PADUA: Parallel Architecture to Detect Unexplained Activities

CRISTIAN MOLINARO, University of Calabria
VINCENZO MOSCATO, ANTONIO PICARIELLO, University of Naples "Federico II"
ANDREA PUGLIESE, ANTONINO RULLO, University of Calabria
V. S. SUBRAHMANIAN, University of Maryland

There are numerous applications (e.g., video surveillance, fraud detection, cybersecurity) in which we wish to identify unexplained sets of events. Most past work has been domain-dependent (e.g., in video surveillance or cybersecurity) and much of it focused on the valuable class of *statistical anomalies* in which statistically unusual events are considered. In contrast, assume that there is a set $\mathcal{A}$ of known activity models (both harmless and harmful) and a log L of time-stamped observations. We define a part $L' \subseteq L$ of the log to represent an *unexplained situation* when none of the known activity models can explain L' with a score exceeding a user-specified threshold. We represent activities via the notion of a *probabilistic penalty graph* (PPG) and show that a set of PPGs can be combined into one Super-PPG. We define an index structure for Super-PPGs. Given a compute cluster of $(K+1)$ nodes (one of which is a master node), we show how to split a Super-PPG into K subgraphs that can be autonomously processed by K compute nodes. We provide algorithms for the individual compute nodes to ensure seamless handoffs that maximally leverage parallelism. PADUA is domain-independent and can be applied to many domains (perhaps with some specialization). We conducted detailed experiments with PADUA on two real-world datasets. First, we tested PADUA on the ITEA CANDELA video surveillance dataset. Second, we tested PADUA on network traffic data appropriate for cybersecurity applications. PADUA scales extremely well with the number of processors and significantly outperforms past work both in accuracy and time. Thus, PADUA represents the first parallel architecture and algorithms for identifying unexplained situations in observation data and—in addition to high accuracy—can scale well.

1. INTRODUCTION

Many organizations continuously monitor transactional data in order to identify irregularities. For instance, security officers at airports need to identify unexplained behavioral patterns (e.g., people who leave unattended packages) in order to identify threats to public safety. Banks monitor transaction streams on their secure web sites to identify suspicious behaviors. Insurance companies look for unexplained patterns in claims data. Stock market regulators look for suspicious trading patterns that may artificially drive stock prices up or down [Palshikar and Apte 2008]. In computer security, attack graphs [Albanese et al. 2011a] have been developed in order to identify known attack patterns by which hackers try to compromise systems. Insurance investigators have also developed patterns of activity to look for [Bordoni et al. 2001] . In all of these applications, experts have identified "known" patterns to look for. These known patterns include both harmless and harmful behavior—much work has focused on learning patterns of behavior [Zhang et al. 2005; Kim and Grauman 2009; Yin et al. 2008; Hu et al. 2009; Zhang et al. 2009; Jiang et al. 2009; Mahajan et al. 2004; Mecocci and Pannozzo 2005; Jiang et al. 2010; Wang et al. 2012] so that statistically significant variations of these known patterns can be flagged.

DOI : http://dx.doi.org/10.1145/0000000.0000000

"Bad guys" are constantly innovating and seeking new ways to carry out their crimes. In this paper, we propose PADUA, the first parallel architecture for the detection of unexplained "situations" that we are aware of. A situation is any subset of a given log of observations. PADUA starts with some set $\mathcal{A}$ of activity models that are *known* in advance. $\mathcal{A}$ may consist of a combination of harmless and harmful activities—when a known harmful activity is detected in a log of observations, PADUA will automatically raise an alert that a security analyst can respond to (or a program specialized to address that harmful activity can be invoked). However, this paper focuses on the problem of identifying situations that collectively cannot be satisfactorily explained by any of the activity models in $\mathcal{A}$. The contributions of this paper are as follows.

(1) Though the goal of PADUA is not to propose a new activity model, in Section 2 we propose *Probabilistic Penalty Graphs* (PPGs for short). PPGs extend stochastic automata [Albanese et al. 2011b] in order to handle "noise". Handling noise is critical in real-world applications as many observed events are probably irrelevant. For instance, in an airport, people exhibit a number of behaviors that were not thought of when models of known activities were developed. Similarly, activity patterns at a bank web site may exhibit a lot of noise. A situation is deemed unexplained if, intuitively, the situation cannot be explained by any known activity model in $\mathcal{A}$ with a score exceeding a user-defined threshold τ.
(2) In Section 3, we propose a data structure that combines a set of PPGs into a single *Super-PPG*, together with algorithms to maintain this Super-PPG data structure and to seamlessly flag unexplained situations when they occur. Super-PPGs offer scalability on single machine implementations.
(3) In Section 4, we show that given a set of $(K+1)$ cluster nodes, we can split a Super-PPG into a set of K sub-PPGs, each of which can be executed on one of K compute nodes. We show 5 different approaches to splitting a Super-PPG for the purpose of detecting unexplained situations.
(4) In Section 5, we provide parallel coordination algorithms for detecting unexplained situations using which each compute node can "hand off" computation to an appropriate other node when necessary.
(5) We implemented all these data structures and algorithms on a parallel architecture with over 160 CPUs and conducted accuracy and timing experiments with two real-world datasets. The ITEA CANDELA dataset[1] contains a wide range of video surveillance data. The Naples Network Traffic dataset contains network traffic from the University of Naples. Our experiments (reported in Section 6) show that: (*i*) PADUA scales extremely well with the number of CPUs and runs faster than past work for detecting unexplained situations, (*ii*) PADUA significantly improves the accuracy of past work [Albanese et al. 2011a]—the F-measure increases from 0.72 to 0.89.

We emphasize that an unexplained situation may not be a harmful one. For instance, if PADUA flags a situation as being unexplained, a security analyst may look at the situation—if it is harmless, it can be added to the set $\mathcal{A}$ of known activities as a harmless activity and if it is harmful, likewise, it can be added to the set $\mathcal{A}$ and flagged as harmful. By flagging unexplained situations we have the potential for a semi-automated method to grow the set of known activity patterns over time.

[1] http://www.multitel.be/image/research-development/research-projects/candela/

1.1. Related Work

A priori definitions. Several researchers have studied how to search for specifically defined patterns of normal/abnormal activities using different models such as Hidden Markov Models [Vaswani et al. 2005; Cuntoor et al. 2008], coupled Hidden Markov Models [Brand et al. 1997; Oliver et al. 2002], Dynamic Bayesian Networks [Hamid et al. 2003], Stochastic Automata [Albanese et al. 2007], Bayesian networks and probabilistic finite state automata [Hongeng et al. 2004]. In contrast, this paper starts with a set $\mathcal{A}$ of known activity models (for normal/abnormal activities) and finds sequences that are not sufficiently explained by any of the known models in $\mathcal{A}$.

Learning and then detecting abnormality. Much work first learns a normal activity model and then detects abnormalities. In data mining, objects (e.g., people, transactions) may have an associated vector of properties. By clustering a set of objects, we can identify objects that either do not belong to any cluster or are "far away" (in the high-dimensional vector space) from any cluster. Here, belonging to a sufficiently big cluster is considered "normal", being far away from a cluster or being part of a tiny cluster is considered anomalous. Examples of such an approach are encompassed in [Xu and II 2005]. [Zhang et al. 2005] proposes a semi-supervised approach to detect abnormal events that are rare, unexpected, and relevant. Detection of unseen or rarely occurring events are also considered in [Kim and Grauman 2009; Yin et al. 2008; Hu et al. 2009; Zhang et al. 2009; Jiang et al. 2009]. [Xiang and Gong 2008] defines an anomaly as an atypical behavior pattern that is not represented by sufficient samples in a training dataset and satisfies an abnormal pattern. [Mahajan et al. 2004] learns patterns of activities over time in an unsupervised way. [Mecocci and Pannozzo 2005] learn trajectory prototypes and detect anomalous behaviors when visual trajectories deviate from the learned representations of typical behaviors. [Jiang et al. 2010] automatically learn high frequency events and declares them normal—events deviating from these rules are anomalies. [Wang et al. 2012] first analyzes and designs features from the data and then detects abnormal activities using the designed features. All these approaches first learn normal activity models and then detect abnormal/unusual events. These papers differ from ours as they consider rare events to be abnormal. In contrast, we may consider situations to be unexplained even if they are not rare—if existing models do not capture them with high probability, they are flagged as unusual. In addition, if a model exists for a rare situation, we would flag it as "explained", while many of these frameworks would not.

Similarity-based abnormality. [Zhong et al. 2004] proposes an unsupervised technique in which each event is compared with all other observed events to determine how many similar events exist. Unusual events are events for which there are no similar events. Similarly, [Au et al. 2006] considers a scene in a video anomalous when the maximum similarity between the scene and all previously viewed scenes is below a threshold. In [Zhou et al. 2007], frequently occurring patterns are normal and patterns that are dissimilar from most patterns are anomalous. An unsupervised approach, where an abnormal trajectory refers to something that has never (or rarely) been seen was proposed in [Brun et al. 2012]. In [Adam et al. 2008], unusual events are detected by computing the likelihood of a new observation w.r.t. the probability distribution of prior observations.

Cybersecurity. Intrusion detection systems (IDSs) monitor network traffic for suspicious behavior and trigger alerts [Mukherjee et al. 1994; García-Teodoro et al. 2009; Jones and Li 2001]. Alert correlation methods aggregate such alerts into multi-step attacks [Wang et al. 2006; Ning et al. 2002; Al-Mamory and Zhang 2009; Albanese et al. 2011a]. Intrusion detection techniques can be broadly classified into *signature-based* [Jones and Li 2001] and *profile-based* (or *anomaly-based*) [García-Teodoro et al.

2009] methods. A signature is a set of conditions that characterize intrusion activities w.r.t. packet headers and payload content. Historically, signature-based methods have been used extensively to detect malicious activities. In profile-based methods, a known deviation from the norm is considered anomalous (e.g. HTTP traffic on a non-standard port). In contrast, in this paper, we consider the case where we have a set $\mathcal{A}$ of known activities (both innocuous and dangerous)—and we are looking for observation sequences that cannot be explained by either (if they were, they would constitute patterns that were known a priori). These need to be flagged as they might represent "zero day" attacks. Correlation techniques try to reconstruct attacks from isolated alerts. The main role of correlation is to provide a higher level view of the actual attacks [Al-Mamory and Zhang 2009; Ning et al. 2002; Qin and Lee 2003; Qin 2005; Oliner et al. 2010]. Both IDSs and correlation techniques rely on models encoding a priori knowledge of either normal or malicious behavior, and cannot appropriately deal with events that are not explained by the underlying models.

2. PROBABILISTIC PENALTY GRAPHS

We assume the existence of a finite set ACT of *action symbols*. A *log tuple* is a $(k+1)$-tuple $l = (s, att_1, \ldots, att_k)$ where $s \in$ ACT, and $att_1, ..., att_k$ are attributes (e.g., source, actor, time-stamp etc.). A *log* is a finite sequence of log tuples. We use $l.action$ to refer to action symbol s of tuple l. Intuitively, a log tuple corresponds to an observation of $l.action$ along with the associated attributes of the observation $att_1, \ldots, att_k$. By convention, if action a_2 occurs after a_1 in a log, then the action a_2 occurred temporally after a_1.

