Subset Node Anomaly Tracking over Large Dynamic Graphs

Xingzhi Guo
xingzguo@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

Baojian Zhou*
bjzhou@fudan.edu.cn
Fudan University
Shanghai, China

Steven Skiena
skiena@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

ABSTRACT

Tracking a targeted subset of nodes in an evolving graph is important for many real-world applications. Existing methods typically focus on identifying anomalous edges or finding anomaly graph snapshots in a stream way. However, edge-oriented methods cannot quantify how individual nodes change over time while others need to maintain representations of the whole graph all the time, thus computationally inefficient.

This paper proposes DynAnom, an efficient framework to quantify the changes and localize per-node anomalies over large dynamic weighted-graphs. Thanks to recent advances in dynamic representation learning based on Personalized PageRank, DynAnom is 1) *efficient*: the time complexity is linear to the number of edge events and independent of node size of the input graph; 2) *effective*: DynAnom can successfully track topological changes reflecting real-world anomaly; 3) *flexible*: different type of anomaly score functions can be defined for various applications. Experiments demonstrate these properties on both benchmark graph datasets and a new large real-world dynamic graph. Specifically, an instantiation method based on DynAnom achieves the accuracy of 0.5425 compared with 0.2790, the best baseline, on the task of node-level anomaly localization while running 2.3 times faster than the baseline. We present a real-world case study and further demonstrate the usability of DynAnom for anomaly discovery over large-scale graphs.

CCS CONCEPTS

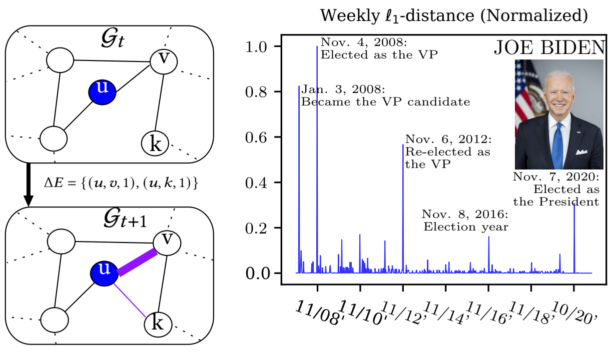
• **Computing methodologies**; • **Information systems** → **Data stream mining**;

KEYWORDS

Dynamic graph; Anomaly detection; Personalized PageRank

ACM Reference Format:
Xingzhi Guo, Baojian Zhou, and Steven Skiena. 2022. Subset Node Anomaly Tracking over Large Dynamic Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539389

*Corresponding author



(a) Dynamic Network (b) Application: Quantify changes of Biden

Figure 1: (a) Two consecutive snapshots $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$ from a dynamic weighted-graph. The inserted edge $(u,v,1)$ further strengthens an existing relation, while edge $(u,k,1)$ builds a new link. (b) Anomaly tracking of Joe Biden over the *Person Knowledge-Graph* using our proposed DynAnom. We track Joe Biden on a weekly basis from 2007 to 2020, and calculate the ℓ_1-distance between the representations on consecutive weeks. The detected peaks correlate well with significant changes in Biden's life.

1 INTRODUCTION

Analyzing the evolution of events in a large-scale dynamic network is important in many real-world applications. We are motivated by the following specific scenario:

> Given a person-event interaction graph containing millions of nodes with several different types (e.g., person, event, location) with a stream of weekly updates over many years, how can we identify when specific individuals significantly changed their context?

The example in Fig. 1 shows our analysis as Joe Biden shifted his career from senator to vice president and finally to president. In each transition he relates with various intensity to other nodes across time. We seek to identify such transitions through analysis of these interaction frequencies and its inherent network structure. This task becomes challenging as the graph size and time horizon scale up. For example, the raw graph data used in Fig. 1 contains roughly 3.2 million nodes and 1196 weekly snapshots (each snapshot averages 4 million edges) of Person Graph from 2000 to 2022.

Despite the extensive literature [2, 7, 38] on graph anomaly detection, previous work focuses on different problem definitions of *anomaly*. On the other hand, works [36] on graph-level anomaly detection cannot identify individual node changes but only uncover the global changes in the overall graph structure. Most representative methods leverage tensor factorization [9] and graph sketching [8], detecting the sudden dis/appearance of dense subgraphs. However, we argue that since most anomalies are locally incurred by a few anomalous nodes/edges, the global graph-level anomaly signal may overlook subtle local changes. Similarly, edge-level anomaly detection [12] cannot identify node evolution when there is no direct edge event associated with that node. When Donald Trump became the president, his wife (*Melania Trump*) changed status to become First Lady, but no explicit edge changes connected her to other politicians except Trump. According to the edge-level anomaly detection, there is no evidence for Melania's status change.

Node representation learning-based methods such as [30, 38] could be helpful to identify anomaly change locally. However, it is impractical to directly apply these methods on large-scale dynamic settings due to 1) the low efficiency: re/training all node embeddings for snapshots is prohibitively expensive; 2) the missing alignment: node representations of each snapshot in the embedding space may not be inherently aligned, making the anomaly calculation difficult.

