THEME ARTICLE: SPECIAL ISSUE ON NON-IID OUTLIER DETECTION IN COMPLEX CONTEXTS

Anomalous Event Sequence Detection

Boxiang Dong, *Montclair State University, USA*
Zhengzhang Chen, Lu-An Tang, and Haifeng Chen, *NEC Labs America, USA*
Hui Wang, *Stevens Institute of Technology, USA*
Kai Zhang, *Temple University, USA*
Ying Lin, *University of Houston, USA*
Zhichun Li, *Stellar Cyber, USA*

Anomaly detection has been widely applied in modern data-driven security applications to detect abnormal events/entities that deviate from the majority. However, less work has been done in terms of detecting suspicious event sequences/paths, which are better discriminators than single events/entities for distinguishing normal and abnormal behaviors in complex systems such as cyber-physical systems. A key and challenging step in this endeavor is how to discover those abnormal event sequences from millions of system event records in an efficient and accurate way. To address this issue, we propose NINA, a network diffusion based algorithm for identifying anomalous event sequences. Experimental results on both static and streaming data show that NINA is efficient (processes about 2 million records per minute) and accurate.

Anomaly detection plays an important role in various real-world data-driven applications, such as fraud detection, cyber security, medical diagnosis, and industrial manufacturer.[1–3] The task of anomaly detection is to identify and understand the underlying regularity and irregularity of the vast amount of data. By identifying underlying anomalies or irregular patterns, critical actionable information can be extracted to facilitate human decisions and mitigate the potential hazard.

Although the recent years have witnessed significant progress of anomaly detection techniques,[4–10] the rise of big data has introduced new challenges for the design of efficient and accurate anomaly detection approaches. First, a real complex system typically deals with a large volume of system event data (normally thousands of events per second). The challenge is how to identify abnormal system behaviors from such large-scale (possibly fast streaming) data. Second, the variety of system entity types may necessitate high-dimensional features in subsequent processing. Such enormous feature space could easily lead to the problem, coined by Bellman as "the curse of dimensionality."

More importantly, it is often the case that a *coordinated or sequential, but not independent*, action of multiple system events to determine the system status. This is because system monitoring data are typically low-level events or interactions between various system entities (e.g., a program connects to a server in an enterprise network), whereas abnormal system behaviors are higher level activities, which usually involve multiple events together. For example, a network attack called Advanced Persistent Threat is composed of a set of stealthy and continuous computer hacking processes, by first attempting to gain a foothold in the environment, then using the compromised systems as the access into the target network, followed by deploying additional tools that help fulfill the attack objective. The gap between low-level system events and high-level abnormal activities makes it particularly challenging to identify events that are truly involved in a real malicious activity, and

1541-1672 © 2020 IEEE
Digital Object Identifier 10.1109/MIS.2020.3041174
Date of publication 27 November 2020; date of current version 2 July 2021.

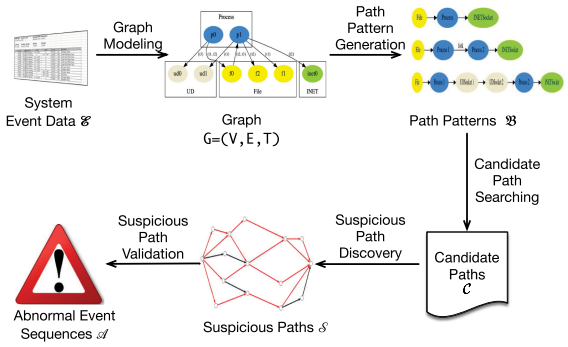


FIGURE 1. Framework of NINA.

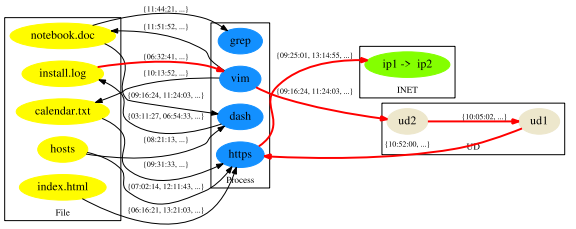


FIGURE 2. Example of a compact graph model for an enterprise network; the red path corresponds to an abnormal event sequence.

especially considering the massive "noisy" events filling the event sequences. Hence, the approaches that identify individual events/entities that confer a given system state are inappropriate to detect sequences of such interactions between different events.

