Event Pattern Discovery by Keywords in Graph Streams

Mohammad Hossein Namaki*, Peng Lin*, Yinghui Wu*†
Washington State University, Pacific Northwest National Laboratory*†
{*mnamaki, plin1, yinghui*}*@eecs.wsu.edu*

***Abstract*—Given an evolving network and a set of user-specified keywords, how to discover and maintain the active events specified by the keywords? In this paper, we study the problem of event pattern discovery by keywords in graph streams. (1) We propose a class of event patterns to capture events relevant to user-specified keywords, by integrating (approximate) topological and value bindings from keywords. We also introduce an activeness measure, to balance the pattern expressiveness and the cost of pattern discovery. (2) We develop both *from-scratch* and *incremental* algorithms to discover and maintain active events in graph streams. Using real-world graph streams, we experimentally verify the effectiveness of the event pattern model and the efficiency of our from-scratch and incremental algorithms.**

I. Introduction

Event detection is a critical task for decision making over evolving networks such as social networks , communication networks [15], and brain connectomes [2]. Conventional methods discover events from item or tuple streams [8], [12]. Complex network events and their occurrences are, nevertheless, usually characterized by *graph patterns* [3], [15] and their matches, which contain heterogeneous entities and temporal interactions. On the other hand, keyword queries are routinely used to access and understand graph data. Given a (possibly infinite) graph stream $\mathcal{G}_T = \{G_0, G_1, \ldots\}$, where each G_i is a snapshot, and a set of keywords $\mathcal{K}$ that specifies user's interest, the *event pattern discovery by keywords* is to discover event patterns Σ from $\mathcal{G}_T$ that are active and relevant to keywords $\mathcal{K}$.

The need for discovering event patterns is evident. Consider the following examples from real-world applications.

Cyber attack detection. Recent studies suggest that the Distributed Denial of Service (DDoS) attacks coincide with "more than 45% (resp. 32%) of malware incidents (resp. network intrusions)" [7], and are frequently used as a decoy to distract IT defense effort. Such a "masked attack" consists of two event patterns illustrated in Fig. 1. (1) Pattern P_1 describes the DDoS attack, where an attacker performs multiple port scans to find vulnerable bot servers (edge attacker, bot), and controls them to create massive requests to victim host x (edge bot, x). (2) Pattern P_2 describes information exfiltration, where victim x takes commands from bot (edge x, bot) and exchanges data with compromised websites that lead to data breach. Detecting frequent occurrences of P_1 and P_2 at servers that match "host" can suggest active

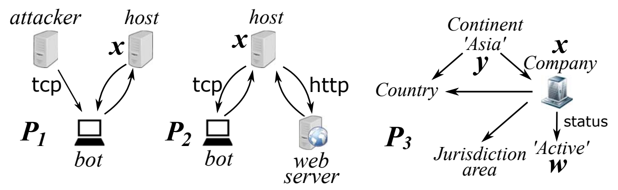


Figure 1: Real-world active events

masked attacks. The servers that match node x of both P_1 and P_2 can be suggested as victims of masked attacks.

Activity analysis. The Offshore dataset[1] contains entities and their financial activities. An active pattern P_3, relevant to keywords "Asia", "Active", and "Company", identifies *"jurisdiction area (a tax haven) selected by many active Asian companies since 1965"* as "*British Virgin Island*".

Although desirable, discovering graph patterns is more challenging than its counterparts over item streams [8], [12]. (1) Conventional graph patterns and frequency measures should be extended to quantify the activeness of events specified by keywords in graph streams. (2) It requires fast pattern mining against graph streams, which is already expensive over their static counterpart [4].

We propose to extend conventional graph patterns to *event patterns* to support event detection with keywords. We show that event pattern discovery by keywords is feasible over large graph streams by developing a *from-scratch* algorithm, which discovers events from scratch, and an *incremental* algorithm, which dynamically maintains the discovered events. Given event patterns Σ computed by a from-scratch algorithm $\mathcal{A}$, it incrementally updates Σ with a cost determined by a bounded number of pattern verifications.

Related work. We categorize the related work as follows. Event detection. Event detection has been studied over item and tuple streams. Events are modeled as consecutive occurrences of items, *e.g.,* episodes [12]. Top-k monitoring aims to answer top-k queries over a single multidimensional (tuple) stream. A general approach [8] assumes a monotone function on attributes with sorted access, and maintains k-skyband (tuples dominated by at most $k-1$ others) to find top-k tuples. In contrast, (1) We study event patterns as general graph patterns, which is more involved than itemsets and tuples; (2) The activeness measures for events are not

[1]https://offshoreleaks.icij.org/pages/database

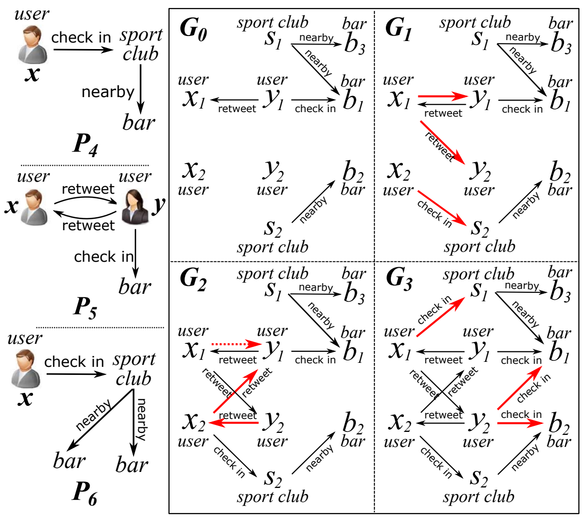


Figure 2: Social graph stream and events

necessarily monotone functions; as in [8] over graph streams. Incremental event discovery is also not addressed.