Inspired by the recent advances in local node representation methods [19, 24] for both static and dynamic graphs, we could efficiently calculate the node representations based on approximated Personalized PageRank vectors. These dynamic node representations are keys for capturing the evolution of a dynamic graph. One important observation is that the time complexity of calculating approximate PPV is $O(\frac{1}{\alpha\epsilon})$ for per queried node, where α is PageRank [23] teleport probability and ϵ is the approximation error tolerance. As the graph evolves, the per-node update complexity is $O(\frac{|\Delta E|}{\epsilon})$, where $|\Delta E|$ is the total edge events. The other key observation is that the representation space is inherently consistent and aligned with each other, thus there is a meaningful metric space defined directly over the Euclidean space of every node representation across different times. Current local node representation method [19] only captures the node-level structural changes over unweighted graphs, which ignores the important interaction frequency information over the course of graph evolution. For example, in a communication graph, the inter-node interaction frequency is a strong signal, but the unweighted setting ignores such crucial feature. This undesirable characteristic makes the node representation becomes less expressive in weighted-graph.

In this paper, we generalize the local node representation method so that it is more expressive and supports dynamic weighted-graph, and the resolve the practical subset node anomaly tracking problem. We summarize our contributions as follows:

- We generalize dynamic forward push algorithm [39] for calculating Personalized PageRank, making it suitable for weighted-graph anomaly tracking. The algorithm updates node representations per-edge event (i.e., weighted edge addition/deletion), and the per-node time complexity linear to edge events ($|\Delta E|$) and error parameter(ϵ) which is independent of the node size.
- We propose an efficient anomaly framework, namely DynAnom that can support both node and graph-level anomaly tracking, effectively localizing the period when a specified node significantly changed its context. Experiments demonstrate its superior performance over other baselines with the accuracy of 0.5425 compared with 0.2790, the baseline method meanwhile 2.3 times faster.
- A new large-scale real-world graph, as new resources for anomaly tracking: PERSON graph is constructed as a more interpretable resource, bringing more research opportunities for both algorithm benchmarking and real-world knowledge discovery. Furthermore, we conduct a case study of it, successfully capturing the career shifts of three public figures in U.S., showing the usability of DynAnom.

The rest paper is organized as follows: Section 2 reviews previous graph anomaly tracking methods. Section 3-4 describes the notation, preliminaries and problem formulation. We present our proposed framework – DynAnom in Section 5, and discuss experimental results in Section 6 Finally, we conclude and discuss future directions in Section 7. Our code and created datasets are accessible at https://github.com/zjlxgxz/DynAnom.

2 RELATED WORK

We review three most common graph anomaly tasks over dynamic graphs, namely graph, edge and node-level anomaly.

Graph-level and edge-level anomaly. Graph anomaly refers to sudden changes in the graph structure during its evolution, measuring the difference between consecutive graph snapshots [1, 5, 12, 13]. Aggarwal et al. [1] use a structural connectivity model to identify anomalous graph snapshots. Shin et al. [29] apply incremental tensor factorization to spot dense connection formed in a short time. Yoon et al. [36] use the first and second derivatives of the global PageRank Vectors as the anomaly signal, assuming that a normal graph stream will evolve smoothly. On the other hand, edge anomaly identifies unexpected edges as the graph evolves where anomalous edges adversarially link nodes in two sparsely connected components or abnormal edge weight changes [6, 12, 26]. Specifically, Eswaran and Faloutsos [12] propose to use the approximated node-pair Personalized PageRank (PPR) score before and after the new edges being inserted. Most recently, Chang et al. [9] estimate the interaction frequency between nodes, and incorporate the network structure into the parameterization of the frequency function. However, these methods cannot reveal the node local anomalous changes, and cannot identify the individual node changes for those without direct edge modification as we illustrated in introduction.

Node-level local anomaly. Node anomaly measures the sudden changes in individual node's connectivity, activity frequency or community shifts [33, 38]. Wang et al. [33] uses hand-crafted features (e.g., node degree, centrality) which involve manual feature engineering processes. Recently, the dynamic node representation learning methods [17, 20, 22, 28, 38, 41] were proposed. For example, a general strategy of them [20, 38] for adopting such dynamic embedding methods for anomaly detection is to incrementally update node representations and apply anomaly detection algorithms over the latent space. To measure the node changes over time, a comparable or aligned node representation is critical. Yu et al. [38] uses auto-encoder and incremental random walk to update the node representations, then apply clustering algorithm to detect the node outliers in each snapshot. However, its disadvantage is that the embedding space is not aligned, making the algorithm only detects anomalies in each static snapshot. Even worse, it is inapplicable to large-scale graph because the fixed neural models is hard to expand without retraining. Similar approaches [20] with more complicated deep learning structure were also studied. These existing methods are not suitable for the subset node anomaly tracking. Instead, our framework is inspired from recent advances in efficient local node representation algorithms [18, 19, 24, 39], which is successful in handling subset node representations over large-scale graph.