2.1. Definition of PPGs

In this paper, we model activities using *probabilistic penalty graphs* which extend the stochastic activity definition of [Albanese et al. 2011b] with a penalty component which allows us to handle noise.

Definition 2.1 (*Probabilistic Penalty Graph (PPG)*). A *probabilistic penalty graph* (PPG for short) is a labeled graph $A = (V, E, \delta, \rho)$ where:

— $V \subseteq$ *ACT* is the set of nodes;
— $E \subseteq V \times V$ is the set of edges (with no self-loops);
— the set of start nodes (i.e., nodes with in-degree zero), denoted $start(A)$, is non-empty;
— the set of end nodes (i.e., nodes with out-degree zero), denoted $end(A)$, is non-empty;
— $\delta : E \rightarrow (0, 1)$ is a function that associates a probability distribution with the outgoing edges of each node, i.e., $\forall v \in V$, $\sum_{(v,v') \in E} \delta(v, v') = 1$;
— $\rho : E \rightarrow (0, 1)$ is a function that associates a noise degradation value with each edge.

The last component (noise degradation function) is new and extends the stochastic automata of [Albanese et al. 2011b]. To understand the intuition behind it, consider an edge $e = (a_1, a_2)$ in some PPG labeled with probability $\delta(e)$ and noise degradation $\rho(e)$. This edge can be read as: if a_1 occurs in a log and a_2 occurs later and there are z actions $b_1, \ldots, b_z$ in the log *strictly* between a_1 and a_2, then the score associated with the subsequence $\langle a_1, b_1, \ldots, b_z, a_2 \rangle$ is $\delta(e) \cdot \rho(e)^z$. As the degradation $\rho(e) \in [0, 1]$, the larger z is, the lower the subsequence score (because $\rho(e)^z$ decreases as z increases). Thus, the subsequence "pays a penalty" as the amount of noise in it increases. For example, consider an edge $e = (a_1, a_2)$ with $\delta(e) = 0.7$ and $\rho(e) = 0.2$. Suppose our log contains the sequence $\langle a_1, b_1, b_2, a_2 \rangle$. The score of a transition from a_1 to a_2 is $0.7 * (0.2)^2 = 0.028$ because there are two "noisy" events (b_1, b_2) in the middle. In contrast, with the same δ and ρ, consider the log $\langle a_1, b_2, b_2, b_3, a_2 \rangle$ where an extra noisy observation b_3 occurs

between a_1, a_2. The score of a transition from a_1 to a_2 is $0.7 * (0.2)^3 = 0.0056$, an even smaller number.

Example 2.2. The PPG in Fig. 1 shows an e-commerce network intrusion scenario from [Albanese et al. 2011a]. Here PostFirewallAccess is the only start node and CentralDBServerAccess is the only end node. Each edge e is labeled with $(\delta(e), \rho(e))$, i.e., the probability and noise degradation values for the edge. For example, the outgoing edges of node PostFirewallAccess show that there is a 90% probability the next action is MobileAppServerAccess and a 10% probability it is CentralDBServerAccess. Furthermore, the former edge has a degradation value 0.2 and the latter has a degradation value 0.4.

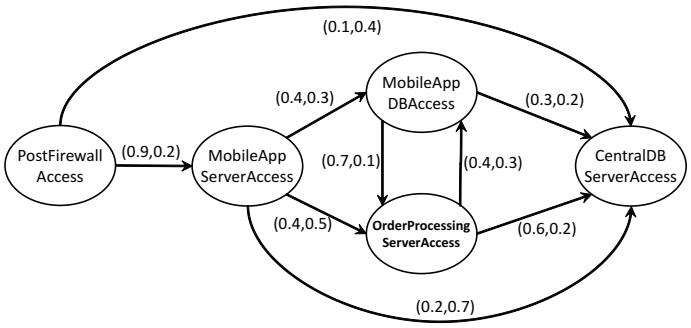

Fig. 1. Probabilistic Penalty Graph.

2.2. Unexplained Situations

In order to define unexplained situations, we first define PPG occurrences in a log.

Definition 2.3 (*PPG Occurrence*). Let $A = (V, E, \delta, \rho)$ be a PPG and L a log. An *occurrence* of A in L is a pair (L^*, I^*) where

(1) $L^* = \langle l_1, \ldots, l_n \rangle$ is a contiguous subsequence of L;
(2) $I^* = \langle i_1, \ldots, i_m \rangle$ is a sequence of indices of L, with $1 \leq i_j \leq n$, $i_1 = 1$, $i_m = n$, and $i_j < i_{j+1}$;
(3) $\forall j \in [1, m-1]$, $(l_{i_j}.action, l_{i_{j+1}}.action) \in E$;
(4) $l_1.action \in start(A)$;
(5) $l_n.action \in end(A)$.

Thus, an occurrence of a PPG A consists of a *contiguous* subsequence L^* of L and a set I^* of indexes specifying the tuples in L^* whose associated actions are a path from a start to an end node in A—the remaining tuples of L^* constitute "noise". Fig. 2 illustrates the definition pictorially—each "box" represents a log tuple. In this example, $L^* = \langle l_1, l_2, \ldots, l_9 \rangle$ consists of the entire sequence shown, while $I^* = \langle 1, 5, 6, 9 \rangle$ refers to the shaded boxes. For this to be a valid occurrence of a PPG A, we need to make sure that A contains an edge from $l_1.action$ to $l_5.action$, from $l_5.action$ to $l_6.action$, and from $l_6.action$ to $l_9.action$, and moreover ensure that $l_1.action$ is a valid start state node and $l_9.action$ is a valid end node for A.

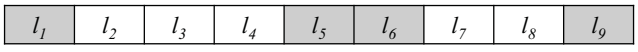

Fig. 2. An example of PPG occurrence.

Of course, some occurrences are "good" while others may not be as good. This "goodness" is captured via the score of an occurrence which is defined below. The *score* of (L^*, I^*) is

$$score(L^*, I^*) = \Pi_{j \in [1,m-1]} \delta(l_{i_j}.action, l_{i_{j+1}}.action) \cdot \rho(l_{i_j}.action, l_{i_{j+1}}.action)^z$$

with $z = i_{j+1} - i_j - 1$. Thus, the score of (L^*, I^*) is computed by taking into account

(1) the probability on the edges belonging to the path $l_{i_1}.action, \ldots, l_{i_m}.action$ specified by I^* (i.e., $\Pi_{j \in [1,m-1]} \delta(l_{i_j}.action, l_{i_{j+1}}.action)$), and
(2) the amount of noise in L^*.

If there are many tuples in L^* which are not part of a path from a start to an end node in A, then the score of (L^*, I^*) decreases. This is because when z (the amount of noise) increases, multiplying $\delta(l_{i_j}.action, l_{i_{j+1}}.action)$ by $\rho(l_{i_j}.action, l_{i_{j+1}}.action)^z$ yields a smaller score, because $\rho(-,-) \in [0,1]$. We illustrate this below.

Example 2.4. Consider a log whose associated sequence of action symbols is ⟨PostFirewallAccess, x, MobileAppServerAccess, OrderProcessingServerAccess, x, x, CentralDBServerAccess, x⟩, where x $\notin V$ and V is the set of vertices of the PPG in Fig. 1. Then, (L^*, I^*), where L^* = ⟨PostFirewallAccess, x, MobileAppServerAccess, OrderProcessingServerAccess, x, x, CentralDBServerAccess⟩ and $I^* = \langle 1, 3, 4, 7 \rangle$, is an occurrence of the PPG in Fig. 1 and its score is $0.9 \cdot 0.2^1 \cdot 0.4 \cdot 0.5^0 \cdot 0.6 \cdot 0.2^2$.

We now come to the critical definition of an unexplained situation.

Definition 2.5 (*Unexplained Situation*). Let $A = (V, E, \delta, \rho)$ be a PPG and L a log. An *unexplained situation* for A is a pair (L_u, I_u) where

(1) Conditions 1–4 in Definition 2.3 hold;
(2) $l_n.action \in V - end(A)$;
(3) there is no occurrence (L^*, I^*) of A such that L_u is a prefix of L^* and I_u is a prefix of I^*;
(4) there is no pair $(L'_u, I'_u) \neq (L_u, I_u)$ such that L_u is a prefix of L'_u, I_u is a prefix of I'_u, and (L'_u, I'_u) satisfies all conditions above.

Thus, an unexplained situation for A consists of a *contiguous* subsequence L_u of L and a set I_u of indexes specifying the tuples in L_u whose associated actions are a path ending in a non-end node in A. The third condition requires that an unexplained situation cannot be extended so as to get an occurrence of A. The fourth condition ensures that unexplained situations are as long as possible.

The *score* of an unexplained situation (L_u, I_u) is given by:

$$score(L_u, I_u) = \Pi_{j \in [1,m-1]} (1 - \delta(l_{i_j}.action, l_{i_{j+1}}.action)) \cdot (1 - \rho(l_{i_j}.action, l_{i_{j+1}}.action))^z$$

with $z = i_{j+1} - i_j - 1$. The score of (L_u, I_u) takes into account the probabilities of the edges along the path specified by I_u and the noise degradation values for the tuples in L_u which are not referenced by I_u; however, edge probabilities and degradation values are complemented. We illustrate this in the following examples.

Example 2.6. Consider the action symbols MobileAppServerAccess, MobileAppDBAccess, and OrderProcessingServerAccess of the PPG in Fig. 1. Let δ(MobileAppServerAccess, MobileAppDBAccess) = 0.1 and δ(MobileAppServerAccess, OrderProcessingServerAccess) = 0.7. Now consider a log containing two subsequences of tuples, L_1 with associated action symbols ⟨MobileAppServerAccess, x, x, x, MobileAppDBAccess⟩, and L_2 with associated action symbols ⟨MobileAppServerAccess, x, x, x, OrderProcessingServerAccess⟩. In this case, L_1

may contribute to the score of an occurrence of the PPG with a probability equal to 0.1, and L_2 with a probability of 0.7. This means that "moving" from MobileAppServerAccess to MobileAppDBAccess is less likely than moving from MobileAppServerAccess to OrderProcessingServerAccess in the activity described by the PPG. Hence, when computing the score of unexplained situations containing L_1 or L_2, the contributions are complemented: moving from MobileAppServerAccess to MobileAppDBAccess is considered "more unexplained" than moving from MobileAppServerAccess to OrderProcessingServerAccess, and their contributions to the score become 0.9 and 0.3, respectively.

Example 2.7. Consider the log $\langle$PostFirewallAccess, x, MobileAppServerAccess, MobileAppDBAccess, x, x$\rangle$. Then, (L_u, I_u), where $L_u = \langle$PostFirewallAccess, x, MobileAppServerAccess, MobileAppDBAccess$\rangle$ and $I_u = \langle 1, 3, 4\rangle$ is an unexplained situation for the PPG in Fig. 1 with score $0.1 \cdot 0.8^1 \cdot 0.6 \cdot 0.7^0$.

We now define the concept of an unexplained situation w.r.t. a PPG when we are given a threshold τ.

Definition 2.8. Let A be a PPG, L a log, and $\tau \in [0, 1]$. A *τ-unexplained situation* for A is an unexplained situation (L_u, I_u) for A with $score(L_u, I_u) \geq \tau$.