Temporal graph mining. Temporal models have been studied for mining dynamic networks. These include subgraph isomorphism [3], [16], and quasi-cliques [14]. By contrast, (1) similar to [11], we adopt approximate pattern matching to capture meaningful event occurrences, instead of using strict subgraph isomorphism [3], [16]. This also reduces verification cost, paving the way to feasible pattern mining over large graph streams. (2) We introduce incremental mining with bounded cost.

Incremental computation. Incremental querying are studied in [5] for path and pattern queries. An incremental algorithm $\mathcal{A}$ is bounded if it incurs a cost that is determined by the size of query Q, the changes of the input ΔG, and results $\Delta Q(G)$. This paper makes the first effort to study incremental boundedness in the context of *pattern mining*.

II. Preliminaries

Graph stream. A graph stream $\mathcal{G}_T$ is a sequence of (possibly infinite) *snapshots*, where a snapshot G_i at time i is a directed graph (V_i, E_i, L) that contains a node set V_i and edge set $E_i \subseteq V_i \times V_i$. Each node $v \in V_i$ (resp. edge $e \in E_i$) has a label $L(v)$ (resp. $L(e)$) that describes its content (*e.g.,* name, type, attribute values, relation).

Equivalently, $\mathcal{G}_T$ can be represented as a sequence of batch updates, where (1) an update is either an insertion (denoted as e^+) or deletion (denoted as e^-) of an edge e, and (2) edge updates ΔE_i yields snapshot G_i when applied to its previous snapshot G_{i-1}.

Example 1: Fig. 2 illustrates a fraction of a social network stream $\mathcal{G}_T$ with four snapshots $\{G_0, \ldots, G_3\}$. Each snapshot contains entities with labels user (*e.g.,* x_1, y_1), sport club (s_1, s_2), and bar (b_1, b_2), as well as their temporal interactions (retweet, check in, via *e.g.,* Facebook Place) and geographical information (nearby). □

Event patterns. An *event pattern* P is a connected graph (V_P, E_P, L_P), with a set of pattern nodes V_P, pattern edges E_P, and a function L_P that assigns a label $L_P(v_p)$ (resp. $L_P(e_p)$) to each pattern node $v_p \in V_P$ (resp. $e_p \in E_P$). Moreover, P *pertains* to a set of keywords $\mathcal{K}$=$\{k_1, \ldots, k_n\}$, if there exists a set of *keyword* nodes V_K=$\{u_1, \ldots, u_n\} \subseteq V_P$ such that $L_P(u_i) = k_i$ ($i \in [1, n]$).

As not every keyword node in P indicates entities of interest (*e.g.,* keyword "Active" simply specifies status values of companies in P_3), P also carries a set of designated *focus* nodes $\bar{x} \subseteq V_K$, which tracks entities of users' interest.

Example 2: Consider patterns P_4 and P_5 by keywords "user" and "bar" illustrated in Fig. 2 that capture a user x influenced by another user y via temporal social influence, which states that "if users x and y who checked in at a bar retweet each other, and x checks in at a nearby sport club, then he may visit the same bar." While both x and y in P_5 are keyword nodes for "user", one can specify x as a focus to recommend potential customers for the bar. □

Event occurrence. We extend dual simulation [6], [9], [15], an established model, to capture occurrences of event patterns with keyword nodes.

Given an event pattern P=(V_P, E_P, L_P) with focus $\bar{x}$, and a snapshot G_i=(V_i, E_i, L) in a graph stream $\mathcal{G}_T$,

(1) A node $v \in V_i$ is a candidate of a node $u \in V_P$ if $L_P(u)$=$L(v)$. An edge $e = (v, v') \in E_i$ is a candidate of edge $e' = (u, u') \in E_P$ if v and v' are candidates of u and u', respectively, and $L_P(e)$=$L(e)$.

(2) Event pattern P *occurs* in $\mathcal{G}_T$ at timestamp i if there exists a nonempty relation $R(P, G_i) \subseteq V_P \times V_i$ such that

- for each $(u, v) \in R(P, G_i)$, v is a candidate of u; and
- for each $u \in V_P$, there exists a match $v \in V_i$ such that (a) $(u, v) \in R(P, G_i)$; (b) for each edge $(u, u') \in E_P$, there is an edge match $(v, v') \in E_i$ with $(u', v') \in R(P, G_i)$; and, moreover, (c) for each edge $(u'', u) \in E_P$, there is an edge match $(v'', v) \in E_i$ with $(u'', v'') \in R(P, G_i)$.