3 NOTATIONS AND PRELIMINARIES

Notations. Let $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ be a directed weighted-graph where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{W}$ is corresponding edge weights of $\mathcal{E}$. For each node v, $\text{Nei}(v)$ stands for out-neighbors of v. For all $v \in \mathcal{V}$, the generalized out-degree vector of $\mathcal{V}$ is $\boldsymbol{d}$ where v-th entry $d(v) \triangleq \sum_{u \in \text{Nei}(v)} w_{(v,u)}$ and $w_{(v,u)}$ could be the weight of edge (v,u) in graph $\mathcal{G}$. The generalized degree matrix is then denoted as $\boldsymbol{D} := \text{diag}(\boldsymbol{d})$ and the adjacency matrix is written as $\boldsymbol{A}$ with $\boldsymbol{A}(u,v) = w_{(u,v)}$.

3.1 PPR and Its Calculations

As a measure of the relative importance among nodes, PPR, a generalized version of original PageRank [23], plays an important role in many graph mining tasks including tasks of anomaly tracking. Our framework is built on PPR. Specifically, the Personalized PageRank vector (PPV) of a node s in $\mathcal{G}$ is defined as the following:

Definition 1 (Personalized PageRank Vector (PPV)). *Given a graph $\mathcal{G}(\mathcal{V},\mathcal{E},\mathcal{W})$ with $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. Define the lazy random walk transition matrix $\boldsymbol{P} \triangleq (1-\beta)\boldsymbol{D}^{-1}\boldsymbol{A} + \beta \boldsymbol{I}_n$, $\beta \in [0,1)$ where $\boldsymbol{D}$ is the generalized degree matrix and $\boldsymbol{A}$ is the adjacency matrix of $\mathcal{G}$.[1] Given teleport probability $\alpha \in [0,1)$ and the source node s, the Personalized PageRank vector of s is defined as:*

$$\boldsymbol{\pi}_{\alpha,s} = (1-\alpha)\boldsymbol{P}^\top \boldsymbol{\pi}_{\alpha,s} + \alpha \mathbf{1}_s, \tag{1}$$

where the teleport probability α is a small constant (e.g. $\alpha = .15$), and $\mathbf{1}_s$ is an indicator vector of node s, that is, s-th entry is 1, 0 otherwise. We simply denote PPV of s as $\boldsymbol{\pi}_s$.

Clearly, $\boldsymbol{\pi}_s$ can be calculated in a closed form, i.e., $\boldsymbol{\pi}_s = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{P}^\top)^{-1}\mathbf{1}_s$ but with time complexity $O(n^3)$. The most commonly used method is *Power Iteration* [23], which approximates $\boldsymbol{\pi}_s$ iteratively: $\boldsymbol{\pi}_s^{(t)} = (1-\alpha)\boldsymbol{P}^\top \boldsymbol{\pi}_s^{(t-1)} + \alpha \mathbf{1}_s$. After $t = \lceil \log_{1-\alpha} \epsilon \rceil$ iterations, one can achieve $\|\boldsymbol{\pi}_s - \boldsymbol{\pi}_s^{(t)}\|_1 \le \epsilon$. Hence, the overall time complexity of power iteration is $O(m \log_{1-\alpha} \epsilon)$ with $O(m)$ memory requirement. However, the per-iteration of the power iteration needs to access the whole graph which is time-consuming. Even worse, it is unclear how one can efficiently use power-iteration to obtain an updated $\boldsymbol{\pi}_s$ from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$. Other types of PPR can be found in [32, 34, 35] and references therein.

3.2 Dynamic Forward Push Algorithm

Instead, *forward push algorithm* [3], a.k.a. the bookmark-coloring algorithm [4], approximates $\pi_s(v)$ locally via an approximate $p_s(v)$. The key idea is to maintain solution vector $\boldsymbol{p}_s$ and a residual vector $\boldsymbol{r}_s$ (at the initial $\boldsymbol{r}_s = \mathbf{1}_s, \boldsymbol{p}_s = \mathbf{0}$). When the graph updates from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$, a variant forward push algorithm [39] dynamically maintains $\boldsymbol{r}_s$ and $\boldsymbol{p}_s$. We generalize it to a weighted version of *dynamic forward push* to support dynamic updates on dynamic weighted-graph as presented in Algo. 1. At each local push iteration, it pushes large residuals to neighbors whenever these residuals are significant ($|r_s(u)| > \epsilon d(u)$). Compare with power-iteration, this operation avoids the access of whole graph hence speedup the calculations. Based on [39], we consider a more general setting where the graph could be weighted-graph. Fortunately, this weighted version of DynamicForwardPush still has *invariant property* as presented in the following lemma.