The definition of a τ-unexplained situation above is given for a *single* PPG. Given a set $\mathcal{A}$ of PPGs, we would like to find a set of τ-unexplained situations *w.r.t. the whole set* $\mathcal{A}$. We define the τ-unexplained situations for a set of PPGs as follows.

Definition 2.9. Let $\mathcal{A}$ be a set of PPGs, L a log, and $\tau \in [0, 1]$. A *τ-unexplained situation* for $\mathcal{A}$ is a maximal contiguous subsequence L_u of L such that for every A in $\mathcal{A}$, there is a τ-unexplained situation (L'_u, I'_u) for A s.t. L_u is a subsequence of L'_u.

Thus, given a set $\mathcal{A}$ of PPGs, a τ-unexplained situation is a maximal contiguous subsequence L_u of the log which is contained in a τ-unexplained situation of every PPG in $\mathcal{A}$ (i.e., L_u is unexplained w.r.t. all PPGs in $\mathcal{A}$). Before computing the set of τ-unexplained situations, we show that our theory has several elegant properties.

As threshold τ is used to select only those unexplained situations for which we have a confidence above τ, higher values of τ are stricter conditions for a situation to be unexplained. The following proposition states that our framework satisfies the natural property that unexplained situations become wider by decreasing the threshold (moreover, new unexplained situations might be found).

PROPOSITION 2.10. *Consider a log L, a set $\mathcal{A}$ of PPGs, and two thresholds $\tau_1, \tau_2 \in [0, 1]$. Let U_1 (resp. U_2) be the set of τ_1- (resp. τ_2-) unexplained situations for $\mathcal{A}$. If $\tau_1 \geq \tau_2$, then for every L^1_u in U_1 there exists L^2_u in U_2 s.t. L^1_u is a contiguous subsequence of L^2_u.*

Given two PPGs $A_1 = (V_1, E_1, \delta_1, \rho_1)$ and $A_2 = (V_2, E_2, \delta_2, \rho_2)$, we write $A_1 \sqsubseteq A_2$ iff *(i)* $V_1 = V_2$, *(ii)* $E_1 = E_2$, and *(iii)* $\delta_1(e) \leq \delta_2(e)$ and $\rho_1(e) \leq \rho_2(e)$ for every $e \in E_1$ (or, equivalently, $e \in E_2$). Given two sets $\mathcal{A}_1$ and $\mathcal{A}_2$ of PPGs we write $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ iff for every $A_1 \in \mathcal{A}_1$ there exists $A_2 \in \mathcal{A}_2$ s.t. $A_1 \sqsubseteq A_2$.

Intuitively, $A_1 \sqsubseteq A_2$ means that A_1 and A_2 are topologically the same, but A_2 has possibly higher edge probabilities or penalties. Notice that higher edge probabilities/penalties for a PPG lower the confidence we have in τ-unexplained situations and thus, we would expect a smaller portion of the log to be unexplained. Indeed, as stated by the following proposition, this is correctly captured by our theory.

PROPOSITION 2.11. *Consider a log L, a threshold $\tau \in [0, 1]$, and two sets of PPGs $\mathcal{A}_1$, $\mathcal{A}_2$. Let U_1 (resp. U_2) be the set of τ-unexplained situations for $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). If*

$\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$, then for every L_u^1 in U_1 there exists L_u^2 in U_2 s.t. L_u^1 is a contiguous subsequence of L_u^2.

As we wish to find situations that are not sufficiently explained by a set $\mathcal{A}$ of PPGs, another natural property is that a smaller portion of the log becomes unexplained by adding PPGs to $\mathcal{A}$. The following corollary says that this property is satisfied by our theory.

COROLLARY 2.12. *Consider a log L, a threshold $\tau \in [0,1]$, and two sets of PPGs $\mathcal{A}_1$, $\mathcal{A}_2$. Let U_1 (resp. U_2) be the set of τ-unexplained situations for $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). If $\mathcal{A}_2 \subseteq \mathcal{A}_1$, then for every L_u^1 in U_1 there exists L_u^2 in U_2 s.t. L_u^1 is a contiguous subsequence of L_u^2.*

2.3. Deriving Noise Degradation Values from a Training Set

We can easily derive noise degradation values from a training set of data as follows. Suppose we have a PPG A and $e = (a_1, a_2)$ is an edge in this PPG. Suppose we have a log L. Let $occ(L, e)$ denote the set of all contiguous sequences in L that start with a_1 and end with a_2. Suppose these are presented to a user for training purposes and the user marks some of these as valid transitions from a_1 to a_2 and marks the others as invalid. Let $valid(L, e)$ be the subset of $occ(L, e)$ marked valid and $invalid(L, e) = occ(L, e) \backslash valid(L, e)$. For any sequence $s = a_1, b_1, \ldots, b_k, a_2$, let $noise(s, e) = k$. Moreover, for each integer i, let $f(i)$ be the percentage of sequences in $occ(L, e)$ with i units of noise between a_1 and a_2 that are marked valid, i.e., $f(i) = \frac{|\{s \mid s \in valid(L,e) \wedge noise(s,e)=i\}|}{|\{s \mid s \in occ(L,e) \wedge noise(s,e)=i\}|}$. We now plot a graph with i on the x-axis and $f(i)$ on the y-axis and look for a value $\rho(e)$ such that the function $g(i) = \delta(e) * \rho(e)^i$ best approximates the function f, i.e. such that the mean square error $\Sigma_i (g(i) - f(i))^2$ is minimized. This can be done by a standard curve fitting procedure [Lancaster and Salkauskas 1986]. [2]

As an example of this procedure, suppose we consider an edge $e = (a_1, a_2)$ and that the training set has a maximum of 3 noisy observations between a_1, a_2. Suppose the table below shows the set $occ(L, e)$ along with the valid/invalid annotation.

Sequence s	$noise(s)$	*Annotation*
a_1, b_1, a_2	1	valid
a_1, b_3, a_2	1	valid
a_1, b_1, b_2, a_2	2	valid
a_1, b_1, b_3, a_2	2	valid
a_1, b_1, b_4, a_2	2	invalid
a_1, b_1, b_2, b_3, a_2	3	valid
a_1, b_2, b_3, b_5, a_2	3	invalid
a_1, b_2, b_6, b_2, a_2	3	invalid

According to this training set, $f(1) = 1, f(2) = 0.67, f(3) = 0.33$. Suppose the transition probability $\delta(e)$ is 0.5. Then we are looking for a function $g(i) = \delta(e) * \rho(e)^i$ such that $\Sigma_{i=1}^3 (g(i) - f(i))^2$ is minimized. Let $\rho(e) = u$. Then we want to minimize $(0.5 * u - 1)^2 + (0.5 * u^2 - 0.67)^2 + (0.5 * u^3 - 0.33)^2$. We can minimize this expression, subject to the requirement that it is non-zero. This is a straightforward polynomial (cubic) constraint solving problem that can be solved using any non-linear constraint solver.

[2]Note that this is just one simple way of learning penalties. Our goal here is not to study machine learning algorithms to learn such graphs, just to show that reasonably simple ways to learn these penalties exist.

3. THE PPG-INDEX: FAST COMPUTATION OF UNEXPLAINED SITUATIONS ON A SINGLE CPU

In this section, we define a PPG-Index to quickly compute the set of all τ-unexplained situations within a log. We first show how to merge all PPGs together into a Super-PPG. We then develop a PPG-Index structure that is automatically updated when new observations come into the log. The PPG-Index is fully geared towards finding the τ-unexplained situations within a log as the log is changing. This section focuses on implementing these operations on a single CPU while subsequent sections define the parallel algorithms within PADUA.

3.1. Super-PPGs

We first define a *Super-PPG*, which is a compact representation of a set of PPGs. Note that a Super-PPG is not a PPG, but a slightly different structure which encapsulates all the information within the given set of PPGs.

Definition 3.1 (*Super-PPG*). Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a set of PPGs, where $\forall i \in [1,k]$, $A_i = (V_i, E_i, \delta_i, \rho_i)$. A *Super-PPG* is a 4-tuple $G(\mathcal{A}) = (V_G, E_G, \delta_G, \rho_G)$ where

— $V_G = \cup_{i\in[1,k]} V_i$ and $E_G = \cup_{i\in[1,k]} E_i$;
— $\delta_G : V_G \times V_G \times \mathcal{A} \rightarrow [0,1]$ is the function s.t. $\delta_G(v, v', A_i) = \delta_i(v, v')$ if $(v, v') \in E_i$, 0 otherwise.
— $\rho_G : V_G \times V_G \times \mathcal{A} \rightarrow [0,1]$ is the function s.t. $\rho_G(v, v', A_i) = \rho_i(v, v')$ if $(v, v') \in E_i$, 0 otherwise.

Basically, the Super-PPG associated with $\mathcal{A}$ has the same vertices as in the graphs in $\mathcal{A}$. The "global" probability function δ_G returns the probability of an edge in this graph w.r.t. a specific activity and the "global" ρ_G noise degradation function does the same.

Example 3.2. Let $\mathcal{A} = \{A_1, A_2\}$ where A_1 is the PPG in Fig. 1 and A_2 is the PPG in Fig. 3(left). Fig. 3(right) shows $G(\mathcal{A})$, where each edge (v_1, v_2) has labels of the form $A : (\delta_G(v_1, v_2, A), \rho_G(v_1, v_2, A))$.

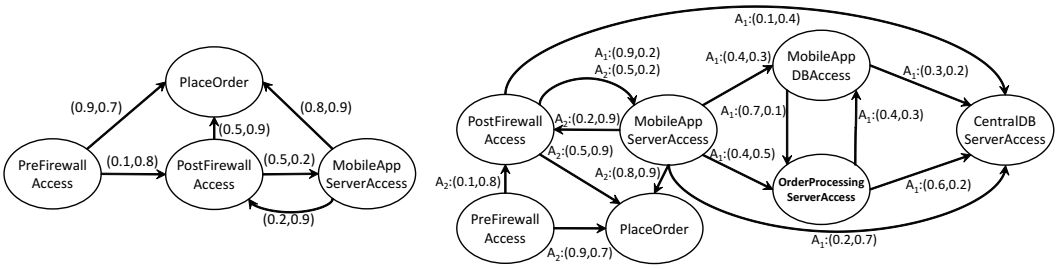

Fig. 3. PPG A_2 (left); Super-PPG (right).

The definition of a Super-PPG yields an immediate way of quickly constructing a Super-PPG.

3.2. The PPG-Index

Our *PPG-Index* uses the Super-PPG to efficiently keep track of all unexplained situations found in a log whose score is above a threshold τ. In the following, we denote the set of references (pointers) to the elements in a set S as *ref*(S).

Definition 3.3 (*PPG-Index*). Let $\mathcal{A}$ be a set of PPGs, $G(\mathcal{A}) = (V_G, E_G, \delta_G, \rho_G)$, and L a log. A *PPG-Index* is a 4-tuple $I_G = (G(\mathcal{A}), tables_G, count_G, completed_G)$, where:

— For each $v \in V_G$, $tables_G(v)$ is a set of tuples of the form $(current, A, score, previous, closed, count)$, where $current \in \mathbf{ref}(L)$, $A \in \mathcal{A}$, $score \in [0,1]$, $previous \in \mathbf{ref}(\mathcal{P})$ with $\mathcal{P} = \bigcup_{v \in V_G} tables_G(v)$, $closed$ is a boolean value, and $count \in \mathbb{N}$;
— $count_G \in \mathbb{N}$;
— $completed_G : \mathcal{A} \to 2^{\mathbf{ref}(\mathcal{P})}$ is a function that associates each PPG with a set of references to the tuples in $tables_G$.