To address these challenges, in this article, we introduce NINA, a **n**etwork d**i**ffusio**n** based **a**nomaly detection technique to capture the abnormal event sequences efficiently and effectively. In particular, we propose a transition probability model to capture the routine entity behavior from the system monitoring graph. An *anomaly score* is calculated for each candidate event sequence that quantifies its "rareness" in comparison with normal profiles. To eliminate the potential score bias from the sequence length, we use the power transformation based approach to normalize the anomaly scores so that the scores of paths with different lengths have the same distribution. To further improve detection efficiency, we design an optimization scheme of the suspicious path discovery method. The optimization scheme allows the graph search procedure to terminate early. It only traverses a small number of paths in the graph to find the most suspicious ones. An extensive set of experiments demonstrate that NINA is efficient (about 2 million records per minute) and accurate.

METHODOLOGY

Overview

Problem Statement: Given the data that contain a set of events $\mathcal{E}$, the user-specified positive integers ℓ, k, and time window size Δt, we aim to find the top k abnormal event sequences in $\mathcal{E}$ that include at most ℓ events occurring within the time period of Δt.

In this article, we propose NINA, a network diffusion based anomaly detection algorithm, that can find abnormal event sequences from a large number of heterogeneous event traces. Figure 1 shows the framework of NINA. In particular, the *graph modeling* component (see the section titled "Graph Modeling") generates a compact graph that captures the complex interactions among event entities, aiming to reduce the computational cost in subsequent analysis components. The *path pattern generation* component constructs the patterns of abnormal event sequences (see the section titled "Path Pattern Generation"). The *candidate path searching* component discovers all the candidate event sequences that may correspond to malicious event sequences that the adversary exploits to disclose sensitive information (see the section titled "Candidate Path Search"). The *suspicious path discovery* component returns those paths of top-k anomaly scores as suspicious paths (see the section titled "Suspicious Path Discovery"). To further reduce false alarms, the *suspicious path validation* component measures the deviation between the suspicious sequences from the normal ones, and mark those sequences as abnormal only if their deviation is sufficiently large (see the section titled "Suspicious Path Validation").

Graph Modeling

The surveillance data from a real-world complex system can be massive. For example, the data collected from a single computer system by monitoring the process interactions in 1 h can easily reach 1 GB. Searching over such massive data is prohibitively expensive in terms of both time and space. Therefore, we devise a compact, graph-based representation of the system event data. Figure 2 shows an example of the compact graph for enterprise surveillance data.

Formally, given the data in a time window, we construct a directed graph $G = (V, E, T)$, with: 1) V as a set of vertices, each representing an entity. For enterprise surveillance data (see the section titled "Experiment Setup"), each vertex of V belongs to any of the following four types: files (F), processes (P), Unix domain sockets (UDSockets) (U), and Internet sockets (INETSockets) (I), namely $V = F \cup P \cup U \cup I$; 2) E as a set of

edges. For each pair of entities (n_i, n_j), if there exists any system event that sends information from n_i to n_j, we construct an edge (v_i, v_j) in the graph, where v_i (v_j) corresponds to n_i (n_j); and 3) T as a set of time stamps. For any edge (v_i, v_j), it is possible that it is associated with multiple timestamps (i.e., the corresponding event happens multiple times). We use $T(v_i, v_j)$ to denote the set of time stamps on which this event has ever happened. Formally, $T(v_i, v_j) = \{e.t | e \in \mathcal{E}, v_i = e.n_b \text{ and } v_j = e.n_d\}$. n_b (n_d) is the source (destination) entity of e.

Path Pattern Generation

The graph constructed by the *graph modeling* component can be densely connected. Path search in such graphs can be time costly. To speed up the path searching procedure, we propose a metapath-based pattern generation.

A metapath[11] based pattern is a path that connects different entity types via a sequence of relations in a heterogeneous graph. Formally, given a graph $G(V, E, T)$, a path pattern B is of the format $\{X_1, \ldots, X_\ell\}$, where each $X_i (1 \leq i \leq \ell)$ is a specific entity type (e.g., P, which can be mapped to any system process). Given a path $p \in G$ and a path pattern B of G, let $p[i]$ and $B[i]$ represent the ith node in p and B, respectively, we say p is *consistent with* B, denoted as $p \prec B$, if: 1) p and B have the same length; and 2) for each i, $p[i] \in B[i]$ (i.e., the specific entity $p[i]$ belongs to the entity type $B[i]$), we say B is a *valid* path pattern if there exists at least one path p in G s.t. that $p \prec B$.