Algorithm 1 DynamicForwardPush [39]

```
1: Input: $\boldsymbol{p}_s, \boldsymbol{r}_s, \mathcal{G}_t, \epsilon, \alpha$
2: while exists $u$ such that $|r_s(u)| > \epsilon d(u)$ do
3:     Push($u$)
4: return $(\boldsymbol{p}_s, \boldsymbol{r}_s)$
5: procedure Push($u$)
6:     $p_s(u) \mathrel{+}= \alpha r_s(u)$
7:     for $v \in \text{Nei}(u)$ do
8:         $r_s(v) \mathrel{+}= \frac{(1-\alpha) r_s(u) w_{(u,v)}}{\sum_{v \in \text{Nei}(u)} w_{(u,v)}}$
9:     $r_s(u) = 0$
```

Lemma 2 (PPR Invariant Property [39]). *Suppose $\boldsymbol{\pi}_s$ is the PPV of node s on graph $\mathcal{G}_t$. Let $\boldsymbol{p}_s$ and $\boldsymbol{\pi}_s$ be returned by the weighted version of DynamicForwardPush presented in Algo. 1. Then, we have the following invariant property.*

$$\pi_s(u) = p_s(u) + \sum_{x \in \mathcal{V}} r_s(x)\pi_s(x), \text{ for all } u \in \mathcal{V}, \tag{2}$$

$$p_s(u) + \alpha r_s(u) = (1-\alpha) \sum_{x \in Nei(u)} \frac{w_{(u,x)} p_s(x)}{d(x)} + \alpha 1_{u=s}, \tag{3}$$

where $1_{u=s} = 1$ if $u = s$, 0 otherwise.

From PPVs to node representations. Obtained PPVs are not directly applicable to our anomaly tracking problem as operations on these n-dimensional vectors is time-consuming. To avoid this, we treat PPVs as the intriguing similarity matrix and project them into low-dimensional spaces using transformations such as SVD [25], random projection [10, 40], matrix sketch[30], or hashing [24] so that the original node similarity is well preserved by the low dimension node representations. We detail this in our main section.

4 PROBLEM FORMULATION

Before we present the subset node anomaly tracking problem. We first define the edge events in the dynamic graph model as the following: A set of edge events ΔE_t from t to $t+1$ is a set of operations on edges, i.e., edge insertion, deletion or weight adjustment while the graph is updating from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$. Mathematically, $\Delta E_t \triangleq \{(u_0, v_0, \Delta w_{(u_0,v_0)}), (u_1, v_1, \Delta w_{(u_1,v_1)}), \ldots, (u_i, v_i, \Delta w_{(u_i,v_i)})\}$ where each $\Delta w_{(u_i,v_i)}$ represents insertion (or weight increment) if it is positive, or deletion (decrement) if it is negative. Therefore, our dynamic graph model can be defined as a sequence of edge events. We state our formal problem definition as the following.

Definition 3 (Subset node Anomaly Tracking Problem). *Given a dynamic weighted-graph $\mathcal{G}_t = \langle \mathcal{V}_t, \mathcal{E}_t, \mathcal{W}_t \rangle, \forall t \in [0,T]$, consisting of initial snapshot $\mathcal{G}_0$ and the following T snapshots that have edge events ΔE_t, $|\Delta E_t| \ge 0$ (e.g., edge addition, deletion, weight adjustment from $t-1$ to t). We want to quantify the status change of a node v in the targeted node subset $v \in \mathcal{S} \triangleq \{v_0, v_1, ..., v_k\}$ from $\mathcal{G}_{t-1}$ to $\mathcal{G}_t$ with the anomaly measure function $f(\cdot,\cdot)$ so that the measurement is consistent to human judgement, or reflects the ground-truth of node status change if available. To illustrate this edge event process, Table 1 presents the process for better understanding.*

[1]Our PPV is defined based on the lazy random walk transition matrix. This definition is equivalent to the one using $\boldsymbol{D}^{-1}\boldsymbol{A}$ but with a different parameter α. Without loss of generality, we use $\beta = 0$ throughout the paper.

Table 1: Illustration of subset node anomaly tracking problem. At each time t, node representations x_i^t are provided, and anomaly of node changes is quantified by $f(x_i^{t-1}, x_i^t)$, which is expected to correlated with the anomaly label if available.

Timestamp	0	1	2	3	...
Edge events	-	ΔE_1	ΔE_2	ΔE_3	...
Snapshots	G_0	G_1	G_2	G_3	...
Score for $v_i \in \mathcal{S}$	-	$f(x_i^0, x_i^1)$	$f(x_i^1, x_i^2)$	$f(x_i^2, x_i^3)$	...
Label for $v_i \in \mathcal{S}$	-	Normal	Normal	Abnormal	...

5 THE PROPOSED FRAMEWORK

To localize the node anomaly across time, our idea is to incrementally obtain the PPVs as the node representation at each time, then design anomaly score function $f(\cdot,\cdot)$ to quantify the PPVs changes from time to time. This section presents our proposed framework DynAnom which has three components: 1) A provable dynamic PPV update strategy extending [39] to weighted edge insertion/deletion; 2) Node level anomaly localization based on incremental PPVs; 3) Graph level anomaly detection based on approximation of global PageRank changes. We first present how local computation of PPVs can be generalized to dynamic weighted graphs, then present the unified framework for node/graph-level anomaly tracking, finally we analyze the complexity of an instantiation algorithm.