For each action symbol $v \in V_G$, the index contains a table $tables_G(v)$. In the table, each tuple $t = (current, A, score, previous, closed, count)$ represents the fact that an unexplained situation for PPG A contains the log tuple pointed by $current$, and $current.action = v$. In particular:

— the score of the sequence up to the $current$ tuple is equal to the value of $score$;
— $previous$ points to the index tuple that precedes t in the sequence;
— $closed$ indicates if the situation cannot be extended with a score above the threshold;
— $count$ is the number of log tuples indexed before $current$ (including $current$ itself);
— $count_G$ is the global counter of indexed log tuples;
— $completed_G$ associates a PPG with those tuples in $tables_G$ that represent the last action of an unexplained situation.

When the log is empty, the PPG-Index is empty (with empty $tables_G$ and $completed_G$, and with $count_G = 0$)—we use I_G^0 to denote this "empty" PPG-Index.

Example 3.4. Let $\mathcal{A} = \{A_1, A_2\}$ be the set of PPGs of Example 3.2 and consider a log whose associated sequence of action symbols is $\langle$PreFirewallAccess, x, PostFirewallAccess$\rangle$. The corresponding index tables are shown in Fig. 4 (dashed box). The index contains an index tuple in $tables_G$(PreFirewallAccess) (denoted t_{eb} in the following) and two tuples in $tables_G$(PostFirewallAccess) (t_{ic}, t'_{ic} in the following). The presence of t_{eb} indicates that PreFirewallAccess is a start node for A_2: it has no $previous$ index tuple and its $score$ is 1. Moreover, its $count$ is 1 because it is the first action in the log. Likewise, t_{ic} means that PostFirewallAccess is a start node for A_1. Finally, t'_{ic} means that PostFirewallAccess can also follow PreFirewallAccess in an unexplained situation for A_2. Thus, its $previous$ index tuple is t_{eb} and its $score$ is $t_{eb}.score \cdot (1 - \delta_G(\text{PreFirewallAccess}, \text{PostFirewallAccess}, A_2)) \cdot (1 - \rho_G(\text{PreFirewallAccess}, \text{PostFirewallAccess}, A_2))$, where the penalty component derives from the presence of x between PreFirewallAccess and PostFirewallAccess. Note that at this point we have $count_G = 3$, $completed_G(A_1) = \{t_{ic}\}$, and $completed_G(A_2) = \{t'_{ic}\}$.

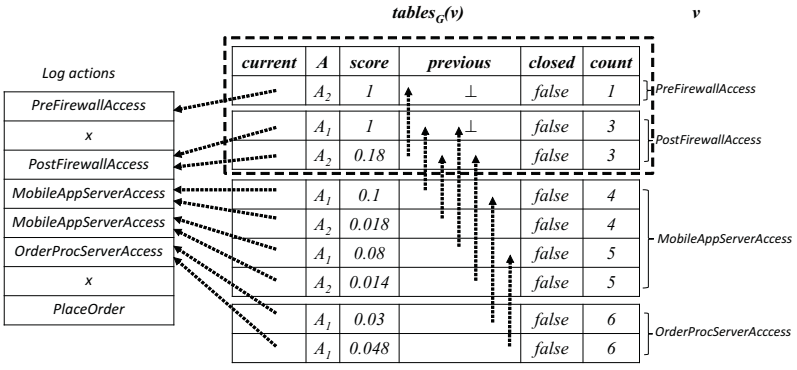

Fig. 4. Sequence of actions and status of $tables_G$.

Fig. 5(left) shows the pseudo-code of the PPG_Insert algorithm that indexes a new log tuple l_{new} with associated action symbol $l_{new}.action = a$. In the algorithm, Lines 3–6 deal with the case where a is a start node for some PPG, by creating a new sequence. Lines 11–12 compute the score associated with the extension of an existing sequence by action a. Finally, Lines 13–17 update $tables_G(a)$ based on information provided by the tuples in each $tables_G(v)$ such that v is an in-neighbor of a in any of the given PPGs.

```
   Algorithm PPG_Insert(l_new, I_G, τ)
   Input: New tuple to be inserted l_new, PPG-Index I_G,
          threshold score τ
   Output: Updated PPG-Index I_G
1  a ← l_new.action
2  count_G ← count_G + 1
   // —— Check whether a is a start node for some PPG ——
3  for each A ∈ 𝒜 s.t. a ∈ start(A)
4    t' ← (l_new, A, 1, ⊥, false, count_G)
5    add t' to tables_G(a) and to completed_G(A)
6  end for
   // —— Append a to an existing sequence, if possible ——
7  for each action symbol v ∈ V_G
8          s.t. ∃A ∈ 𝒜 : δ_G(v, a, A) ≠ 0
9    for each tuple t ∈ tables_G(v)
10           s.t. ¬t.closed and t.A = A
11     z ← count_G − t.count − 1
12     p ← t.score · (1 − δ_G(v, a, A)) · (1 − ρ_G(v, a, A))^z
13     if p ≥ τ and a ∉ end(A)
14           t' ← (l_new, A, p, t, false, count_G)
15           add t' to tables_G(a) and to completed_G(A)
16           remove t from completed_G(A), if present
17     end if
18     if a ∈ end(A)
19       remove t from completed_G(A), if present
20   end for
21 end for
```

```
   Algorithm PPG_Retrieve(I_G)
   Input: PPG-Index I_G
          built using threshold τ
   Output: the set of all τ-unexplained
          situations for 𝒜 = {A_1, ..., A_k}
1  for each A_i ∈ 𝒜
2    U_i ← ∅
3    for each t ∈ completed_G(A_i)
4      l_e ← t.current
5      while t.previous ≠ ⊥
6        t ← t.previous
7      end while
8      l_s ← t.current
9      add (subLog(L, l_s, l_e)) to U_i
10   end for
11 end for
12 return maxComSubseq(U_1, ..., U_k)
```

Fig. 5. PPG_Insert and PPG_Retrieve algorithms.

The PPG_Insert algorithm works with a pruning algorithm PPG_Prune that updates the value of the $closed$ attribute. For each (v, t) such that $v \in V_G$ and $t \in tables_G(v)$, PPG_Prune sets $t.closed$ to $true$ iff $t.score \cdot maxP < \tau$ where $maxP = max_{x|(v,x)\in E}[(1 - \delta_G(v, x, t.A)) \cdot (1 - \rho_G(v, x, t.A))^z]$, E is the set of edges of $t.A$, and $z = count_G - t.count - 1$.

Example 3.5. Consider a log whose associated sequence of action symbols is ⟨PreFirewallAccess, x, PostFirewallAccess, MobileAppServerAccess, MobileAppServerAccess, OrderProcessingServerAccess, x, PlaceOrder⟩. The status of the PPG-Index after indexing this log is shown in Fig. 4. The full details of the process are reported in the Appendix.

The PPG_Retrieve algorithm, shown in Fig. 5(right), uses the Super-PPG to return the set of τ-unexplained situations for $\mathcal{A}$. For each $A \in \mathcal{A}$, PPG_Retrieve computes the set of τ-unexplained situations for A by (*i*) retrieving the index tuples pointed by $completed_G(A)$ and then (*ii*) following $previous$ pointers until $previous = \perp$ for each retrieved index tuple, while storing the needed log tuples ($current$ attribute). Finally, it returns the maximal common contiguous subsequences of the computed sets. The following result establishes correctness and completeness of our algorithms.

PROPOSITION 3.6. *Consider a set $\mathcal{A}$ of PPGs, a log $L = \langle l_1, \ldots, l_n \rangle$, and a threshold $\tau \in [0, 1]$. Let I_G^i be the PPG-Index returned by PPG_Insert(l_i, I_G^{i-1}, τ). Then:*

$$I_G^i = \textit{PPG\_Insert}(l_i, \textit{PPG\_Prune}(I_G^{i-1}, \tau), \tau).$$

Moreover, PPG_Retrieve(I_G^n) is the set of all τ-unexplained situations for $\mathcal{A}$.

4. PARTITIONING SUPER-PPGS ACROSS A COMPUTE CLUSTER

We implement the PPG-Index on a cluster of $(K + 1)$ nodes in two steps. We propose 5 ways of partitioning the Super-PPG into K parts in a way that tries to minimize the expected inter-node communication. Each compute node is allocated one of these parts—the one remaining node is a master node. Within a compute node, a PPG-Index for the portion of the Super-PPG allocated to it is constructed. Each compute node also has a handoff protocol that governs inter-node communications—this will be discussed in the next section.

Let $\mathcal{A}$ be a set of PPGs and $G(\mathcal{A}) = (V_G, E_G, \delta_G, \rho_G)$ the corresponding Super-PPG. A vertex partition of $G(\mathcal{A})$ is a set of graphs $\mathcal{G} = \{G_1 = (V_1, E_1), \ldots, G_K = (V_K, E_K)\}$ such that $\{V_1, \ldots, V_K\}$ is a partition of V_G and $E_i = \{(v, v') \in E_G | \ v, v' \in V_i\}$.

Given an edge $e = (v_i, v_j)$, an *edge cost function* $cost(e)$ can be defined in different ways. We introduce different options in the rest of this section. These cost functions employ probability and noise degradation functions that consider the average of the probability and degradation values appearing on e across all the PPGs, that is, if $e = (v_i, v_j)$, then we define: $\delta'(e) = avg\{\delta_G(v_i, v_j, A) \mid A \in \mathcal{A}, \delta_G(v_i, v_j, A) > 0\}$ and $\rho'(e) = avg\{\rho_G(v_i, v_j, A) \mid A \in \mathcal{A}, \delta_G(v_i, v_j, A) > 0\}$. The *cost* of $\mathcal{G}$ is $cost(\mathcal{G}) = \sum_{1 \le i,j \le K \wedge i \ne j} cost(G_i, G_j)$, where $cost(G_i, G_j) = \sum_{e=(v_i,v_j) \in E_G, v_i \in V_i, v_j \in V_j} cost(e)$. We will try to find a $\mathcal{G}$ that minimizes $cost(\mathcal{G})$. This can be computed through a standard minimum cut algorithm such as [Karger and Stein 1996].

4.1. Probability Partitioning (PP) and Probability-Penalty Partitioning (PPP)

Our first two cost functions set the cost of an edge in terms of probability alone and in terms of both probability and penalty: $cost_{PP}(e) = 1 - \delta'(e)$ and $cost_{PPP}(e) = (1 - \delta'(e)) \cdot (1 - \rho'(e))$. Intuitively, $cost_{PP}$ tries to keep edges with a high transition probability on the same compute node. Likewise, $cost_{PPP}$ tries to keep edges with both a high transition probability and a high noise degradation value together on the same compute node. This is because the higher these values, the higher the probability that the two actions will be connected in an unexplained situation with a score above the threshold.