Candidate Path Search

Based on the valid path patterns $\mathcal{B}$, the *candidate path searching* component searches for the paths in G that are consistent with $\mathcal{B}$. Formally, given a set of path patterns $\mathcal{B}$, the *candidate path searching* component aims to find the set of candidate paths $\mathcal{C}$

$$\mathcal{C} = \{p | p \in G, \exists B \in \mathcal{B} \text{ s.t. } p \prec B\}. \tag{1}$$

Besides the consistency requirement, we also impose the following *time-order constraint* on the search procedure, demanding that for each path, its corresponding event sequence must follow the time order. Formally, a path $p = \{n_1, \ldots, n_{r+1}\}$ satisfies the *time-order constraint* if $\forall i \in [1, r-1]$, there exists $t_1 \in T(n_i, n_{i+1})$ and $t_2 \in T(n_{i+1}, n_{i+2})$ such that $t_1 \leq t_2$. This condition enforces the time order in the corresponding event sequences.

A straightforward way to generate candidate paths is to apply the path pattern and time-order constraints in a *breadth-first search*. One scan of the system event graph G is sufficient to find all candidate paths. To reduce the space overhead, we calculate the anomaly score (as discussed in the section titled "Suspicious Path Discovery") for each candidate path once it is discovered, and only save the path in memory if it is in the top-k list.

Suspicious Path Discovery

It is possible that some candidate paths discovered by the *candidate path searching* component are not truly associated with abnormal behaviors. Hence, it is necessary to identify those suspicious paths that are highly likely to be associated with abnormal event sequences among a large set of candidate paths. Our basic idea is to define the anomaly based on both the system entities and the interactions among them. Each path is assigned an *anomaly score* that quantifies the degree of anomaly. Next, we discuss how to calculate the anomaly scores.

First, we assign each system entity two scores, namely, a *sender* score and a *receiver* score. The sender (receiver, resp.) score measures the activeness that the entity serves as an information flow source (destination, resp.) Both sender and receiver scores are computed by following the information flow in the system event graph G. In particular, given the graph G, we produce a $N * N$ square transition matrix A, where N is the total number of entities, and

$$A[i][j] = \text{prob}(v_i \rightarrow v_j) = \frac{|T(v_i, v_j)|}{|\sum_{k=1}^{N} |T(v_i, v_k)||} \tag{2}$$

where $T(v_i, v_j)$ denotes the set of time stamps on which the event between v_i and v_j has ever happened.

Intuitively, $A[i][j]$ denotes the probability that the information flows from v_i to v_j in G. We denote A as

$$A = \begin{vmatrix} & P & F & I & U \\ P & 0 & A^{P\rightarrow F} & A^{P\rightarrow I} & A^{P\rightarrow U} \\ F & A^{F\rightarrow P} & 0 & 0 & 0 \\ I & A^{I\rightarrow P} & 0 & 0 & 0 \\ U & A^{U\rightarrow P} & 0 & 0 & A^{U\rightarrow U} \end{vmatrix} \tag{3}$$

where 0 represents a zero submatrix. Note that the nonzero submatrices of A (3) only appear between processes and files, processes and sockets, as well as UDSockets. This is because, according to the design of Unix systems, information can only flow between these entities.

Let x be the sender score vector, with $x(v_i)$ denoting the node v_i's sender score. Similarly, we use y to denote the receiver score vector. To calculate each

node (entity)'s sender and receiver scores, first, we assign initial scores. We randomly generate the initial vector x_0 and y_0 and iteratively update the two vectors by the following:

$$\begin{cases} x_{m+1}^T = A * y_m^T \\ y_{m+1}^T = A^T * x_m^T \end{cases} \tag{4}$$

where T denotes the matrix transpose.

From (4), we derive

$$\begin{cases} x_{m+1}^T = (A * A^T) * x_{m-1}^T \\ y_{m+1}^T = (A^T * A) * y_{m-1}^T \end{cases}. \tag{5}$$

In (5), we update the two score vectors independently. It is easy to see that the learned scores x_m and y_m depend on the initial score vector x_0 and y_0. Different initial score vectors lead to different learned score values. It is difficult to choose "good" initial score vector in order to learn the accurate sender and receiver scores. However, we find an important property in matrix theory, namely the *steady-state property* of the matrix, to eliminate the effect of x_0 and y_0 on the result scores. Specifically, let M be a general square matrix, and π be a general vector. By repeatedly updating π with

$$\pi_{m+1}^T = M * \pi_m^T \tag{6}$$

there is a possible convergence state such that $\pi_{m+1} = \pi_m$ for sufficiently large m value. In this case, there is only one unique π_n, which can reach the convergence state, i.e.,

$$\pi_n^T = M * \pi_n^T. \tag{7}$$

The convergence state has a good property that the converged vector is only dependent on the matrix M, but independent from the initial vector value π_0. Based on this property, we prefer that the sender and receiver vectors can reach the convergence state. Next, we discuss how to ensure the convergence.