5.1 Maintenance of Dynamic PPVs

Multi-edges record the interaction frequency among nodes, and reflect the evolution of the network not only in topological structure, but also in the strength of communities. Previous works [19, 39] focus only on the structural changes over unweighted graph, ignoring the multi-edge cases. In a communication graph (e.g., Email graph), the disadvantage of such method is obvious: as more interactions/edges are observed, the graph becomes denser, or even turn out to be fully-connected in an extreme case. Afterwards, all the upcoming interactions will not change the node status since the ignorance of interaction frequency. This aforementioned scenario motivates us to generalize dynamic PPV maintenance to weighted graph, expanding its usability for more generic graphs and diverse applications. In order to incorporate edge weights into the dynamic forward push algorithm to obtain PPVs, we modify the original algorithm [3, 39] by adding a weighted push mechanism as presented in Algo. 1. Specifically, for a specific node v, at each iteration of the push operation, we update its neighbor residual $r_s(v)$ as follows:

$$r_s(v) \mathrel{+}= \frac{(1-\alpha) r_s(u) w_{(u,v)}}{\sum_{v \in \mathrm{Nei}(u)} w_{(u,v)}}. \tag{4}$$

The modified update of $r_s(v)$ in Equ. (4) efficiently approximate the PPR over static weighted graph with same time complexity as the original one. At time snapshot $\mathcal{G}_t$, the weight of each edge event $\Delta w_{(u,v)}$ could be either positive or negative, representing edge insertion or deletion) at time t so that the target graph $\mathcal{G}_t$ updates to $\mathcal{G}_{t+1}$. However, this set of edge events will break up the invariant property shown in Lemma 2. To maintenance the invariant property on weighted graphs, we present the following Thm. 4, a generalized version from [39]. The key idea is to update $\boldsymbol{p}_s, \boldsymbol{r}_s$ so that updated weights satisfy the invariant property. We present the theorem as follows:

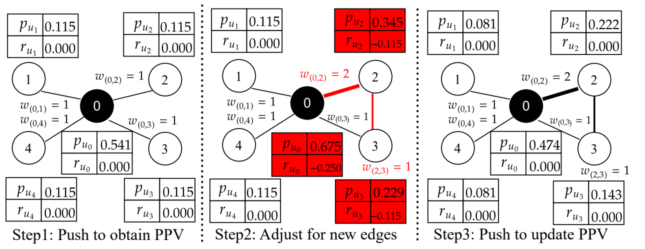


Figure 2: Illustration of the PPR adjustment over a weighted graph of five nodes. *Step1*: Apply Algo.1 to calculate the initial p_s, r_s where $s = v_0, \alpha = 0.15$ over the initial graph. *Step2*: After inserting new edges $(v_0, v_2, 1)$, $((v_2, v_3, 1))$, the strength between v_0 and v_2 increases, and v_2 builds a new connection to v_3, which both break the invariant in Lemma 2. We adjust to recover the invariant by applying Theorem 4 in the red-colored blocks. *Step3:* We re-apply Algo.1 and update p_s for better approximation error.

Generalized PPR update rules. Figure 2 illustrates the dynamic maintenance of PPV over a weighted graph where two new edges are added, incurring edge weight increment between node 0 and 2, and new connection between node 2 and 3. Note that one only needs to have $O(1)$ time to update $p_s(u), r_s(u)$, and $r_s(v)$ for each edge event. As proved, the total cost of m edge events will be linearly dependent on, i.e., $O(m)$. This efficient update procedure is much better than naive updates where one needs to calculate all quantities from scratch.

Theorem 4 (Generalized PPR update rules). *Given an edge event $(u, v, \Delta w_{(u,v)})$. We use the the following update rules*

$$p'_s(u) = p_s(u) \frac{\sum_{v \in \mathrm{Nei}(u)} w_{(u,v)} + \Delta w_{(u,v)}}{\sum_{v \in \mathrm{Nei}(u)} w_{(u,v)}}, \tag{5}$$

$$r'_s(u) = r_s(u) - \frac{\Delta w_{(u,v)} p_s(u)}{\alpha \sum_{v \in \mathrm{Nei}(u)} w_{(u,v)}}, \tag{6}$$

$$r'_s(v) = r_s(v) + \frac{(1-\alpha)}{\alpha} \frac{\Delta w_{(u,v)} p_s(u)}{\sum_{v \in \mathrm{Nei}(u)} w_{(u,v)}}. \tag{7}$$

After applying a sequence of updating rules and using DynamicForwardPush algorithm, then $\boldsymbol{p}_s$ and $\boldsymbol{r}_s$ satisfy invariant property in Lemma 2 and accurately approximate PPV, as proved in Appendix A.

From dynamic PPVs to node representations. The above theorem provides efficient rules to incrementally update PPVs for dynamic weighted graphs. To obtain dynamic node representation, two universal hash functions h_{d} and h_{sgn} are used (see details in [24]). In the following section, we describe how to leverage PPVs as node representation for tracking node/graph-level anomaly.