Example 4.1. Consider the Super-PPG of Example 3.2 (cf. Fig. 3(right)). Suppose we have two compute nodes. A possible vertex partition is the one consisting of two graphs: G_1 containing vertices PostFirewallAccess, MobileAppServerAccess, PreFirewallAccess, and PlaceOrder, and G_2 cointaining vertices MobileAppDBAccess, OrderProcessingServerAccess, and CentralDBServerAccess. The edges of the Super-PPG that go from one of the two graphs to the other are e_1 = (PostFirewallAccess, CentralDBServerAccess), e_2 = (MobileAppServerAccess, MobileAppDBAccess), e_3 = (MobileAppServerAccess, OrderProcessingServerAccess), and e_4 = (MobileAppServerAccess, CentralDBServerAccess). Moreover, we have $\delta'(e_1) = 0.1$, $\rho'(e_1) = 0.4$, $\delta'(e_2) = 0.4$, $\rho'(e_2) = 0.3$, $\delta'(e_3) = 0.4$, $\rho'(e_3) = 0.5$, $\delta'(e_4) = 0.2$, and $\rho'(e_4) = 0.7$. The following costs are obtained by considering the probability partitioning: $cost_{PP}(e_1) = 1 - 0.1 = 0.9$, $cost_{PP}(e_2) = 1 - 0.4 = 0.6$, $cost_{PP}(e_3) = 1 - 0.4 = 0.6$, and $cost_{PP}(e_4) = 1 - 0.2 = 0.8$. Thus, the overall cost of the partition is $0.9+0.6+0.6+0.8 = 2.9$. On the other hand, the costs obtained via the probability-penalty partitioning are: $cost_{PPP}(e_1) = (1-0.1) \cdot (1-0.4) = 0.54$, $cost_{PPP}(e_2) = (1-0.4) \cdot (1-0.3) = 0.42$, $cost_{PPP}(e_3) = (1-0.4) \cdot (1-0.5) = 0.3$, and $cost_{PPP}(e_4) = (1-0.2) \cdot (1-0.7) = 0.24$. In this case, the overall cost of the partition is $0.54 + 0.42 + 0.3 + 0.24 = 1.5$.

4.2. Expected Penalty Partitioning (EPP)

Suppose $occ(v_i, v_j)$ is the expected number of log tuples appearing between two tuples with actions v_i and v_j in the log. The function occ can be easily learned from a historical

log. We can then define $cost_{EPP}(e) = (1 - \rho'(e))^{occ(v_i,v_j)}$. Thus, $cost_{EPP}$ assigns to an edge, a cost that takes into account the expected penalty value between the two actions involved in the edge. Again, the higher this value, the higher the probability that the two actions will be connected in an unexplained situation with a high score.

Example 4.2. Consider the vertex partition of Example 4.1 and assume function occ is defined as follows: occ(PostFirewallAccess, CentralDBServerAccess) $= 3$, occ(MobileAppServerAccess, MobileAppDBAccess) $= 4$, occ(MobileAppServerAccess, OrderProcessingServerAccess) $= 5.7$, and occ(MobileAppServerAccess, CentralDBServerAccess) $= 2$. The following costs are obtained by considering the expected penalty partitioning: $cost_{EPP}(e_1) = (1 - 0.4)^3 = 0.216$, $cost_{EPP}(e_2) = (1 - 0.3)^4 = 0.2401$, $cost_{EPP}(e_3) = (1 - 0.5)^{5.7} = 0.019$, and $cost_{EPP}(e_4) = (1 - 0.7)^2 = 0.09$. Thus, the overall cost of the partition is $0.216 + 0.2401 + 0.019 + 0.09 = 0.5651$.

4.3. Temporally Discounted Expected Penalty Partitioning (tEPP)

Given a log $L = \langle l_1, \ldots, l_{|L|} \rangle$, we consider temporal windows of length t and discount for old occurrences. We define $occ_t(v_i, v_j, w)$ as the expected number of log tuples appearing between two tuples with actions v_i and v_j *in the w-th temporal window in the log*. Temporal windows are ordered starting from the end of the log, i.e., tuple t_k is in the w-th temporal window if $|L| - w \cdot t + 1 \leq k \leq |L| - (w-1) \cdot t$. Then we consider a discount factor $d \in [0, 1]$ and define $cost_{tEPP}(e) = \sum_{w>0, occ_t(v_i,v_j,w)>0} (1 - \rho'(e))^{\frac{1}{d^w} occ_t(v_i,v_j,w)}$.

Example 4.3. Consider the vertex partition of Example 4.1. Suppose we have a log with 800 tuples and we want to consider temporal windows of length 500. This means that there are two temporal windows to be considered: one goes from tuple 301 to tuple 800, while the other goes from tuple 1 to tuple 300. Suppose the discount factor $d = 2$ and occ is defined as follows: occ(PostFirewallAccess, CentralDBServerAccess, 1) $= 3$, occ(PostFirewallAccess, CentralDBServerAccess, 2) $= 2$, occ(MobileAppServerAccess, MobileAppDBAccess, 1) $= 4$, occ(MobileAppServerAccess, MobileAppDBAccess, 2) $= 1$, occ(MobileAppServerAccess, OrderProcessingServerAccess, 1) $= 5.7$, occ(MobileAppServerAccess, OrderProcessingServerAccess, 2) $= 4$, occ(MobileAppServerAccess, CentralDBServerAccess, 1) $= 2$, and occ(MobileAppServerAccess, CentralDBServerAccess, 2) $= 3.3$. The following costs are obtained by considering the temporally discounted expected penalty partitioning: $cost_{tEPP}(e_1) = (1-0.4)^{0.5 \cdot 3} + (1-0.4)^{0.25 \cdot 2} = 1.239$, $cost_{tEPP}(e_2) = (1-0.3)^{0.5 \cdot 4} + (1-0.3)^{0.25 \cdot 1} = 1.405$, $cost_{tEPP}(e_3) = (1 - 0.5)^{0.5 \cdot 5.7} + (1 - 0.5)^{0.25 \cdot 4} = 0.639$, and $cost_{tEPP}(e_4) = (1 - 0.7)^{0.5 \cdot 2} + (1 - 0.7)^{0.25 \cdot 3.3} = 0.670$. Thus, the overall cost of the partition is 3.953.

4.4. Occurrence Partitioning (OP)

In this case, we consider a pruning threshold c and define $cost_{OP}(e) = \frac{1}{occ(v_i,v_j)}$ if $\frac{1}{occ(v_i,v_j)} \geq c$, and 0 otherwise. Thus, $cost_{OP}$ is inversely proportional the expected number of log tuples appearing between two tuples with actions v_i and v_j. However, if this ratio is too small (i.e., below threshold c), we assume there is no cost to placing v_i, v_j on different nodes.

Example 4.4. Consider the partition of Example 4.1 and suppose occ is defined as in Example 4.2. Let $c = 0.3$. Then: $cost_{OP}(e_1) = 0.33$, $cost_{OP}(e_2) = 0$, $cost_{OP}(e_3) = 0$, and $cost_{OP}(e_4) = 0.5$. Thus, the overall cost of the partition is 0.83 .

5. PARALLEL DETECTION

In this section we show how the PPG-Index and the PPG_Retrieve algorithm can be adapted to a cluster of K compute nodes and a master node — the required modifications involve coordination through inter-node communication.

After partitioning a Super-PPG $G(\mathcal{A}) = (V_G, E_G, \delta_G, \rho_G)$ into K components, each component is assigned to a different compute node—thus, each vertex $v \in V_G$ is managed by a single compute node denoted $node(v)$. All compute nodes store the $tables_G$ structures for the vertices they manage, and their own $completed_G$ structures. However, each tuple t in $tables_G$ has a seventh component, called $starting$, i.e. the ID of the log tuple from which the (partial) unexplained situation represented by t started. All nodes in the cluster also store a copy of $G(\mathcal{A})$ and the value of the threshold τ.

The master node, in order to index a new log tuple l_{new}, sends it to the compute node that manages $l_{new}.action$ after updating the value of $count_G$ (pPPG_Send algorithm, reported in Fig. 6). At any time, the master node can execute the pPPG_Retrieve algorithm (Fig. 6) to build the set of τ-unexplained situations detected so far. Note that the only difference between the pPPG_Retrieve algorithm and its single-CPU counterpart PPG_Retrieve is that, for each PPG A_i, instead of retrieving tuples from the local $completed_G(A_i)$, the pPPG_Retrieve algorithm queries compute nodes—then, instead of following backward pointers through the PPG-Index, it can directly add the corresponding sub-logs to the set U_i by using the $starting$ component of the index tuples.

```
    Algorithm pPPG_Send(l_new)
    Input: New log tuple l_new
1   count_G ← count_G + 1
2   node(l_new.action).pPPG_Insert(l_new, count_G)
```

```
    Algorithm pPPG_Retrieve()
    Output: τ-unexplained situations for A = {A_1, ..., A_k}
1   for each A_i ∈ A
2     U_i ← ∅
3     for each compute node node managing a vertex in A_i
4       get completed_G(A_i) from node
5       for each t ∈ completed_G(A_i)
6         add (subLog(L, t.starting, t.current)) to U_i
7       end for
8     end for
9   end for
10  return maxCommSubseq(U_1, ..., U_k)
```

Fig. 6. Algorithms executed by the master node.

The compute nodes execute the pPPG_Insert algorithm (reported in Fig. 7) when requested by the master node, and communicate with one another through the pPPG_Get algorithm (Fig. 7). In particular, Lines 19–22 of the pPPG_Insert algorithm retrieve from another node a set T of index tuples that represent unexplained situations that can be extended, and update the local $tables_G$ and $completed_G$. The pPPG_Get algorithm corresponds to Lines 8–16 of PPG_Retrieve — in this case, the set T is obviously returned to the requesting node. All compute nodes run the PPG_Prune algorithm given in Section 3 as well, independently and concurrently with pPPG_Insert and pPPG_Get.

6. EXPERIMENTAL RESULTS

The Super-PPG management/partitioning and detection functionalities were developed in C++ while the parallel detection algorithms were implemented using the Message Passing Interface parallel programming model. We used METIS[3] libraries for partitioning. Super-PPGs with several hundreds of vertices were always partitioned in a

[3]http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

	Algorithm pPPG_Insert($l_{new}, count_G$) **Input:** New tuple to be inserted l_{new}, tuple count $count_G$
1	$a \leftarrow l_{new}.action$
	// —— Check whether a is a start node ——
2	**for each** $A \in \mathcal{A}$ s.t. $a \in start(A)$
3	$t' \leftarrow (l_{new}, A, 1, \perp, false, count_G, l_{new})$
4	add t' to $tables_G(a)$ and to $completed_G(A)$
5	**end for**
	// —— Append a to an existing sequence, if possible ——
6	**for each** action symbol $v \in V_G$ s.t. $\exists A \in \mathcal{A} : \delta_G(v, a, A) \neq 0$
7	**if** $node(v)$ is the local node
8–17	// —— Same as PPG_Insert (lines 8–17) ——
18	**else** // —— In this case, coordinate with another cluster node ——
19	**for each** tuple $t \in node(v)$.pPPG_Get($v, a, A, count_G$)
20	$t' \leftarrow (l_{new}, A, p, t, false, count_G, t.starting)$
21	add t' to $tables_G(a)$ and to $completed_G(A)$
22	**end for**
23	**end if**
24	**end for**

	Algorithm pPPG_Get($v, a, A, count_G$) **Input:** Action symbols v, a, PPG A, tuple count $count_G$ **Output:** Set T of index tuples
1	$T \leftarrow \emptyset$
2	**for each** tuple $t \in tables_G(v)$ s.t. $\neg t.closed$ and $t.A = A$
3	$z \leftarrow count_G - t.count - 1$
4	$p \leftarrow t.score \cdot (1 - \delta_G(v, a, A)) \cdot (1 - \rho_G(v, a, A))^z$
5	**if** $p \geq \tau$ and $a \notin end(A)$
6	add t to T
7	remove t from $completed_G(A)$, if present
8	**if** $a \in end(A)$ **then** remove t from $completed_G(A)$, if present
9	**end for**
10	**return** T

Fig. 7. Algorithms executed by the compute nodes.

few milliseconds. We tested PADUA's accuracy and processing time using the *SCOPE*[4] distributed computing infrastructure at the University of Naples. SCOPE consists of over 300 compute nodes (quad-core Intel Xeon 2.33GHz processors with 8GB RAM) communicating by dedicated fiber channels. Tests were done with both video surveillance and network traffic data.