We use the following notions. (1) The *occurrence* of P at time i in $\mathcal{G}_T$, denoted as $\mathsf{occ}(P, G_i)$, is the subgraph of G_i induced by all the node and edge matches in $R(P, G_i)$. (2) The *focus occurrence* (denoted as $\mathsf{occ}(P(\bar{x}), G_i)$) is the match set $\{v | (u, v) \in R(P, G_i), u \in \bar{x}\}$.

It has been shown that a unique, maximum dual simulation can be computed in $O((|V_P| + |V|)(|E_P| + |E|))$ time [6], [10]. The tractable results carry over to compute the occurrences of P, in contrast to its intractable counterparts in terms of subgraph isomorphism [3], [16].

III. Event Discovery in Graph Streams

To discover active events in $\mathcal{G}_T$ as meaningful regularities, we introduce an activeness measure. Let $\{G_0, \ldots, G_T\}$ specifies a finite sequence of the snapshots that are already seen, where timestamp T denotes the "current" time.

Active events. Given event pattern P and graph stream $\mathcal{G}_T$ = $\{G_0, \ldots, G_T\}$, the *activeness* of P in $\mathcal{G}_T$ is defined as:

$$\textbf{Act}(P, \mathcal{G}_T) = \sum_{i \in [0,T]} \alpha^{T-i} \frac{|\mathsf{occ}(P(\bar{x}), G_i)|}{|V_i|}$$

where (a) $\mathsf{occ}(P(\bar{x}), G_i)$ is the focus occurrence of P at time i (specified by keywords), normalized with $|V_i|$ of the snapshot G_i, and (b) $\alpha \in (0, 1]$ refers to a "forgetting factor". That is, $\textbf{Act}(\cdot)$ favors events with more accumulated focus occurrences and more "recent" occurrences. An event is *active* over $\mathcal{G}_T$ *w.r.t.* a threshold θ if $\textbf{Act}(P, \mathcal{G}_T) \geq \theta$. Otherwise, it is *inactive*.

Active events can be of no interests if (1) they are too small, *e.g.,* routine TCP connections as a single-edge event; or (2) they contain pattern nodes that always match the same set of nodes in $\mathcal{G}_T$. To balance the informativeness and conciseness, we introduce *canonical event patterns*.

Ordering of patterns. An event pattern P_1 with focus $\bar{x_1}$ =$\{u_1, \ldots, u_n\}$ is *embedded* in another event pattern P_2 with the same set of focus renamed as $\bar{x_2}$=$\{u'_1, \ldots, u'_n\}$, denoted as $P_1 \preceq P_2$, if (1) there exists a dual simulation R from P_1 to P_2, and (2) $(u_i, u'_i) \in R$ for $i \in [1, n]$. P_1 and P_2 are *equivalent* (denoted as $P_1 \sim P_2$), if $P_1 \preceq P_2$ and $P_2 \preceq P_1$.

The result below verifies that event patterns with focus preserve anti-monotonicity in terms of their ordering.

Lemma 1: *For any graph stream $\mathcal{G}_T$ and two events P_1 and P_2, (a) $\textbf{Act}(P_2, \mathcal{G}_T) \leq \textbf{Act}(P_1, \mathcal{G}_T)$ if $P_1 \preceq P_2$; and (b) $\textbf{Act}(P_1, \mathcal{G}_T) = \textbf{Act}(P_2, \mathcal{G}_T)$ if $P_1 \sim P_2$.* □

We present the detailed proof in [1].

Canonical patterns. Consider a "self" matching R^P from event pattern P to itself. Two nodes u and u' in $P(\bar{x})$ are *equivalent* if $(u, u') \in R^P$ and $(u', u) \in R^P$. We say P is *canonical*, if there exists no equivalent node pair in P. Indeed, equivalent pattern nodes always specify the same set of matches for any $\mathcal{G}_T$, thus should be avoided.

Maximal patterns. A canonical pattern P is maximal *w.r.t.* a threshold θ, if there exists no canonical event pattern P' ($|P'|>|P|$) that pertains to the same focus and is obtained by adding edges to P, such that P' is active *w.r.t.* θ. It is a b-maximal pattern if P has at most b edges.

We prefer active patterns that are both canonical and maximal, which are both concise and informative.

Example 3: Consider event patterns P_4-P_6 in Fig. 2. One may verify the following. (1) P_4 and P_5 are canonical patterns. (2) P_6 is not canonical. (3) $P_4 \sim P_6$; and P_4 is a "minimal" representation of P_6. □

From-scratch discovery problem. Given a graph stream $\mathcal{G}_T$, an activeness threshold $\theta \in [0, 1]$, user-specified keywords $\mathcal{K}$ and a size bound b ($b \geq |\mathcal{K}| - 1$), compute all the b-maximal canonical event patterns Σ *w.r.t.* θ over $\mathcal{G}_T$.

Incremental discovery problem. Given $\mathcal{G}_T$ and a set of active event patterns Σ^{i-1} at time i-1, and a newly arrived snapshot G_i at time i, *incrementally* update Σ^{i-1} to Σ^i without computing from scratch, and report Σ^i on request by a user.