To reach the convergence state, the matrix M must satisfy two conditions: *irreducibility* and *aperiodicity*. As our system event graph G is not always strongly connected, the iteration in (5) may not reach the convergence state. To ensure convergence, we add a *restart matrix* R, which is widely used in random walk on homogeneous graph[12] and bipartite graph.[5] Typically, R is an $N * N$ matrix whose entries are all $\frac{1}{N}$s. With R, we get a new transition matrix $\bar{A}$

$$\bar{A} = (1 - c) * A + c * R \tag{8}$$

where c is a value between 0 and 1. We call c the *restart ratio*. With the restart technique, $\bar{A}$ is guaranteed to be an irreducible and aperiodic matrix. By replacing A with $\bar{A}$ in (5), we are able to get the converged sender score vector x and receiver score vector y. We can also control the convergence rate by controlling the restart rate value. Our experiments show that the convergence often can be reached within ten iterations.

Given a path $p = (v_1, \ldots, v_{r+1})$, based on the sender and receiver score, the anomaly score is calculated as

$$\text{Score}(p) = 1 - NS(p) \tag{9}$$

where $NS(p)$ is the regularity score of the path calculated by the following formula:

$$NS(p) = \prod_{i=1}^{r} x(v_i) * A[i][j] * y(v_{i+1}) \tag{10}$$

where x and y are the sender and receiver vectors, and A is calculated by (3). In (10), $x(v_i) * A[i][j] * y(v_{i+1})$ measures the normality of the event (edge) that v_i sends information to v_{i+1}. Intuitively, any path that involves at least one abnormal event is assigned a high anomaly score.

For each path $p \in \mathcal{C}$, we calculate the anomaly score by (9). However, it is easy to see that longer paths tend to have higher anomaly scores than the shorter paths. To eliminate the score bias from the path length, we *normalize* the anomaly scores so that the scores of paths of different lengths have the same distribution. Let $\mathcal{T}$ denote the normalization function. We use the Box-Cox power transformation function[13] as our normalization function. In particular, let $Q(r)$ denote the set of anomaly scores of r-length paths before normalization. For each score $q \in Q(r)$, we apply

$$\mathcal{T}(q, \lambda) = \begin{cases} \frac{q^\lambda - 1}{\lambda} & : \lambda \neq 0 \\ \log q & : \lambda = 0 \end{cases} \tag{11}$$

where λ is a normalization parameter. Different λ values yield different transformed distributions. Our goal is to find the optimal λ value to make the distribution after normalization as close to the normal distribution as possible (i.e., $\mathcal{T}(Q, \lambda) \sim N(\mu, \sigma^2)$).

Next, we discuss how to compute the optimal λ. First, we assume that such λ exists to make $\mathcal{T}(Q, \lambda) \sim N(\mu, \sigma^2)$. The density of a normalized score is

$$\text{Prob}(\mathcal{T}(q, \lambda)) = \frac{\exp(-\frac{1}{2\sigma^2}(\mathcal{T}(q, \lambda) - \mu)^2)}{\sqrt{2\pi}\sigma}. \tag{12}$$

The profile logarithm likelihood of the normalized distribution is

$$\mathcal{L}(Q,\lambda) = -\frac{n}{2}\log\left(\sum_{i=1}^{n}\frac{(\mathcal{T}(q_i,\lambda)-\mathcal{T}(\bar{q},\lambda))^2}{n}\right) + (\lambda-1)\sum_{i=1}^{n}\log q_i \tag{13}$$

where $\mathcal{T}(\bar{q},\lambda) = \frac{1}{n}\sum_{i=1}^{n}T(q_i,\lambda)$.

To minimize the margin between the normalized distribution and the Gaussian distribution, we find a λ that maximizes the log-likelihood. A possible solution is to take derivatives of $\mathcal{L}(Q,\lambda)$ on λ, and pick λ that makes $\frac{\partial\mathcal{L}}{\partial\lambda}=0$.

Suspicious Path Validation

To further validate the discovered suspicious paths, we calculate the *t-value* between the two groups of paths: the set of candidate but nonsuspicious paths $\mathcal{C}\setminus\mathcal{S}$, and the set of discovered suspicious paths $\mathcal{S}$. The *t*-test returns a confidence score that determines the statistical difference between the two sets of paths. If the confidence score is greater than 0.9 with *p*-value smaller than 0.05, all paths in $\mathcal{S}$ are considered as abnormal paths that are relevant to abnormal behaviors. Otherwise, we treat those paths as normal and do not raise alerts. The suspicious path validation component prevents NINA from sending false alarms when there is no real anomaly at all.