6.1. Video Surveillance Domain

PADUA was tested on the video surveillance experimental setup and measures used in [Albanese et al. 2011b]. The log was generated using the ITEA CANDELA video dataset[5] which contains several staged package exchanges and object drop-offs and pick-ups. 64 different action symbols were defined in a semi-automatic way using both image processing libraries and human annotation.

Result Quality. Precision and recall were evaluated against ground truth provided in [Albanese et al. 2011b] by human annotators who were given a set of known activity descriptions along with graphical representation of the PPGs (we used 30 PPGs), and then asked to watch the video and identify video segments which where unexplained. Precision P and recall R were defined as $P = \frac{|\{L_u^a \text{ in } U^a | \exists L_u^h \text{ in } U^h \text{ s.t. } L_u^a \simeq L_u^h\}|}{|U^a|}$ and $R = \frac{|\{L_u^h \text{ in } U^h | \exists L_u^a \text{ in } U^a \text{ s.t. } L_u^a \simeq L_u^h\}|}{|U^h|}$ where U^a is the set of unexplained situations returned by our algorithm, U^h is the set of sequences identified as unexplained by human annotators, and $L_u^a \simeq L_u^h$ iff L_u^a and L_u^h overlap by a percentage not smaller than 75%. Fig. 8 shows the results obtained.

[4] www.scope.unina.it

[5] http://www.multitel.be/image/research-development/research-projects/candela/

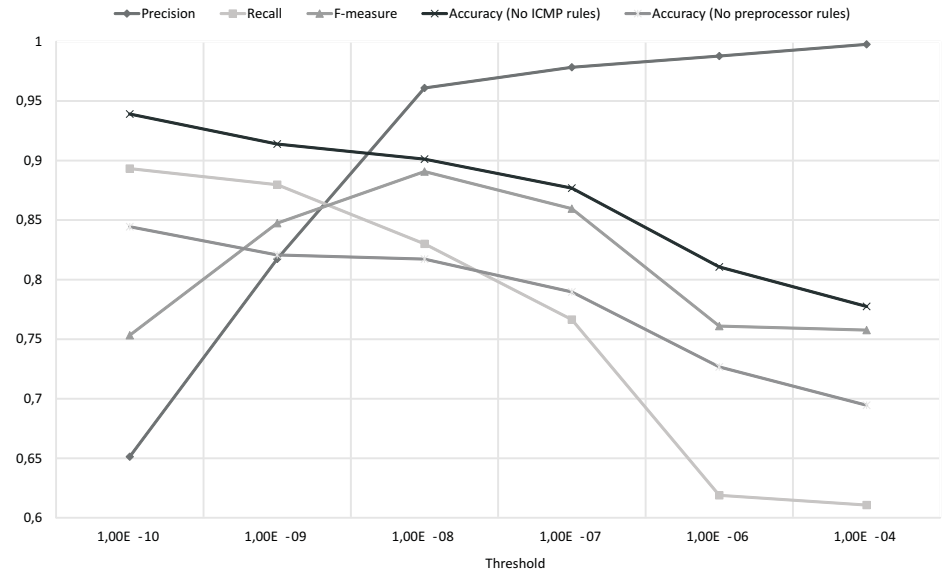

Fig. 8. Precision, recall, and F-measure for the video surveillance dataset, and accuracy for the cybersecurity dataset.

The τ-value that yielded the highest *F-measure*[6], as well as the highest recall, was 10^{-8}. PADUA's precision, recall, and F-measure, respectively, were 0.96, 0.83, and 0.89. In contrast, the corresponding values obtained with the best possible parameter settings in [Albanese et al. 2011b] were $0.73, 0.72,$ and 0.72—significantly lower in all respects than PADUA. Moreover, the experiments confirm the claim in Proposition 2.10: with higher values of τ, the average length of unexplained situations decreases and, as a consequence, we obtain better precision and worse recall.

Processing Times. To evaluate PADUA's scalability when using each of our 5 partitioning schemes, we fixed τ to the value that maximized the F-measure and measured how processing times varied as we varied the length of the video and the number of compute nodes. Figs. 9 and 10(dark line) show the results obtained.

The results show that PADUA provides extremely good performance and scalability. It is able to process up to 294K tuples per second on the longest video sequence (each second of video corresponds to 25 log tuples). Moreover, a 16x increase in the log size only results in a 10x increase in processing time. Though the different partitioning schemes show similar performance for long video sequences, OP, PP, and PPP show a performance advantage for video lengths of up to 125 minutes.

Finally, PADUA clearly outperforms the approach in [Albanese et al. 2011b] — for instance, their processing times for 500 minutes of video were around 10^5 ms. This essentially due to (*i*) the fact that [Albanese et al. 2011b] considers possible worlds and there are exponentially many of them, while we do not, (*ii*) we use a specifically designed index, and (*iii*) the efficiency of our parallel algorithms.

[6]F-measure is given by $\frac{2PR}{P+R}$.

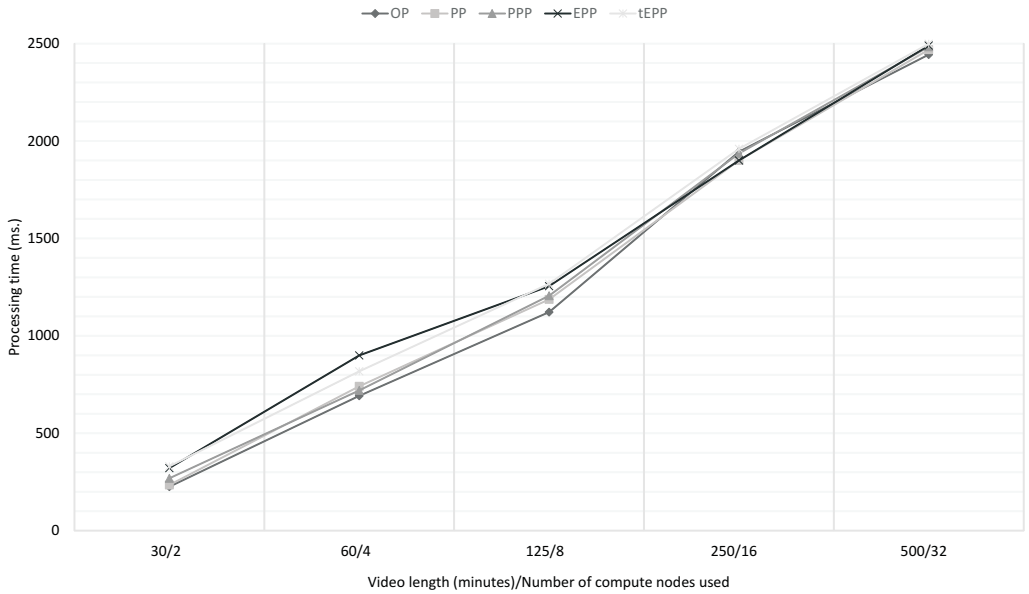

Fig. 9. Processing times for the video surveillance dataset.

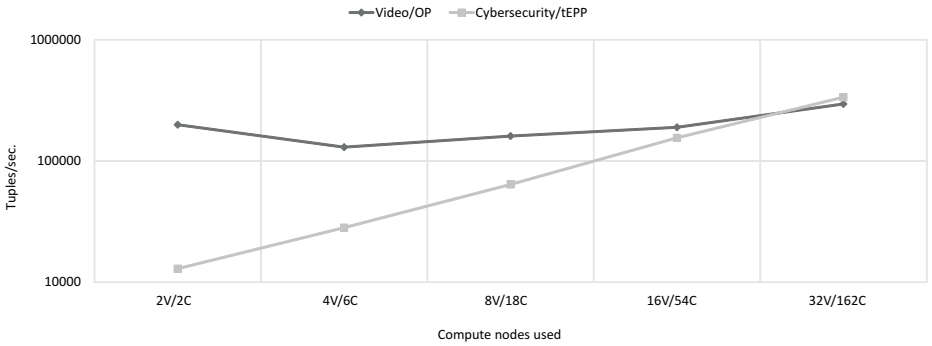

Fig. 10. Tuples processed per second for two dataset/cost function combinations. Label xV/yC on the x-axis stands for "x nodes for the video surveillance dataset, y nodes for the cybersecurity dataset". The video/traffic lengths considered are those corresponding to x nodes in Fig. 9 and y nodes in Fig. 11.

6.2. Cybersecurity Domain

The Naples Network Traffic dataset is built based on a *Network Sniffer*, a *Network Intrusion Detection System* (IDS), and an *Alert Aggregation* component. The Sniffer (implemented by Wireshark[7]) captures network traffic, whereas the IDS (implemented via *SNORT*[8]) analyzes such traffic and generates a sequence of action symbols. As the IDS may return lots of alerts, the Alert Aggregation module aggregates multiple alerts triggered by the same action into a macro-alert based on a set of aggregation rules. The dataset contained 2 full days of traffic (about 1,215,000 log tuples). We defined 350 PPGs, corresponding to the available SNORT rules, containing 722 action symbols.

[7]http://www.wireshark.org/
[8]http://www.snort.org/

The set of SNORT rules comprised ICMP rules, designed to analyze ICMP packets, and preprocessing rules that handle situations where packets have to be decoded into plain text for the actual SNORT rules to trigger.

Result Quality. In the cybersecurity domain, we measured the accuracy of the results as follows. First, we detected all occurrences of the set of SNORT rules in the log. We then ignored a certain subset of the rules and identified the unexplained situations. Occurrences of ignored PPGs were expected to have a relatively high probability of being unexplained situations, as there is no model for them. We measured the fraction of such occurrences that were correctly flagged as unexplained for different values of τ. Specifically, we considered two settings: one where only ICMP rules were ignored, and another where only preprocessor rules were ignored from a single IP. The results for a log containing 135K tuples, corresponding to 270 minutes of network traffic, are reported in Fig. 8.

The results show good accuracy values. When ICMP rules were ignored, sequences where ICMP activities were occurring were flagged as unexplained situations in the majority of cases—the same happened when preprocessor rules were ignored. As expected, the better accuracy obtained with lower τ-values corresponded to higher processing times: the full log was processed in 1,875 ms. with $\tau = 10^{-4}$, 2,345 ms. with $\tau = 10^{-8}$, and 3,502 ms. with $\tau = 10^{-10}$.

Processing Times. As in the video surveillance case, we measured how processing times varied with length of the network traffic, using an increasing number of compute nodes, with $\tau = 10^{-8}$. Figs. 10(light line) and 11 show the results obtained.