IV. From-scratch Discovery

We first present an algorithm, denoted as BDis, to discover event patterns from scratch. Underlying the algorithm BDis is the maintenance of a pattern lattice $\mathcal{T}$ that controls the generation of patterns. Each pattern is associated with its activeness and candidates in the latest snapshot G_T of $\mathcal{G}_T$.

From-scratch algorithm. We outline algorithm BDis below.
(1) Given $\mathcal{G}_T$ and keywords $\mathcal{K}$, BDis initializes $\mathcal{T}$ with all events that pertain to $\mathcal{K}$ (a set of independent keyword nodes) and verifies their activeness in $\mathcal{G}_T$ and updates $\mathcal{T}$.
(2) At each level j of lattice $\mathcal{T}$, (a) BDis spawns a set of new patterns at level $j+1$, where each new pattern $P'(\bar{x})$ is obtained by adding a single pattern edge e_P=(u', u) to a verified pattern $P(\bar{x})$, with either u' or u in $P(\bar{x})$, and (b) it verifies if P' is a canonical b-maximal pattern by invoking a procedure Verify. Once all level-$j+1$ events patterns are verified, it finds the maximal canonical ones at level-j by definition, and updates Σ when necessary.

The algorithm BDis stops spawning from maximal canonical event patterns or inactive ones (Lemma 1). It returns Σ if no new patterns can be spawned.

Pattern verification. We "plug-in" an operator Verify to the level-wise mining framework. Given an event pattern P, Verify performs two steps.
(1) It computes the self matching R^P by invoking the pattern matching algorithm in [6], and checks if P is canonical by definition (Section III). This operation subsumes the validation of whether P is connected.
(2) If P is canonical, it then invokes the algorithm in [6] to compute the focus occurrences of P in each snapshot G_i of $\mathcal{G}_T$. The overall activeness $\textbf{Act}(P, \mathcal{G}_T)$ and the matches of P is then aggregated.

Example 4: Fig. 3 illustrates a fraction of event lattice $\mathcal{T}^1$ over $\mathcal{G}_1$ =$\{G_0, G_1\}$ in Fig. 2. Given the size bound b = 3, BDis discovers P_4, P_5 in $\mathcal{T}^1$, which are two of verified maximal canonical events *w.r.t.* θ=0.11. □

V. Incremental Discovery

Recomputing event patterns Σ each time a new snapshot arrives with BDis is not practical for large $\mathcal{G}_T$. We present an incremental algorithm that correctly updates Σ by (1) only accessing *two* snapshots, and (2) incurring a bounded cost determined by necessary computation. These justify its feasibility in space and time.

Necessary computation. We start with a characterization of "necessary" computation cost, using BDis as a "yardstick" from-scratch algorithm. To this end, we associate auxiliary structure to the data structures used by BDis.
(1) Recall the pattern lattice $\mathcal{T}$. Each event pattern P has a status, which encodes (a) its activeness, (b) its candidates

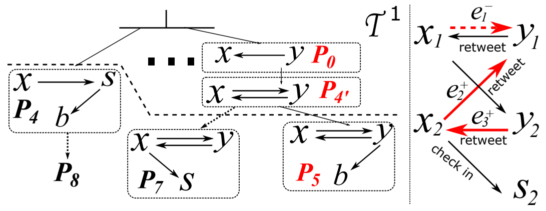


Figure 3: Event pattern lattice and affected area

and matches in the latest seen snapshot in $\mathcal{G}_T$, and (c) one of the three values below: *inactive or unverified* (encoded as '**I**'), if P has activeness smaller than θ or unknown; *active and not maximal* ('**A**'), and *maximal* ('**M**'). These structures are constructed only when needed.

(2) Each node v (resp. edge e) in $\mathcal{G}_T$ has a status matrix $v.\mathsf{M}$ (resp. $e.\mathsf{M}$), where each entry $v.\mathsf{M}[P][u]$($e.\mathsf{M}[P][e_P]$) is a flag that indicates whether it is a match, candidate or irrelevant (with a different label) of node u (edge e_P) in a pattern P. These status matrices are only used to characterize the cost, and are not physically constructed.

Affected mining area. We consider the behavior of BDis at two consecutive timestamps $i-1$ and i. (1) At time $i-1$, BDis mines active event patterns Σ^{i-1} over graph stream $\mathcal{G}_{i-1}$=$\{G_0, \ldots, G_{i-1}\}$ and terminates with pattern lattice $\mathcal{T}^{i-1}$. (2) At time i, BDis mines Σ^i from scratch over updated stream $\mathcal{G}_i = \mathcal{G}_{i-1} \cup \{G_i\}$, and yields lattice $\mathcal{T}^i$.

The *affected mining area*, denoted as AFF, characterizes the total "changes" of the data inspected by BDis between time $i-1$ and i, and is defined as:

$$\mathsf{AFF} = \mathsf{AFF}_{\mathcal{T}} \cup \mathsf{AFF}_{\mathcal{G}}$$

where (1) $\mathsf{AFF}_{\mathcal{T}}$ is a set of *affected patterns* with their status updated (including those newly introduced) in $\mathcal{T}^i$; and (2) $\mathsf{AFF}_{\mathcal{G}}$ includes all the nodes and edges with status updated (including those newly introduced) in G_i.