5.2 Anomaly Tracking Framework: DynAnom

The key idea of our proposed framework is to measure the individual node anomaly in the embedding space based on Personalized PageRank [19, 24], and aggregate the individual anomaly scores to represent graph-level anomaly. By the fact that the PPV-based embedding space is aligned, we can directly measure the node anomaly by any distance function.[2] We present our design as follows:

[2]In this paper, we explored ℓ_1-distance function where given $\boldsymbol{x}, \boldsymbol{y}$, $f(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_1$.

5.2.1 *Node-level Anomaly Tracking.*

Since the inherently local characteristic of our proposed framework, we can efficiently measure the status of arbitrary nodes in two consecutive snapshots in an incrementally manner. Therefore, we could efficiently solve the problem defined in Def. 3 and localize where the anomaly actually happens by ranking the anomalous scores. To do this, the key ingredient is to measure node difference from $t-1$ to t as $\delta_s^t = f(\boldsymbol{x}_s^{t-1}, \boldsymbol{x}_s^t)$ where $\boldsymbol{x}_s^t$ is the node representation of node s at time t. Based on the above motivation, we first present the node-level anomaly detection algorithm DynAnom in Algo. 2 where the score function $f(\cdot,\cdot)$ is realized by ℓ_1 norm in our experiments. There are three key steps: *Step 1.* calculating dynamic PPVs for all nodes in S over T snapshots (Line 3); *Step 2.* obtaining node representations of S by local hash functions. Two universal hash functions[3] are used in DynNodeRep (Line 5); *Step 3.* calculating node anomaly scores based on node representation $\boldsymbol{x}_s$ for all $s \in S$.

Noted that our design directly use the first-order difference (∇x_s) between two consecutive time. Although there are more complex design of $f(\cdot,\cdot)$ under this framework, such as, second-order measurement $\nabla^2 x_s$ similar as in Yoon et al. [36], clustering based anomaly detection [38], or supervised anomaly learning. We restrict our attention on the simplest form by setting $p = 1$ or 2 with the first-order measure. In our framework, the definition of anomaly score can be highly flexible for various applications.

5.2.2 *Graph-level Anomaly Tracking.*

Based on the node-level anomaly score proposed in Algo. 2, we propose the graph-level anomaly. The key property we used for graph-level anomaly tracking is the linear relation property of PPV. More specifically, let $\boldsymbol{\pi}$ be the PageRank vector. We note that the PPR vector is linear to personalized vector $\boldsymbol{\pi}_s$. Therefore, global PageRank could be calculated by the weighted average of single source PPVs.

$$\boldsymbol{\pi} = \alpha \frac{\vec{1}}{|\mathcal{V}|} \sum_{i=0}^{\infty} (1-\alpha)^i \boldsymbol{P}^i = \sum_{s\in\mathcal{V}} \frac{1_s}{|\mathcal{V}|} \boldsymbol{\pi}_s = \frac{1}{|\mathcal{V}|} \sum_{s\in\mathcal{V}} \boldsymbol{\pi}_s, \qquad (8)$$

From the above linear equality, we formulate the global PPV, denoted as $\pi_{,g}$,as the linear combination of single-source PPVs as following:

$$\boldsymbol{\pi}_g = \sum_{s\in\mathcal{V}} \gamma_s \pi_s, \gamma_s = \frac{d(s)}{m}, m = \sum_{i\in\mathcal{V}} d(i) \qquad (9)$$

In order to capture the graph-level anomaly, we use the similar heuristic weights [36] $\gamma_s = \frac{d(s)}{m}$, which implies that $\boldsymbol{\pi}_\alpha$ is dominated by high degree nodes, thus the anomaly score is also dominated by the greatest status changes from high degree nodes as shown below:

$$\begin{aligned} \|\boldsymbol{\pi}_g^t - \boldsymbol{\pi}_g^{t-1}\|_1 &= \| \sum_{s\in\mathcal{V}} \gamma_s^{t-1} \boldsymbol{\pi}_s^{t-1} - \sum_{s\in\mathcal{V}} \gamma_s^t \boldsymbol{\pi}_s^t \|_1 \\ &= \| \sum_{s\in\mathcal{V}} \frac{d_s^{t-1}}{m^{t-1}} \boldsymbol{\pi}_s^{t-1} - \sum_{s\in\mathcal{V}} \frac{d_s^t}{m^t} \boldsymbol{\pi}_s^t \|_1 \\ &\approx \sum_{s\in\mathcal{V}_{high}^t} \frac{d_s^t}{m^t} \|\boldsymbol{\pi}_s^{t-1} - \boldsymbol{\pi}_s^t\|_1 \\ &\le \sum_{s\in\mathcal{V}_{high}^t} \|\boldsymbol{\pi}_s^{t-1} - \boldsymbol{\pi}_s^t\|_1, \end{aligned} \qquad (10)$$

[3] We use *sklearn.utils.murmurhash3_32* in our implementation.