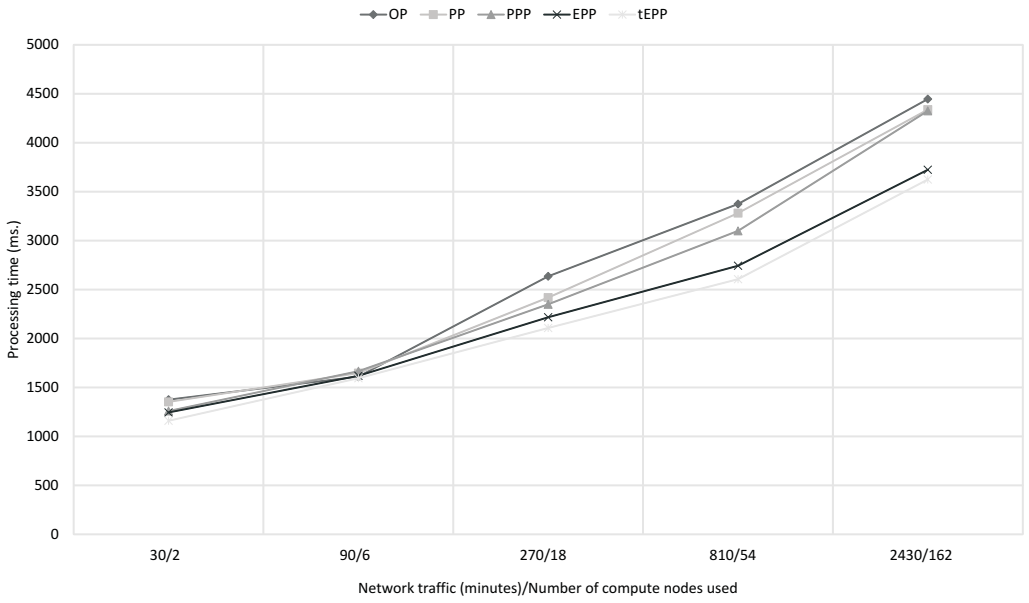

Fig. 11. Processing times for the cybersecurity dataset.

The results confirm that PADUA is able to process up to 335K tuples per second on the largest network traffic length. More importantly, an 81x increase in the traffic length only results in a 3x increase in processing time. Moreover, the different partitioning schemes show similar performance for shorter traffic lengths—for long traffic

lengths, EPP and tEPP appear to be the best schemes. This suggests OP, OPP, and PPP to be better options when dealing with smaller logs and fewer action symbols (as in the case of the video surveillance domain), while EPP and tEPP appear better suited for larger logs and more action symbols (resulting in larger Super-PPGs).

7. CONCLUSIONS

To the best of our knowledge, PADUA is the only parallel approach to date for identifying unexplained situations in historical transaction logs which occur naturally in many domains. Starting with the same activity model as [Albanese et al. 2011b] (extended slightly to incorporate penalties via the notion of a probabilistic penalty graph or PPG), We showed how we can merge a set of PPGs into a single Super-PPG and provided the PPG-Index data structure to store, update, and analyze observations as they come in (e.g., from a video surveillance application or a cybersecurity application). This Super-PPG can then be "split" across K compute nodes in a cluster of $(K + 1)$ nodes in many ways—we present 5 such ways of doing so. The resulting partitioned graph has one part stored on each of K compute nodes with one node serving as a master node. We develop parallel algorithms that achieve coordination amongst these different compute nodes so as to leverage parallelism when detecting unexplained situations.

Our experimental results show great promise. On both video and cybersecurity datasets, we were able to significantly improve on past results for unexplained situation detection. Specifically, our precision, recall, and F-measure go up to 0.96, 0.83, 0.89 compared to 0.73, 0.72, and 0.72 respectively from past work—a significant improvement. Second, our scaling is substantial. We were able to process 500 minutes of video (after image processing) on 32 compute nodes in under 2.5 seconds which is two orders of magnitude better than past work. On the cybersecurity dataset, we were able to process up to 335K observation (log) tuples per second.

REFERENCES

Amit Adam, Ehud Rivlin, Ilan Shimshoni, and David Reinitz. 2008. Robust Real-Time Unusual Event Detection using Multiple Fixed-Location Monitors. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 3 (2008), 555–560.

Safaa O. Al-Mamory and Hongli Zhang. 2009. IDS alerts correlation using grammar-based approach. *Journal of Computer Virology* 5, 4 (November 2009), 271–282.

Massimiliano Albanese, Sushil Jajodia, Andrea Pugliese, and V. S. Subrahmanian. 2011a. Scalable Analysis of Attack Scenarios. In *ESORICS*. Springer, Leuven, Belgium, 416–433.

Massimiliano Albanese, Cristian Molinaro, Fabio Persia, Antonio Picariello, and V. S. Subrahmanian. 2011b. Finding Unexplained Activities in Video. In *IJCAI*. 1628–1634.

Massimiliano Albanese, Vincenzo Moscato, Antonio Picariello, V. S. Subrahmanian, and Octavian Udrea. 2007. Detecting Stochastically Scheduled Activities in Video. In *IJCAI*. 1802–1807.

Carmen E. Au, Sandra Skaff, and James J. Clark. 2006. Anomaly Detection for Video Surveillance Applications. In *ICPR*. 888–891.

Stefano Bordoni, Emilia Reggio, and Gisella Facchinetti. 2001. Insurance Fraud Evaluation - A Fuzzy Expert System. In *FUZZ-IEEE*. IEEE, 1491–1494.

Matthew Brand, Nuria Oliver, and Alex Pentland. 1997. Coupled hidden Markov models for complex action recognition. In *CVPR*. 994–999.

Luc Brun, Alessia Saggese, and Mario Vento. 2012. A Clustering Algorithm of Trajectories for Behaviour Understanding Based on String Kernels. In *SITIS*. 267–274.

Naresh P. Cuntoor, B. Yegnanarayana, and Rama Chellappa. 2008. Activity Modeling Using Event Probability Sequences. *IEEE Transactions on Image Processing* 17, 4 (2008), 594–607.

P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* 28, 1-2 (February-March 2009), 18–28.

Raffay Hamid, Yan Huang, and Irfan Essa. 2003. ARGMode - Activity Recognition Using Graphical Models. In *CVPRW*. 38–43.

Somboon Hongeng, Ramakant Nevatia, and François Brémond. 2004. Video-based event recognition: activity representation and probabilistic recognition methods. *Computer Vision and Image Understanding* 96, 2 (2004), 129–162.

Derek Hao Hu, Xian-Xing Zhang, Jie Yin, Vincent Wenchen Zheng, and Qiang Yang. 2009. Abnormal Activity Recognition Based on HDP-HMM Models. In *IJCAI*. 1715–1720.

Fan Jiang, Ying Wu, and Aggelos K. Katsaggelos. 2009. Detecting contextual anomalies of crowd motion in surveillance video. In *ICIP*. 1117–1120.

Fan Jiang, Junsong Yuan, Sotirios A. Tsaftaris, and Aggelos K. Katsaggelos. 2010. Video anomaly detection in spatiotemporal context. In *ICIP*. 705–708.

A. Jones and S. Li. 2001. Temporal Signatures for Intrusion Detection. In *ACSAC*. IEEE Computer Society, New Orleans, LA, USA, 252–261.

David R. Karger and Clifford Stein. 1996. A New Approach to the Minimum Cut Problem. *J. ACM* 43, 4 (1996), 601–640.

Jaechul Kim and Kristen Grauman. 2009. Observe locally, infer globally: A space-time MRF for detecting abnormal activities with incremental updates. In *CVPR*.

Peter Lancaster and Kestutis Salkauskas. 1986. Curve and surface fitting. An introduction. *London: Academic Press, 1986* 1 (1986).

Dhruv Mahajan, Nipun Kwatra, Sumit Jain, Prem Kalra, and Subhashis Banerjee. 2004. A Framework for Activity Recognition and Detection of Unusual Activities. In *ICVGIP*.

Alessandro Mecocci and Massimo Pannozzo. 2005. A completely autonomous system that learns anomalous movements in advanced videosurveillance applications. In *ICIP*. 586–589.

Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. 1994. Network Intrusion Detection. *IEEE Network* 8, 3 (May 1994), 26–41.

Peng Ning, Yun Cui, and Douglas S. Reeves. 2002. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *CCS 2002*. ACM, Washington, DC, USA, 245–254.

Adam J. Oliner, Ashutosh V. Kulkarni, and Alex Aiken. 2010. Community Epidemic Detection Using Time-Correlated Anomalies. In *RAID (Lecture Notes in Computer Science)*, Somesh Jha, Robin Sommer, and Christian Kreibich (Eds.), Vol. 6307. Springer, Ottawa, Canada, 360–381.

Nuria Oliver, Eric Horvitz, and Ashutosh Garg. 2002. Layered Representations for Human Activity Recognition. In *ICMI*. 3–8.

Girish Keshav Palshikar and Manoj M Apte. 2008. Collusion set detection using graph clustering. *Data Mining and Knowledge Discovery* 16, 2 (2008), 135–164.

Xinzhou Qin. 2005. *A Probabilistic-Based Framework for INFOSEC Alert Correlation*. PhD thesis. Georgia Institute of Technology.

Xinzhou Qin and Wenke Lee. 2003. Statistical Causality Analysis of INFOSEC Alert Data. In *RAID (Lecture Notes in Computer Science)*, Giovanni Vigna, Christopher Kruegel, and Erland Jonsson (Eds.), Vol. 2820. Springer, Pittsburgh, PA, USA, 73–93.

Namrata Vaswani, Amit K. Roy Chowdhury, and Rama Chellappa. 2005. "Shape Activity": A Continuous-State HMM for Moving/Deforming Shapes With Application to Abnormal Activity Detection. *IEEE Transactions on Image Processing* 14, 10 (2005), 1603–1616.

Junbo Wang, Zixue Cheng, Mengqiao Zhang, Yinghui Zhou, and Lei Jing. 2012. Design of a Situation-Aware System for Abnormal Activity Detection of Elderly People. In *AMT*. 561–571.

Lingyu Wang, Anyi Liu, and Sushil Jajodia. 2006. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications* 29, 15 (September 2006), 2917–2933.

Tao Xiang and Shaogang Gong. 2008. Video Behavior Profiling for Anomaly Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 5 (2008), 893–908.

Rui Xu and Donald C. Wunsch II. 2005. Survey of clustering algorithms. *IEEE Transactions on Neural Networks* 16, 3 (2005), 645–678.

Jie Yin, Qiang Yang, and Jeffrey Junfeng Pan. 2008. Sensor-Based Abnormal Human-Activity Detection. *IEEE Trans. Knowl. Data Eng.* 20, 8 (2008), 1082–1090.

Dong Zhang, Daniel Gatica-Perez, Samy Bengio, and Iain McCowan. 2005. Semi-Supervised Adapted HMMs for Unusual Event Detection. In *CVPR*. 611–618.

Xian-Xing Zhang, Hua Liu, Yang Gao, and Derek Hao Hu. 2009. Detecting Abnormal Events via Hierarchical Dirichlet Processes. In *PAKDD*. 278–289.

Hua Zhong, Jianbo Shi, and Mirkó Visontai. 2004. Detecting Unusual Activity in Video. In *CVPR*. 819–826.