The affected mining area AFF indicates a "mandatory" fraction of data that must be inspected, *i.e.,* the changes of data and structures inspected by a from-scratch counterpart. To characterize such cost, we use extended size of AFF, and denote as $\|\mathsf{AFF}_{\mathcal{T}}\|_r$ (resp. $\|\mathsf{AFF}_{\mathcal{G}}\|_r$) the number of of patterns in $\mathsf{AFF}_{\mathcal{T}}$ (resp. number of nodes and edges in $\mathsf{AFF}_{\mathcal{G}}$) and their r-hop neighbors in $\mathcal{T}^i$ (resp. G_i).

Theorem 2: *Given pattern size bound b, threshold θ, active event pattern Σ and a new snapshot G_i, there is an incremental mining algorithm that correctly updates Σ in $O(\|\mathsf{AFF}_{\mathcal{T}}\|_1(b^2 + b\|\mathsf{AFF}_{\mathcal{G}}\|_b + \|\mathsf{AFF}_{\mathcal{G}}\|_b^2))$ time, by only accessing two snapshots G_i and G_{i-1}, at any time i.* □

That is, the incremental algorithm incurs bounded cost determined by the affected area and size bound b only. To prove Theorem 2, we next introduce an incremental mining algorithm, denoted as IncDis, with the bounded cost.

Bounded incremental mining. Not knowing exact $\mathsf{AFF}_{\mathcal{T}}$ and $\mathsf{AFF}_{\mathcal{G}}$, the algorithm IncDis maintains a set of affected patterns $\mathsf{AFF}'_{\mathcal{T}}$, and a set of affected nodes and edges $\mathsf{AFF}'_{\mathcal{G}}$ in G_{i-1}, as "over-estimated" $\mathsf{AFF}_{\mathcal{T}}$ and $\mathsf{AFF}_{\mathcal{G}}$, respectively. It only updates the activeness of the patterns in $\mathsf{AFF}'_{\mathcal{T}}$, and for each pattern, it only visits a fraction of G_{i-1} up to $\mathsf{AFF}'_{\mathcal{G}}$.

To see Theorem 2, it suffices to guarantee the following two invariants: (**I$_1$**): $|\mathsf{AFF}'_{\mathcal{T}}|$ is in $O(\|\mathsf{AFF}_{\mathcal{T}}\|)$, and (**I$_2$**): $|\mathsf{AFF}'_{\mathcal{G}}|$ is in $O(\|\mathsf{AFF}_{\mathcal{G}}\|_b)$. Indeed, given the invariants, the total cost is $O(|\mathsf{AFF}'_{\mathcal{T}}|(b + \|\mathsf{AFF}_{\mathcal{G}}\|_b)^2)$, where $O(b + \|\mathsf{AFF}_{\mathcal{G}}\|_b)^2)$ is the verification cost (guaranteed by dual simulation matching; see Section III). This is bounded by $O(\|\mathsf{AFF}_{\mathcal{T}}\|(b^2 + b\|\mathsf{AFF}_{\mathcal{G}}\|_b + \|\mathsf{AFF}_{\mathcal{G}}\|_b^2))$.

Identifying affected patterns. The operator findAFF answers the question "*what are the patterns that need to be verified to update Σ, given newly verified patterns $\mathcal{P}$ over $\mathcal{G}_i$?*" Given $\mathcal{P}$, it updates Σ with $\mathcal{P}$, and "propagates" $\mathsf{AFF}'_{\mathcal{T}}$ with a set of new patterns $\mathcal{P}'$ to be verified by incVerify.

More specifically, for each newly verified pattern $P \in \mathcal{P}$, it propagates $\mathsf{AFF}'_{\mathcal{T}}$ by taking two actions below.

Downward propagation: If $\mathbf{Act}(P, \mathcal{G}_i) \geq \theta$ (*i.e.,* with status '**A**'), it updates $\mathsf{AFF}'_{\mathcal{T}} := \mathsf{AFF}'_{\mathcal{T}} \cup P^+$, where P^+ refers to the patterns obtained by adding an edge to P within size bound b, *i.e.,* the "children" of P in $\mathcal{T}^i$. That is, it explores larger event patterns in $\mathcal{T}^i$ as P remains to be active.

Upward propagation: If $\mathbf{Act}(P, \mathcal{G}_i) < \theta$ (*i.e.,* with status changed to '**I**'), it updates $\mathsf{AFF}'_{\mathcal{T}} := \mathsf{AFF}'_{\mathcal{T}} \cup P^-$, where P^- refers to the "parents" of P in $\mathcal{T}^i$, obtained by removing an edge from P. That is, it explores smaller event patterns that may later become new b-maximal ones, due to that P becomes an inactive pattern.

Using $\mathcal{T}^{i-1}$, both P^+ and P^- can be constructed without reconstructing $\mathcal{T}^i$. The operator findAFF terminates downward (resp. upward) propagation at pattern P if P^+ (resp. P^-) is $\emptyset$ due to size bound or inactiveness of its children. It inserts P to Σ as a newly discovered b-maximal pattern.