OPTIMIZED SUSPICIOUS PATH DISCOVERY

The suspicious path discovery method (see the section titled "Suspicious Path Discovery") calculates the anomaly score for each candidate path. However, the number of candidate paths can be prohibitively large. It would be desirable if we only need to check a small number of candidate paths to find those suspicious ones. In this section, we devise an **opt**imized scheme *OPT* that addresses this issue by integrating the *threshold algorithm*[14] with our NINA algorithm. The optimized scheme notably improves the efficiency of suspicious path discovery (see the section titled "Static Evaluation").

Intuitively, the top-*k* suspicious paths are those candidate paths with the *k* largest *anomaly score* $\text{Score}(p)$. We observe that the anomaly score function has the *monotone* property. In particular, given two paths p and p' of the same length, where $p=(v_1,\ldots,v_{\ell+1})$, and $p'=(v'_1,\ldots,v'_{\ell+1})$, if $x(v_i)\le x(v'_i)$, $y(v_i)\le y(v'_i)$, and $A[i][j]\le A[i'][j']$ for $i\in[1,\ell+1]$, it must be true that $NS(p)\le NS(p')$, thus $\text{Score}(p)\ge\text{Score}(p')$. Based on the monotone property, we design an efficient procedure to find the top-*k* suspicious paths, without the need to calculate the anomaly score of each path.

Our algorithm is adapted from the well-known *threshold algorithm*.[14] First, we apply random walk on the graph G to calculate the two vectors x and y. Second, for each type of entities, we create two queues sorted in the descendent order of the sender score and the receiver score, respectively. Also, we sort the edges according the probability $A[i][j]$. After that, in each iteration of the WHILE loop, we fetch the entity or edge with the smallest score from each queue, and identify all the valid paths that contain these entities and edges. Assume that there is a path p consisting of these entities and edges, we calculate $\text{Score}(p)$. Apparently, $\text{Score}(p)$ is the highest anomaly score for all the paths that are not explored yet. If $\text{Score}(p)$ is no larger than the minimum anomaly score of all paths in the output *SP*, we stop the iterations and output *SP*. Otherwise, we discover the paths $\mathcal{P}$ that involve at least one un-checked entity that is of the highest score in any queue, and calculate the anomaly scores of these paths. Let the *k*th path $p_k\in SP$ be the path in *SP* that is of the minimal anomaly score. For any path $p\in\mathcal{P}$ such that $\text{Score}(p)>\text{Score}(p_k)$, we replace p_k with p. In other words, if the maximum anomaly score of the paths that have yet discovered is lower than $\text{Score}(p_k)$, the search process can be terminated and the exact solution can be guaranteed. Then, we only need to calculate the anomaly score and perform Box-Cox transformation and *t*-test for a small number of valid paths to find the top-*k* suspicious paths. It has proven that the *threshold algorithm* can correctly find the top *k* answers if the aggregation function is monotone.[14] Therefore, our optimization algorithm can find exact top-*k* suspicious paths efficiently.

EXPERIMENTS

Experiment Setup

Dataset: We use a real-world system monitoring dataset in our experiments. The data were collected from an enterprise network composed of 33 UNIX machines, in a time span of three consecutive days (i.e., 72 h). The sheer size of the dataset is around 157 GB. We consider four different types of system entities: 1) *files*, 2) *processes*, 3) UDSockets, and 4) INETSockets. Each type of entities is associated with a set of attributes and a unique identifier. Two types of events (i.e., interactions between the system entities) are considered in this article: 1) file accessed by the processes; and 2) communication between processes. In total, there are around

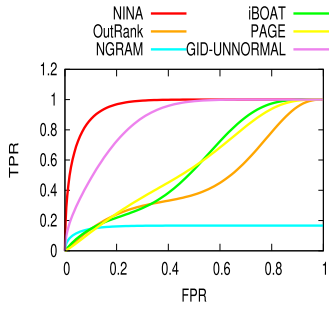


FIGURE 3. ROC curve over static data.

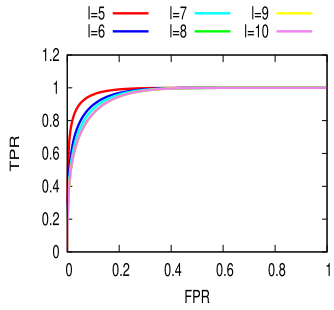


FIGURE 4. ROC curve *w.r.t.* various ℓ.