Algorithm 2 DynAnom$(\mathcal{G}_0, \Delta E_{1,\dots,T}, \mathcal{S}, \epsilon, \alpha, p)$

```
1:  Input: Initial graph $\mathcal{G}_0$, Edge events $\Delta E_{1,\dots,T}$ , Subset target
    nodes $\mathcal{S}$, PPV quality $\epsilon$, teleport factor $\alpha$.
2:  //Step 1: Incrementally calculate PPVs
3:  $\boldsymbol{p}_{s\in\mathcal{S}}$ = IncrementPush($\mathcal{G}_0, \Delta E_{1,\dots,T}, S, \epsilon, \alpha$)
4:  //Step 2: Obtain node representations in target set $\mathcal{S}$
5:  $\boldsymbol{x}_{s\in\mathcal{S}}$ = DynNodeRep($p_s^0, \dots, P_s^T$)
6:  //Step 3: Calculate node anomaly for all nodes in $\mathcal{S}$
7:  for $s \in S, t \in [1,T]$ do
8:      $\delta_s^t = f(x_s^{t-1}, x_s^t) = \left(\sum_i \left|x_s^t(i) - x_s^{t-1}(i)\right|^p\right)^{1/p}$
9:  return $\boldsymbol{\delta}_s = [\delta_s^1, \delta_s^2, \dots, \delta_s^T], \forall s \in S$

10: procedure IncrementPush($\mathcal{G}_0, \Delta E_{1,\dots,T}, \mathcal{S}, \epsilon, \alpha$)
11:     $t = 0$
12:     for $s \in \mathcal{S} := \{v_1, v_2, \dots, v_k\}$ do
13:         $p_s^t = \boldsymbol{0}, \quad r_s^t = 1_s$
14:         $p_s^t, r_s^t$ =DynamicForwardPush($\mathcal{G}^0, s, p_s^t, r_s^t, \epsilon, \alpha$)
15:     for $t \in [1,T]$ do
16:         for $s \in \mathcal{S}$ do
17:             for $(u, v, \Delta w_{(u,v)}) \in \Delta E_t$ do
18:                 update $p_s^t(u), r_s^t(u), r_s^t(v)$ uses rules in Thm. 4.
19:             $p_s^t, r_s^t$ =DynamicForwardPush($\mathcal{G}^0, s, p_s^t, r_s^t, \epsilon, \alpha$)
20:         $\mathcal{G}_t$+ = $(u, v, \Delta w_{(u,v)})$
21:     return $\boldsymbol{p}_s = [p_s^0, \dots, p_s^T], \forall s \in S$
22: procedure DynNodeRep($\boldsymbol{p}_s^0, \dots, \boldsymbol{p}_s^T$)
23:     $\epsilon_c$ = MIN($\frac{1}{|\mathcal{V}|}$, 1e-5), $dim = 1024$
24:     for $t \in [1,T)$ do
25:         for $i \in \cup_{t'\in\{t,t-1\}}$SUPP$(p_s^{t'})$ do:
26:             $p_s^{t-1}(i) = 0$ if $p_s^{t-1}(i) \le \epsilon_c$
27:             $p_s^t(i) = 0$ if $p_s^t(i) \le \epsilon_c$
28:         $x_s^t$ = ReduceDim($\frac{p_s^t}{\|p_s^t\|_1}, dim$)
29:         $x_s^{t-1}$ = ReduceDim($\frac{p_s^{t-1}}{\|p_s^{t-1}\|_1}, dim$)
30:         return $\boldsymbol{x}_s = [x_s^0, \dots, x_s^T], \forall s \in S$
31: procedure ReduceDim($x, dim$)
32:     // Hash function $h_{dim}(i) : \mathbb{N} \to [dim]$
33:     // Hash function $h_{\text{sgn}}(i) : \mathbb{N} \to \{\pm 1\}$
34:     if DIM$(x) \le dim$ then
35:         return $x$
36:     else
37:         $\bar{x} = \boldsymbol{0} \in \mathcal{R}^{dim}$
38:         for $i \in$ Supp$(x)$ do
39:             $\bar{x}(h_{dim}(i))$ += $h_{\text{sgn}}(i) \log(x(i))$
40:         return $\frac{\bar{x}}{\|\bar{x}\|_1}$
```

where $\mathcal{V}_{high}^t$ denotes the set of high-degree nodes. Note that the approximate upper bound of $\|\boldsymbol{\pi}_g^t - \boldsymbol{\pi}_g^{t-1}\|_1$ can be lower bounded :

$$\sum_{s\in\mathcal{V}_{high}^t} \|\boldsymbol{\pi}_s^{t-1} - \boldsymbol{\pi}_s^t\|_1 \ge \text{MAX}(\{\left\|\boldsymbol{\pi}_s^{t-1} - \boldsymbol{\pi}_s^t\right\|_1, \forall s \in \mathcal{V}_{high}^t\}). \quad (11)$$

By the approximation of global PPV difference in Equ. (11), we discover that the changes of high degree nodes will greatly affect ℓ_1-distance, and similarly for ℓ_2-distance. The above analysis assumes $d_s^{t-1}/m^{t-1} \approx d_s^t/m^t$. We can also hypothesize that the changes in high-degree node will flag graph-level anomaly signal. Therefore, we use high-degree node tracking strategy for graph-level anomaly:

$$\textsc{DynAnom}^t_{graph} := \textsc{max}(\{\|\boldsymbol{\pi}_s^{t-1} - \boldsymbol{\pi}_s^t\|_1, \forall s \in \mathcal{V}_{high}\}). \quad (12)$$

In practice, we incrementally add high degree nodes (e.g., top-50) into the tracking list from each snapshots and extract the maximum PPV changes among them as the graph-level anomaly score. This proposed score could capture the anomalous scenario where one of the top nodes (e.g., popular hubs) encounter great structural changes, similar to the DDoS attacks on the internet.