Coping with new patterns. Both propagation actions guarantee that any new pattern not in $\mathcal{T}^i$ in the propagation is included to $\mathsf{AFF}'_{\mathcal{T}}$. Note that such patterns must be in $\mathsf{AFF}_{\mathcal{T}}$.

Lemma 3: *When* IncDis *terminates, (1)* $\mathsf{AFF}_{\mathcal{T}} \subseteq \mathsf{AFF}'_{\mathcal{T}}$, *and (2)* $|\mathsf{AFF}'_{\mathcal{T}}|$ *is in* $O(\|\mathsf{AFF}_{\mathcal{T}}\|_1)$. □

Thus, findAFF guarantees invariant **I$_1$** (see [1] for proof).

Incremental verification. Operator incVerify updates the activeness of patterns $\mathcal{P}'$ by identifying and accessing $\mathsf{AFF}'_{\mathcal{G}}$ only. It initializes $\mathsf{AFF}'_{\mathcal{G}}$ as $\|\Delta E_i\|_b$, *i.e.,* the set of edges ΔE_i and their b-hop neighbors in G_i. If $\mathcal{P}'$ is $\emptyset$, it initializes $\mathcal{P}'$ as a set of patterns in Σ, which have candidates inserted or matches removed in ΔE. Note that this also initializes $\mathsf{AFF}'_{\mathcal{T}}$ as $\mathcal{P}'$ when findAFF takes over.

It then copes with two cases for each pattern $P \in \mathcal{P}'$.

(1) If P has no edge candidate in $\mathsf{AFF}'_{\mathcal{G}}$ (*i.e.,* b-hop neighbors of the updated edges ΔE), then $\mathsf{occ}(P, G_{i-1})$ = $\mathsf{occ}(P, G_i)$.

(2) Otherwise, incVerify solves an *incremental query processing* problem to update $\mathsf{occ}(P, G_i)$ by only accessing necessary part of $\mathsf{AFF}'_{\mathcal{G}}$. (a) It performs necessary canonical

```
Algorithm IncDis
Input: latest snapshot $G_{i-1}$, new snapshot $G_i$,
       $b$-maximal events $\Sigma^{i-1}$, lattice $\mathcal{T}^{i-1}$, $b$, and threshold $\theta$,
Output: updated event patterns $\Sigma^i$.
1.  initialize $\Sigma^i$:=$\Sigma^{i-1}$, $\mathcal{T}^i$=$\mathcal{T}^{i-1}$, set $\mathsf{AFF}'_{\mathcal{G}}$=$\emptyset$; set $\mathsf{AFF}'_{\mathcal{T}}$=$\emptyset$;
2.  initializes $\mathsf{AFF}'_{\mathcal{G}}$ and $\mathcal{P}$ with incVerify;
3.  while $\mathcal{P}$ is not $\emptyset$ do
        /* finding affected patterns */
4.      $\mathcal{P}$ := findAFF ($\mathcal{P}$, $\mathcal{T}^i$, $\Sigma^i$);
5.      $\mathsf{AFF}'_{\mathcal{T}}$ := $\mathsf{AFF}'_{\mathcal{T}} \cup \mathcal{P}$;
        /* incremental verification */
6.      incVerify ($\mathcal{P}$, $\mathsf{AFF}'_{\mathcal{G}}$); update $\Sigma_i$;
7.  return $\Sigma^i$;
```

Figure 4: Algorithm IncDis

checking of P as in its batch counterpart Verify (Section IV). (b) For canonical pattern P, it induces a fraction $\|\Delta E(P)\|_b \subseteq \mathsf{AFF}'_{\mathcal{G}}$, where $\Delta E(P)$ includes insertions of P's candidates, and deletions of its matches in G_i ($\Delta E(P) \subseteq \Delta E_i$). (c) It then recomputes the matches of P against $\|\Delta E(P)\|_b$ (instead of G_i) by removing invalid matches and inserting new occurrences, as in its from-scratch counterpart. It finally updates $\mathbf{Act}(P, \mathcal{G}_T \cup \{G_i\})$.

Lemma 4: *When* IncDis *terminates, (1)* $\mathsf{AFF}_{\mathcal{G}} \subseteq \mathsf{AFF}'_{\mathcal{G}}$, *and (2)* $|\mathsf{AFF}'_{\mathcal{G}}|$ *is in* $O(\|\mathsf{AFF}_{\mathcal{G}}\|_b)$. □

incVerify guarantees Invariant $\mathbf{I_2}$ (detailed proof in [1]).

Algorithm IncDis. Upon receiving snapshot G_i, IncDis (Fig. 4) first invokes incVerify to initialize $\mathsf{AFF}'_{\mathcal{G}}$ and $\mathcal{P}$ (line 2). It then interleaves operators findAFF and incVerify in a "superstep" (lines 3-6) to track and update the activeness of the affected patterns $\mathsf{AFF}'_{\mathcal{T}}$ and updates Σ only when necessary. The interaction step repeats until no new affected patterns can be identified ($\mathcal{P}$ is $\emptyset$; line 3), *i.e.,* all the affected patterns are inspected. It then returns the updated Σ^i.