440 million system events. These events are related to 410 166 processes, 1 797 501 files, 185 076 UDSockets, and 18 391 INETSockets.

Attack Description: There are ten different types of attacks, including Snowden attack, botnet attack, and Trojan attack (see[15] for full attack details), with various lengths from 3 to 5. For each type of attacks, we tried ten attack scenarios at different time slots throughout the data collection period, which results in total 300 event sequences that correspond to intrusion attacks into the data. All the ten types of attacks exploit event sequences to transmit sensitive information to an unauthorized party.

Baseline: We compare our algorithm with a number of state-of-the-art algorithms and the variations of NINA. We briefly introduce these baseline approaches.

- *OutRank*[6]: This approach leverages graph-based approach to detect anomalies from a set of objects.
- *NGRAM*[16]: This method has been widely studied for the identification of attacks and malicious software.
- *iBOAT*[8]: This method has shown its effectiveness in suspicious trajectory discovery in GPS traces.
- *PAGE*: This approach exploits the famous PageRank[12] algorithm to compute the entity score. The anomaly score calculation is similar to (9) and (10), except that $A[i][j]$ is ignored.
- NINA-*UNNORMAL* In this approach, we intentionally avoid the normalization of the anomaly scores of paths with different lengths.

Experiment Settings: We evaluate NINA in the following settings.

1) *Static:* We fetch the events in the monitoring data collected in the tenth hour, and execute the detection algorithms on these events offline. In specific, the monitoring data are fed to the detection algorithms all at once. All the required data are stored in memory. In total, there are *8 million* system events. We launched 12 attacks during the tenth hour.

2) *Streaming:* The monitoring data are delivered to NINA in a streaming fashion, i.e., one system event is processed at a time. NINA updates the graph (in particular, the edge weights) and the sender and receiver scores of all the entities according to the incoming events. A snapshot of the entity scores is retained every hour for evaluation, i.e., we set Δt as 1 h.

Static Evaluation

Detection Accuracy: We compare the detection accuracy of NINA with the baseline approaches. To quantify the detection accuracy, we choose different k values (from 10 to 1000) and compare the detected alerts with the ground truth event sequences related to attacks. Based on the result, we plot the receiver operating characteristic (ROC) curves in Figure 3. We omit the optimized scheme *OPT*, as it has the same accuracy as NINA.

The result demonstrates the effectiveness and accuracy of NINA in detecting the attacks. The accuracy provided by NINA is much better than the baselines.

In practice, it is unrealistic to expect users to provide an accurate estimation of the length ℓ of real attack sequences. Therefore we measure the impact of the choice of ℓ on the detection accuracy. As the length of ground truth event sequences is at most 5, we vary ℓ from 5 to 10, and report the ROC curves in Figure 4. We observe that even though a larger ℓ tends to hamper the accuracy, the effect is quite negligible especially when $\ell \geq 8$. This is because the number of long event sequences is quite small. For example, the number of sequences of length $l = 5$ is 13 675, whereas that for $\ell = 6$ is only 625. Therefore, given the limited number of long sequences, the detection accuracy of NINA is not very sensitive to the choice of ℓ. This makes our approach more plausible in practical scenarios.

Time Performance: We compare the time performance of NINA and the optimized approach *OPT* with the baseline approaches. The result is displayed in

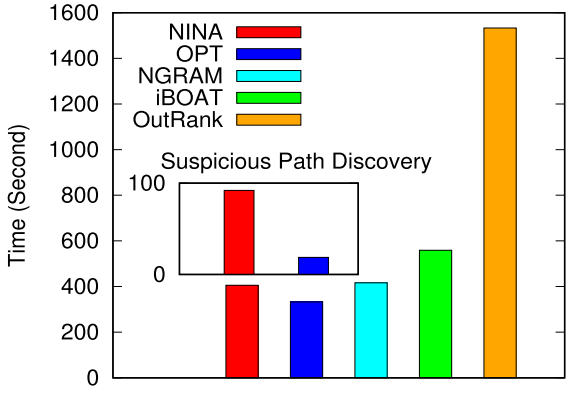


FIGURE 5. Time performance over static data.

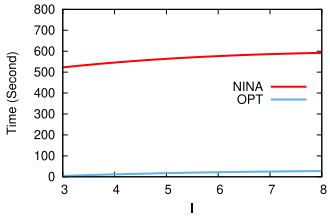


FIGURE 6. Time performance *w.r.t.* various ℓ.

