurisdiction "Panama" in* 4 *years* ($<$ 1500

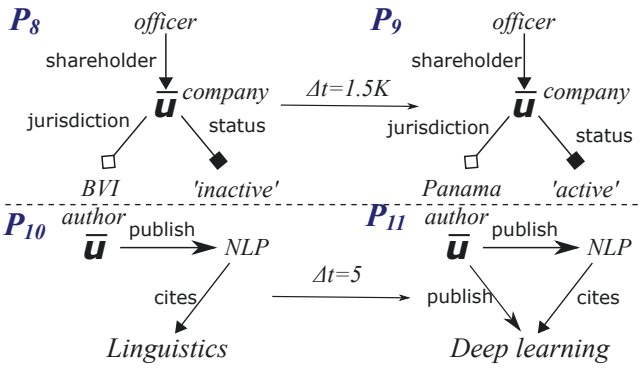


Figure 10: Real-life GTARs

days). One of such entity that support a minimal occurrence of φ_3 is "F.Geneve Project Management". Interestingly, φ_3 corresponds to a fact that when BVI cracks down the bearer shares, many companies moved bearer share clients to Panama.

(2) GTAR φ_4 = ($P_{10} \Rightarrow P_{11}, \bar{u}, \Delta t = 5$): The rule identifies a research trend that states "*Authors who publish in NLP area (Natural Language Processing) and cited linguistic papers (P_{10}) start to cite/publish deep learning papers. in* 5 *years.*" One of the authors that support the minimal occurrence of φ_4 is Chris Manning (Stanford U).

GTARs *for prediction.* We also evaluated the application of GTARs as prediction rules over "unseen" data, using Panama. We induce a temporal graph $\mathcal{G}_T$ from Panama with 75% of in total $12K$ snapshots as training dataset, and another graph $\mathcal{G}'_T$ with the rest 25% (newer) snapshots as the testing set. We mined top 10% GTARs from $\mathcal{G}_T$ using the same setting as in Table 2 for Panama, and evaluated their confidence in $\mathcal{G}'_T$ as prediction rate for P_2.

We found that these GTARs have prediction rate on average at 87%. with the highest achieving 94%. This suggests that GTARs are quite effective in predicting the events.

7 CONCLUSION

We have proposed a class of temporal association rules over graphs (GTARs) including rule model, semantics and interestingness measures. We have studied the discovery problem of GTARs and developed a mining algorithm for GTARs in temporal graphs. Our experimental study has verified that despite the enhanced expressive power of GTARs, it is feasible to find and apply GTARs in practice. We also found that GTARs can be used for activity prediction, among other applications.

We are exploring other quality metrics for GTARs, and experimenting with larger-scale real-world graphs. Another topic is to develop fast online discovery of GTARs over graph streams.

Acknowledgments. Namaki, Wu and Song are supported in part by NSF IIS-1633629 and Google Faculty Research Award. Tingjian Ge is supported in part by the NSF, under the grants IIS-1149417, IIS-1319600, and IIS-1633271.

REFERENCES

[1] *http://memeburn.com/2015/10/ddos-attacks-increasingly-used-as-smokescreen-for-other-security-threats/.*
[2] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *VLDB*, pages 336–347, 2015.
[3] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On dense pattern mining in graph streams. *VLDB*, pages 975–984, 2010.
[4] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.
[5] J. M. Ale and G. H. Rossi. An approach to discovering temporal association rules. In *SIGAPP*, pages 294–300, 2000.
[6] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He. Online frequent episode mining. In *ICDE*, pages 891–902, 2015.
[7] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In *VLDB*, pages 914–925, 2007.
[8] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis. Mining graph evolution rules. In *ECML PKDD*, pages 115–130. 2009.
[9] P. Bogdanov, M. Mongiovi, and A. K. Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, pages 81–90, 2011.
[10] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *ICDM*, 2006.
[11] X. Chen and I. Petrounias. Mining temporal features in association rules. In *PKDD*, pages 295–300. 1999.
[12] W. Fan. Graph pattern matching revised for social network analysis. In *ICDT*, 2012.
[13] W. Fan, X. Wang, Y. Wu, and J. Xu. Association rules with graph patterns. *VLDB*, pages 1502–1513, 2015.
[14] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.
[15] M. Fellendorf and P. Vortisch. Microscopic traffic flow simulator vissim. In *Fundamentals of traffic simulation*, pages 63–93. 2010.
[16] J. Gao, C. Zhou, and J. X. Yu. Toward continuous pattern detection over evolving large graph with snapshot isolation. *VLDB*, pages 1–22, 2015.
[17] X. Gao and M. P. Singh. Mining contracts for business events and temporal constraints in service engagements. *TSC*, pages 427–439, 2014.
[18] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *CSUR*, page 9, 2006.
[19] S. Gurukar, S. Ranu, and B. Ravindran. Commit: A scalable approach to mining communication motifs from dynamic networks. In *SIGMOD*, 2015.
[20] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.
[21] H.-P. Hsieh and C.-T. Li. Mining temporal subgraph patterns in heterogeneous information networks. In *SocialCom*, pages 282–287, 2010.
[22] P.-s. Kam and A. Fu. Discovering temporal patterns for interval-based events. *DaWaK*, pages 317–326, 2000.
[23] B. Liu, Y. Ma, and R. Lee. Analyzing the interestingness of association rules from the temporal dimension. In *ICDM*, pages 377–384, 2001.
[24] D. Lo and S.-C. Khoo. Mining patterns and rules for software specification discovery. *VLDB*, pages 1609–1616, 2008.
[25] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. *VLDB*, pages 310–321, 2011.
[26] S. Mansfield-Devine. The growth and evolution of ddos. *Network Security*, pages 13–20, 2015.
[27] M. H. Namaki, K. Sasani, Y. Wu, and T. Ge. Beams: Bounded event detection in graph streams. In *ICDE*, pages 1387–1388, 2017.
[28] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *ICDE*, pages 412–421, 1998.
[29] A. Paranjape, A. R. Benson, and J. Leskovec. Motifs in temporal networks. In *WSDM*, pages 601–610, 2017.
[30] F. Pennerath and A. Napoli. The model of most informative patterns and its application. In *MLKDD*, pages 205–220. 2009.
[31] C. Rainsford and J. Roddick. Adding temporal semantics to association rules. *PKDD*, pages 504–509, 1999.
[32] K. Semertzidis and E. Pitoura. Durable graph pattern queries on historical graphs. In *ICDE*, 2016.
[33] B. Shaparenko, R. Caruana, J. Gehrke, and T. Joachims. Identifying temporal patterns and key players in document collections. In *TDM*, 2005.
[34] A. Silva, W. Meira Jr, and M. J. Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *VLDB*, pages 466–477, 2012.
[35] C. Song, T. Ge, C. Chen, and J. Wang. Event pattern matching over graph streams. *VLDB*, pages 413–424, 2014.
[36] V. Srinivasan, S. Moghaddam, A. Mukherji, K. K. Rachuri, C. Xu, and E. M. Tapia. Mobileminer: Mining your frequent patterns on your phone. In *UbiComp*, pages 389–400, 2014.
[37] Q. Yuan, G. Cong, and A. Sun. Graph-based point-of-interest recommendation with geographical and temporal influences. In *CIKM*, 2014.
[38] Y. Zhang, Q. Ji, and H. Lu. Event detection in complex scenes using interval temporal constraints. In *ICCV*, pages 3184–3191, 2013.