We present the detailed algorithms in [1].

VI. Experimental evaluation

Using real-world graph streams, we experimentally evaluate (1) the performance of the algorithms BDis and IncDis, and (2) a case study of real-world event patterns.

Experimental Setting. We used the following setting.

Datasets. We use three real-world datasets. (1) Citation[2] 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 dates. (2) IDS [13] records daily intrusion activities over a cybernetwork. We induced a graph stream with 12 snapshots, where each snapshot contains 3.7M entities (*e.g.,* alert, host, logs) and 5.6M relations (*e.g.,* application, protocol), with 34 labels. (3) Offshore (Section I) contains in total $839K$ offshore entities (*e.g.,* companies, countries, people), 3.6M relationships (*e.g.,* establish, close) and 433 labels, with 12 snapshots covering 40 years of offshore entities and financial activities.

[2]https://aminer.org/citation

We extracted keywords randomly from node labels and properties with a proper candidate size (300-1000).

Algorithms. We have implemented the algorithms below in Java: (1) Algorithm BDis (including operator Verify, Section IV), (2) Algorithm IncDis (including operator incVerify, Section V), compared with: (a) IncDisn, its counterpart that processes ΔE_i in multiple batches of a tunable buffer size n; and (b) IncSub, a counterpart of IncDis that mines event patterns as frequent subgraphs (by replacing incVerify with VF2, a subgraph matching algorithm);

We ran all our tests 3 times on a Linux machine powered by an Intel 2.30 GHz CPU with 64 GB of memory.

Experimental results. We report the details of our findings.

Exp-1: Performance of BDis. Our first observation is that it is quite feasible to discover active event patterns over large graph streams. Over Offshore, Citation and IDS with in total $2.5M$, $17.3M$ and $9.3M$ nodes and edges, it takes BDis less than 6.5, 3.1 and 1.4 minutes to find all 5-maximal active event patterns. It takes more time over denser and more diversified graphs, as expected. In contrast, even the incremental version of subgraph mining IncSub takes more than 1000 seconds over the smallest snapshot of Offshore with $|\Delta E| = 10K$, and does not run to completion in other datasets. We hence omit the performance of IncSub.

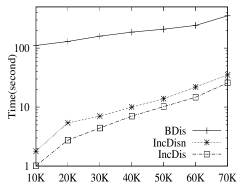
Exp-2: Performance of IncDis. We evaluate the performance of IncDis, compared to BDis, IncDisn and IncSub.

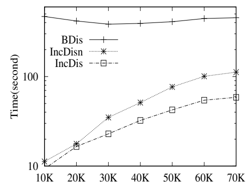
Varying ΔE. We report the impact of $|\Delta E|$ to IncDis over Citation, Offshore, and IDS, in Fig 5(a)-Fig. 5(c), respectively. We set buffer size n=$1K$ for IncDisn.

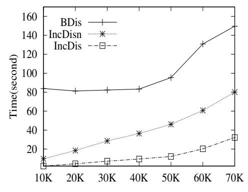
For Citation, we start with G_0=$(4.3M, 13M)$, and vary the size of ΔE_0 from $10K$ to $70K$. Fig. 5(a) tells us the following. (1) IncDis and IncDisn further improve BDis by 21.1 and 14.62 times, respectively. Indeed, both explore a bounded affected area. (2) IncDis is feasible over large graph streams. It takes 7.0 seconds to maintain active patterns over $40K$ updates. (3) Both IncDis and IncDisn take more time with larger $|\Delta E|$ due to larger affected mining area. IncDis is on average 1.44 times faster. This is due to more patterns can be pruned when more updates are "foreseen" for IncDis.

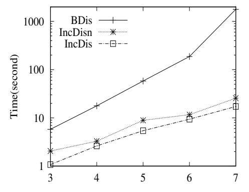
Fig. 5(b) (resp. Fig. 5(c)) tells us that IncDis is on average 12.55 and 1.71 (resp. 8.04 and 3.19) times faster than BDis and IncDisn, respectively, over Offshore (resp. IDS). IncDis improves BDis better for Citation than Offshore and IDS. This is because from-scratch computation is relatively more expensive over denser and more heterogeneous graphs.

Varying b. We next evaluate the impact of the size bound b. Fixing $|\Delta E| = 40K$, we vary b from 3 to 7 over Citation. As shown in Fig. 5(d), all the algorithms take more time with larger b, due to more event patterns need to be verified. On the other hand, both IncDis and IncDisn are less sensitive


(a) IncDis: Citation


(b) IncDis: Offshore


(c) IncDis: IDS


(d) Varying b: Citation

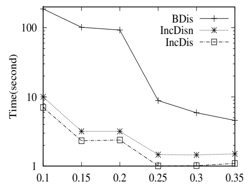

(e) Varying θ : Citation

Figure 5: Performance evaluation

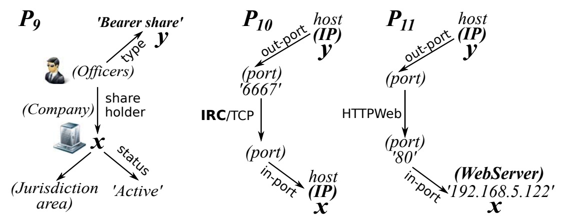


Figure 6: Real-life event patterns (Offshore & IDS)

compared to BDis, due to that they only verify affected patterns rather than all the possible ones.