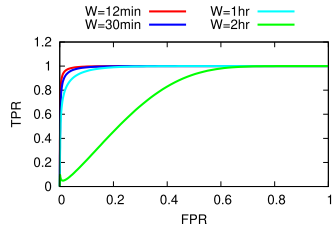
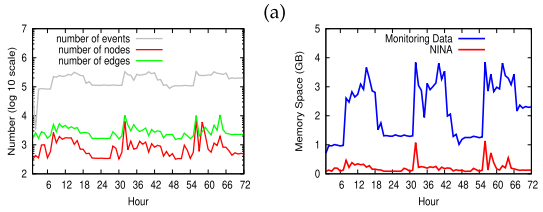


FIGURE 7. ROC curve *w.r.t.* update period.

	Process	File	INETSocket	UDSocket	Edge
Avg.	117.3	191.36	0.93	41.42	1668.4
Max	1468	23290	130	6735	58555

(a)



(b) (c)

FIGURE 8. Graph compactness. (a) Avg./max. number of system entities and edges. (b) Graph size versus number. (c) Memory usage comparison.

Figure 5. We omit the time performance of *PAGE* and NINA-*UNNORMAL* as they are very close to that of NINA. NINA and *OPT* are significantly more efficient than *OutRank*. To emphasize the advantage of *OPT*, we show the time performance of suspicious path discovery for both approaches in Figure 5 (note that we exclude the time consumed by random walk to calculate entity scores). *OPT* saves up to 80% time in the step of suspicious path discovery, and 20% total detection time. The result will be further discussed in the following paragraph.

The main difference between NINA and *OPT* lies in suspicious path discovery, in which *OPT* applies the *threshold algorithm* to avoid examining all the candidate paths. In Figure 6, we display the time performance of suspicious path discovery for both NINA and *OPT*. First, we notice that a larger ℓ leads to a higher time consumption in both approaches. Second, *OPT* is much more efficient than NINA. The reason is that *OPT* avoids discovery and anomaly score computation for a large fraction of candidate paths. When $\ell = 3$, the time taken by *OPT* is only 8% of NINA, due to the large number of candidate paths of small lengths.

Streaming Evaluation

Detection Accuracy: The frequency at which snapshots of the entity scores are updated can have a dramatic impact on the detection accuracy. Let W denote the period to update the snapshot. Intuitively, a smaller W leads to more dynamic updates in the normal profile in locating anomalies from incoming event sequences. In Figure 7, we evaluate the detection accuracy of NINA with regard to various W. It is obvious that NINA yields the best accuracy when the update period W is small. But it is also worth noting that when W is smaller than 1 h, the benefit of decreasing W can be trivial. Given the fact that smaller W induces more overhead in updating the snapshot, we figure out that NINA reaches the best balance between detection accuracy and update overhead when $W = 1$ h. Therefore, in the experiment, we update the snapshot every 1 h.

Memory Usage: To show the compactness of the graph model, we measure the size of the constructed graph.

Figure 8(a) shows the size of our system event graph in terms of the number of system entities and edges. On average, each graph contains around 351 nodes with four different types and less than 1.7k edges. Even for the worst case, the graph is still within the size of 60k edges. Figure 8(b) shows the average size of the graphs for each hour in a 72-h time window. The number of nodes and edges is measured by averaging the size of the 33 graphs (for 33 hosts) in 1 h.

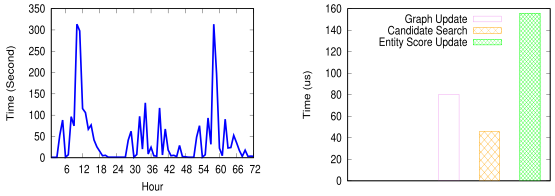


FIGURE 9. Time performance over streaming data. (a) Snapshot update time. (b) Average snapshot update time per event.

The results show that the graph can indeed reduce the space dramatically. Figure 8(c) shows that the memory usage of NINA is around one-tenth of that demanded by the monitoring data.

Time Performance: When executing NINA on the streaming data, the main computational bottleneck stems from the snapshot update. We show the time performance of snapshot update in Figure 9(a). In most cases, the snapshot update is efficient (within 2 min). However, at the peak hours when the incoming events explode, the update can take up to 8 min. We further analyze the update time for each incoming event, and show the result in Figure 9(b). We observe that for each event, the time overhead is negligible. On average, the operations triggered by each event only takes 0.28 ms. Therefore, NINA can be scaled up to 3600 events per second, which is comparable with NGRAM and iBOAT (around 4600 and 3800 events per second, respectively), and is significantly faster than OutRank (around 1200 events per second).