Yue Zhou, Shuicheng Yan, and Thomas S. Huang. 2007. Detecting Anomaly in Videos from Trajectory Similarity Analysis. In *ICME*. 1087–1090.

APPENDIX

Example: updating and pruning a PPG-Index

Consider the log of Example 3.5, whose associated sequence of action symbols is ⟨PreFirewallAccess, x, PostFirewallAccess, MobileAppServerAccess, MobileAppServerAccess, OrderProcessingServerAccess, x, PlaceOrder⟩. Assume the first three log tuples are already indexed as in Example 3.4, and that we execute PPG_Insert on the remaining tuples with $\tau = 10^{-3}$. When the first MobileAppServerAccess tuple is handled by PPG_Insert, two index tuples are added to $tables_G$(MobileAppServerAccess) (Fig. 4). The first one completes the sequence ⟨PostFirewallAccess, MobileAppServerAccess⟩ for PPG A_1, so the score decreases by $1 - \delta_G$(PostFirewallAccess, MobileAppServerAccess, A_1) w.r.t. its $previous$ index tuple. The second one completes the sequence ⟨PreFirewallAccess, PostFirewallAccess, MobileAppServerAccess⟩ for PPG A_2, so the score decreases by $1 - \delta_G$(PostFirewallAccess, MobileAppServerAccess, A_2). In both cases, on Lines 7–18 we have v = PostFirewallAccess, a = MobileAppServerAccess, and $z = 0$ because there are no log tuples between PostFirewallAccess and MobileAppServerAccess. The execution of PPG_Insert on the second MobileAppServerAccess tuple produces two additional tuples in $tables_G$(MobileAppServerAccess). The $score$ values for these tuples are further decreased using the penalties associated with the (PostFirewallAccess, MobileAppServerAccess) edges in A_1 and A_2, respectively. This is due to the MobileAppServerAccess log tuple between PostFirewallAccess and MobileAppServerAccess, that must be now interpreted as noise—in this case, $z = 1$. After indexing the remaining log tuples, the situation of the tables is that of Fig. 4. Note that the PlaceOrder log tuple completed an occurrence of A_2 that extended all of the unexplained situations for A_2. Thus, in order to satisfy Condition 3 in Definition 2.3, PPG_Insert (Line 16) removed all the corresponding index tuples from $completed_G(A_2)$. $completed_G(A_1)$ contains instead two pointers to the tuples in $tables_G$(OrderProcessingServerAccess), and we have $count_G = 8$.

Assume now that PPG_Prune is executed on the index and consider tuple $t_{eb} \in tables_G$(PreFirewallAccess). We have $z = count_G - t_{eb}.count - 1 = 6$, so moving from PreFirewallAccess to PostFirewallAccess in A_2 would now make the score decrease by $(1 - 0.1) \cdot (1 - 0.8)^6 = 5.8 \cdot 10^{-5}$. In the case of PlaceOrder, the score would instead decrease by $(1 - 0.9) \cdot (1 - 0.7)^6 = 7.3 \cdot 10^{-5}$. Thus, we have $maxP = 7.3 \cdot 10^{-5}$. Since $t_{eb}.score \cdot maxP < \tau$, we know that t_{eb} can no longer be linked to a new tuple. As a consequence, PPG_Prune sets $t_{eb}.closed$ to $true$, and t_{eb} is no longer considered by PPG_Insert (at Line 8).

Proofs

PROPOSITION 2.10. *Consider a log L, a set $\mathcal{A}$ of PPGs, and two thresholds $\tau_1, \tau_2 \in [0, 1]$. Let U_1 (resp. U_2) be the set of τ_1- (resp. τ_2-) unexplained situations for $\mathcal{A}$. If $\tau_1 \geq \tau_2$, then for every L_u^1 in U_1 there exists L_u^2 in U_2 s.t. L_u^1 is a contiguous subsequence of L_u^2.*

PROOF. Let L_u^1 be a τ_1-unexplained situations for $\mathcal{A}$, i.e., $L_u^1 \in U_1$. Definition 2.9 implies that for every $A \in \mathcal{A}$ there is a τ_1-unexplained situation (L'_u, I'_u) for A s.t. L_u^1 is a subsequence of L'_u. It is easy to check that $\tau_1 \geq \tau_2$ implies that (L'_u, I'_u) is a τ_2-unexplained situation for A. This means that for every $A \in \mathcal{A}$ there is a τ_2-unexplained situation (L'_u, I'_u) for A s.t. L_u^1 is a subsequence of L'_u. The following two cases may occur. *(i)* If L_u^1 is maximal, then the claim trivially holds (we are in the case where $L_u^2 = L_u^1$). *(ii)* If L_u^1 is not maximal, then there exists $L_u^2 \neq L_u^1$ s.t. L_u^1 is a proper contiguous subsequence of L_u^2, and for every $A \in \mathcal{A}$ there is a τ_2-unexplained situation (L'_u, I'_u) for A s.t. L_u^2 is a subsequence of L'_u. □

PROPOSITION 2.11. *Consider a log L, a threshold $\tau \in [0,1]$, and two sets of PPGs $\mathcal{A}_1$, $\mathcal{A}_2$. Let U_1 (resp. U_2) be the set of τ-unexplained situations for $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). If $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$, then for every L_u^1 in U_1 there exists L_u^2 in U_2 s.t. L_u^1 is a contiguous subsequence of L_u^2.*

PROOF. Recall that, by definition of $\sqsubseteq$, if $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$, then for every $A_2 \in \mathcal{A}_2$ there exists $A_1 \in \mathcal{A}_1$ s.t. $A_2 \sqsubseteq A_1$. Notice that if (L_u, I_u) is a τ-unexplained situation for A_1, then (L_u, I_u) is a τ-unexplained situation also for A_2. This is because (L_u, I_u) is clearly an unexplained situation for both A_1 and A_2 (because they are topologically the same and the definition of unexplained situation involves only the topology of the PPG and the log—see Definition 2.5) and the score of (L_u, I_u) computed w.r.t. A_1 is less than or equal to the score of (L_u, I_u) computed w.r.t. A_2.

Suppose $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$ and let L_u^1 be a τ-unexplained situations for $\mathcal{A}_1$. Definition 2.9 implies that for every $A_1 \in \mathcal{A}_1$ there is a τ-unexplained situation (L'_u, I'_u) for A_1 s.t. L_u^1 is a subsequence of L'_u. Suppose by contradiction that there does not exist a τ-unexplained situation L_u^2 for $\mathcal{A}_2$ s.t. L_u^1 is a contiguous subsequence of L_u^2. In other words, neither L_u^1 nor any contiguous subsequence of the log containing L_u^1 is a τ-unexplained situations for $\mathcal{A}_2$. Then, by Definition 2.9, there is at least one PPG A_2 in $\mathcal{A}_2$ s.t. there does not exist a τ-unexplained situation (L'_u, I'_u) for A_2 s.t. L_u^1 is a subsequence of L'_u. The definition of $\sqsubseteq$ implies that there exists $A_1 \in \mathcal{A}_1$ s.t. $A_2 \sqsubseteq A_1$. Since L_u^1 is a τ-unexplained situations for $\mathcal{A}_1$, then there must be a τ-unexplained situation (L'_u, I'_u) for A_1 s.t. L_u^1 is a subsequence of L'_u. As shown at the beginning of the proof, this means that (L'_u, I'_u) is a τ-unexplained sequence for A_2, and, furthermore, L_u^1 is a subsequence of L'_u, which is a contradiction. □

COROLLARY 2.12. *Consider a log L, a threshold $\tau \in [0,1]$, and two sets of PPGs $\mathcal{A}_1$, $\mathcal{A}_2$. Let U_1 (resp. U_2) be the set of τ-unexplained situations for $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). If $\mathcal{A}_2 \subseteq \mathcal{A}_1$, then for every L_u^1 in U_1 there exists L_u^2 in U_2 s.t. L_u^1 is a contiguous subsequence of L_u^2.*

PROOF. It is straightforward to check that $\mathcal{A}_2 \subseteq \mathcal{A}_1$ implies $\mathcal{A}_2 \sqsubseteq \mathcal{A}_1$ (see the definition of $\sqsubseteq$). Thus, the claim follows from Proposition 2.11. □

PROPOSITION 3.6. *Consider a set $\mathcal{A}$ of PPGs, a log $L = \langle l_1, \ldots, l_n \rangle$, and a threshold $\tau \in [0,1]$. Let I_G^i be the PPG-Index returned by PPG_Insert(l_i, I_G^{i-1}, τ). Then:*

$$I_G^i = \textit{PPG\_Insert}(l_i, \textit{PPG\_Prune}(I_G^{i-1}, \tau), \tau).$$

Moreover, PPG_Retrieve(I_G^n) is the set of all τ-unexplained situations for $\mathcal{A}$.

PROOF. We start by showing that the correctness of Line 10 of PPG_Insert, which ignores those tuples in I_G^{i-1} whose $closed$ attribute is $true$, is not affected by the application of PPG_Prune to I_G^{i-1}. PPG_Prune sets $t.closed = true$ when t points to a log tuple with associated action v that end a sequence $L^* \subseteq L$ representing an unexplained situation w.r.t. a PPG $t.A$ with score $t.score$. It is easy to see that the score of any sequence extending L^* cannot exceed $t.score \cdot maxP$, with $maxP$ being the maximum possible value of $(1 - \delta_G(e, t.A)) \cdot (1 - \rho_G(e, t.A))^z$ where e is an outgoing edge of v in the Super-PPG and z is the number of noise actions encountered after indexing t (i.e., $z = count_G - t.count - 1$). As a consequence, based on Definition 2.4, if $t.score \cdot maxP < \tau$, L^* cannot be further extended.

The PPG_Insert algorithm indexes tuple l_i with associated action $l_i.action = a$. If $a \in start(A)$ for some $A \in \mathcal{A}$, the index tuple t' (with $t'.current = l_i$) that is added to $tables_G(a)$ must have no $previous$ pointer, and its score must be set to 1 by Definition 2.4. Moreover, t' by itself represents an unexplained situation, so it is correctly added to $completed_G(A)$. If some $A \in \mathcal{A}$ has an edge from v to a, then the sequence obtained by adding l_i to the sequence represented by any index tuple t in $tables_G(v)$ has a score equal to $t.score \cdot (1 - \delta_G(v, a, A)) \cdot (1 - \rho_G(v, a, A))^z$, with $z = count_G - t.count - 1$.

If this score is above τ and $a \notin end(A)$, then it is correct (*i*) to extend the existing sequence with l_i (obviously, we have $t'.previous = t$ in this case) and (*ii*) to remove t from $completed_G(A)$ based on Condition 4 of Definition 2.4, as it no longer represents an unexplained situation. Finally, if $a \in end(A)$, then it is correct to remove t from $completed_G(A)$ based on Condition 3 of Definition 2.4.

Finally, the correctness of the PPG_Retrieve algorithm immediately follows from the correctness of PPG_Insert and PPG_Prune. In fact, PPG_Retrieve reconstructs all unexplained situations for any PPG A_i by just following backward pointers in $table_G$. It is easy to see that, at the end of each iteration, t_s and t_e are the start and end log tuples of an unexplained situation for A_i—the set of all τ-unexplained situations is then the maximal common subsequence of such unexplained situations by Definition 2.9. □