Varying θ. Fixing $|\Delta E|$=40K, we vary θ from 0.1 to 0.35. As shown in Fig. 5(e), all the algorithms take less time with larger θ, since more patterns can be pruned. Again, IncDis and IncDisn are less sensitive compared to BDis, due to that the affected patterns are less sensitive to the change of θ.

Exp-3: Effectiveness. We inspected the top active event patterns and their matches discovered by IncDis.

Real-world events. We report 3 active event patterns (Fig. 6). (1) Event P_9 (Offshore) pertains to keywords "Bearer share" and "Company" (focus) along with its matches reveals a significant move of active bearer shares companies (which are not required to register owner's name) in 2005 from tax heaven "British Virgin Islands" to "Panama", due to that the former cracked down on bearer shares.

(2) Event patterns P_{10} and P_{11} capture two active communication patterns in IDS. P_{10} pertains to keyword and focus "IP" with x and y as focus indicates frequent communication via internet relay chat, as indicated by the edge labeled IRC. P_{11} carries keywords and focus "IP" and "WebServer" finds potential attack net flows.

Event analysis. By issuing online query supported by IncDis, we show the activeness of patterns P_{10} and P_{11} over a stream of 80 snapshots with 5-minutes interval in Fig. 7. We found that the peaks of their activeness trace back to active upload of IRC softwares to targeted hosts (P_{10}) by attackers, and a trend of DDoS attacks (P_{11}) that takes advantage of IRC botnets [13]. Interestingly, larger α (hence more weights on recent events) helps to "amplify" such trend. This suggests practical applications of our methods in event analysis.

VII. Conclusions

We have introduced a class of event patterns to capture approximate occurrence of events over graph streams. We have developed both from-scratch and incremental algorithms to discover event patterns. We have shown that the incremental algorithm incurs a bounded cost determined by necessary amount of computation. Our experimental study has verified the efficiency and effectiveness of the algorithms.

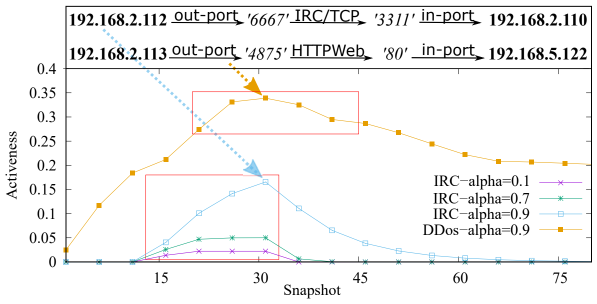


Figure 7: Activeness of P_{10} (IRC) and P_{11} (DDos) in IDS over 5-minutes (80 snapshots)

Acknowledgments. Namaki and Wu are supported in part by NSF IIS-1633629 and Google Faculty Research Award.

References

[1] Full version. *http://eecs.wsu.edu/%7emnamaki/BigData17.pdf.*

[2] M. Bola and B. A. Sabel. Dynamic reorganization of brain functional networks during cognition. *Neuroimage*, pages 398–413, 2015.

[3] S. Choudhury, L. Holder, G. Chin, A. Ray, S. Beus, and J. Feo. Streamworks: a system for dynamic graph search. In *SIGMOD*, pages 1101–1104, 2013.

[4] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *PVLDB*, pages 517–528, 2014.

[5] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.

[6] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. *PVLDB*, pages 310–321, 2011.

[7] S. Mansfield-Devine. The growth and evolution of ddos. *Network Security*, 2015, 2015.

[8] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, 2006.

[9] M. H. Namaki, K. Sasani, Y. Wu, and T. Ge. Beams: Bounded event detection in graph streams. In *ICDE*, pages 1387–1388, 2017.

[10] M. H. Namaki, K. Sasani, Y. Wu, and A. H. Gebremedhin. Performance prediction for graph queries. In *NDA*, 2017.

[11] M. H. Namaki, Y. Wu, Q. Song, P. Lin, and T. Ge. Discovering temporal graph association rules. In *CIKM*, 2017.

[12] M. Qin and K. Hwang. Frequent episode rules for intrusive anomaly detection with internet datamining. In *USENIX Security*, 2004.

[13] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, pages 357 – 374, 2012.

[14] A. Silva, W. Meira Jr, and M. J. Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *PVLDB*, 2012.

[15] C. Song, T. Ge, C. Chen, and J. Wang. Event pattern matching over graph streams. *PVLDB*, pages 413–424, 2014.

[16] B. Zong, X. Xiao, Z. Li, Z. Wu, Z. Qian, X. Yan, A. K. Singh, and G. Jiang. Behavior query discovery in system-generated temporal graphs. *VLDB*, pages 240–251, 2015.