CONCLUSION

In this article, we tackled a novel and challenging problem of anomaly detection on heterogeneous event sequence data. Different from traditional methods that focus on detecting single entities/events, we proposed NINA, a network diffusion based method to capture the interaction behavior among different entities and identify abnormal event sequences. The experimental results on both static and streaming data demonstrated the effectiveness and efficiency of our approach. An interesting direction for further exploration would be applying the proposed framework to other applications (such as financial fraud detection).

REFERENCES

1. K. Padmanabhan, Z. Chen, S. Lakshminarasimhan, S. S. Ramaswamy, and B. T. Richardson, "Graph-based anomaly detection," in *Practical Graph Mining With R*. Boca Raton, FL, USA: CRC Press, 2013, p. 311.
2. S. Wang *et al.*, "Heterogeneous graph matching networks for unknown malware detection," in *Proc. Int. Joint Conf. Artif. Intell. Org.*, 2019, pp. 3762–3770.
3. V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009, Art. no. 15.
4. Z. Chen, W. Hendrix, and N. F. Samatova, "Community-based anomaly detection in evolutionary networks," *J. Intell. Inf. Syst.*, vol. 39, no. 1, pp. 59–85, 2012.
5. J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *Proc. Int. Conf. Data Mining*, 2005, pp. 418–425.
6. H. Moonesignhe and P.-N. Tan, "Outlier detection using random walks," in *Proc. Int. Conf. Tools Artif. Intell.*, 2006, pp. 532–539.
7. Y. Lin *et al.*, "Collaborative alert ranking for anomaly detection," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, 2018, p. 1987–1995.
8. C. Chen *et al.*, "iBOAT: Isolation-based online anomalous trajectory detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 2, pp. 806–818, Jun. 2013.
9. Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, "Detecting and tracking community dynamics in evolutionary networks," in *Proc. IEEE Int. Conf. Data Mining Workshops*, 2010, pp. 318–327.
10. G. Pang, L. Cao, and L. Chen, "Outlier detection in complex categorical data by modelling the feature value couplings," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1902–1908.
11. Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "PathSim: Meta path-based top-k similarity search in heterogeneous information networks," in *Proc. Int. Conf. Very Large Databases*, 2011, pp. 992–1003.
12. L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Stanford Digit. Library Technol. Project, Stanford InfoLab, Stanford, CA, USA, Tech. Rep., 1999, pp. 1–17.
13. J. W. Osborne, "Improving your data transformations: Applying the Box-Cox transformation," *Practical Assessment, Res. Eval.*, vol. 15, pp. 1–9, 2010.
14. R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, pp. 614–656, 2003.
15. C. Cao, Z. Chen, J. Caverlee, L.-A. Tang, C. Luo, and Z. Li, "Behavior-based community detection: Application to host assessment in enterprise information networks," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, 2018, pp. 1977–1985.
16. M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proc. Workshop Cyber-Phys. Syst. Secur.*, 2015, pp. 13–24.

BOXIANG DONG is currently an Assistant Professor with the Computer Science Department, Montclair State University, Montclair, NJ, USA. His research interests include deep learning and cybersecurity. Contact him at bxdong7@gmail.com.

ZHENGZHANG CHEN is currently a Senior Research Scientist with the NEC Labs America, Princeton, NJ, USA. His research interests include data mining, graph AI, machine learning, and their applications. Contact him at zchen@nec-labs.com. He is the corresponding author of this article.

HUI (WENDY) WANG is currently an Associate Professor with the Computer Science Department, Stevens Institute of Technology, Hoboken, NJ, USA. Her research interests include data management, data mining, data security, and privacy. Contact her at hwang4@stevens.edu.

LU-AN TANG is currently a Senior Research Scientist with the NEC Labs America, Princeton, NJ, USA. His research interests include big data analytics and cybersecurity. Contact him at ltang@nec-labs.com.

KAI ZHANG is currently an Associate Professor with the Computer Science Department, Temple University, Philadelphia, PA, USA. His research interests include large-scale machine learning and complex networks. Contact him at zhang.kai@temple.edu.

YING LIN is currently an Assistant Professor with the Industrial Engineering Department, University of Houston, Houston, TX, USA. Her research interests include data analytics, quality engineering, and healthcare. Contact her at ylin53@central.uh.edu.

ZHICHUN LI is currently a Chief Security Scientist with Stellar Cyber, Santa Clara, CA, USA. His research interests include security, system, networking, big-data, and AI related areas. Contact him at zhichun@nec-labs.com.

HAIFENG CHEN is currently the Department Head with the NEC Labs America, Princeton, NJ, USA. His research interests include data management and mining, and artificial intelligence. Contact him at haifeng@nec-labs.com.