5.2.3 Comparison between DynAnom and other methods. The significance of our proposed framework compared with other anomaly tracking methods is illustrated in Table 2. DynAnom is capable of detecting both node and graph-level anomaly, supporting all types of edge modifications, efficiently updating node representations (independent of $|\mathcal{V}|$), and working with flexible anomaly score functions customized for various downstream applications.

Table 2: The supported features of DynAnom and other baselines.

Feature / Method	Anomaly		Edge Event Types			Algorithm		
	Node level	Graph level	Edge Stream	Add/ Delete	Weight Adjust	$\lvert\mathcal{V}\rvert$ Ind.	Align Repr.	Flex. Score
SedanSpot	✗	✗	✓	✗	✗	✓	✗	✗
AnomRank	✗	✓	✗	✓	✓	✗	✗	✗
NetWalk	✓	✗	✗	✓	✗	✗	✗	✗
DynPPE	✓	✗	✓	✓	✗	✓	✓	✗
DynAnom	✓	✓	✓	✓	✓	✓	✓	✓

Time Complexity Analysis. The overall complexity of tracking subset of k nodes across T snapshots depends on run time of three main steps as shown in Algo. 2: 1. the run time of IncrementPush for nodes $\mathcal{S}$; 2. the calculation of dynamic node representations, which be finished in $O(kT|\operatorname{supp}(\boldsymbol{p}_{s'})|)$ where $|\operatorname{supp}(\boldsymbol{p}_{s'})|$ is the maximal support of all k sparse vectors, i.e. $|\operatorname{supp}(\boldsymbol{p}_{s'})| = \max_{s\in\mathcal{S}}|\operatorname{supp}(\boldsymbol{p}_s)|$. The overall time complexity of our proposed framework is stated as in the following theorem.

Theorem 5 (Time Complexity of DynAnom). *Given the set of edge events where $\mathbb{E} = \{e_1, e_2, \ldots, e_m\}$ and T snapshots and subset of target nodes $\mathcal{S}$ with $|\mathcal{S}| = k$, the instantiation of our proposed DynAnom detects whether there exist events in these T snapshots in $O(km/\alpha^2 + k\bar{d}^t/(\epsilon\alpha^2) + k/(\epsilon\alpha) + kTs)$ where $\bar{d}^t$ is the average node degree of current graph $\mathcal{G}_t$ and $s \triangleq \max_{s\in\mathcal{S}}|\operatorname{supp}(\boldsymbol{p}_s)|$.*

Note that the above time complexity is dominated by the time complexity of IncrementPush following from [19]. Hence, the overall time complexity is linear to the size of edge events. Notice that in practice, we usually have $s \ll n$ due to the sparsity we controlled in DynNodeRep.

6 EXPERIMENTS

In this section, we demonstrate the effectiveness and efficiency of DynAnom for both node-level and graph-level anomaly tasks over three state-of-the-art baselines, followed by one anomaly visualization of ENRON benchmark, and another case study of a large-scale real-world graph about changes in person's life.

6.1 Datasets

DARPA: The network traffic graph [21]. Each node is an IP address, each edge represents network traffic. Anomalous edges are manually annotated as network attack (e.g., DDoS attack). **ENRON**[11, 27]: The email communication graph. Each node is an employee in Enron Inc. Each edge represents the email sent from one to others. There is no human-annotated anomaly edge or node. Following the common practice used in [9, 36], we use the real-world events to align the discovered highly anomalous time period. **EU-CORE**: A small email communication graph for benchmarking. Since it has no annotated anomaly, we use two strategies (Type-s and Type-W), similar to [12], to inject anomalous edge into the graph. **EU-CORE-S** includes anomalous edges, creating star-like structure changes in 20 random snapshots. Likewise, **EU-CORE-L:** is injected with multiple edges among randomly selected pair of nodes, simulating node-level local changes. We describe details in appendix. E. **Person Knowledge Graph (PERSON)**: We construct PERSON graph from EventKG [14, 15]. Each node represents a person or an event. The edges represent a person's participation history in events with timestamps. The graph has no explicit anomaly annotation. Similarly, we align the detected highly anomalous periods of a person with the real-world events, which reflects the person's significant change as his/her life events update. We summarize the statistics of dataset in the following table:

Table 3: Dataset Statistics. Graphs are converted into undirected by repeating edges with reversed source and destination node. We detect anomalies after the initial snapshot (Init. t).

Dataset	DARPA	EU-CORE	ENRON	PERSON
$\lvert V\rvert$	25,525	986	151	609,930
$\lvert E\rvert$	4,554,344	333,734	50,572	8,782,630
$\lvert\mathcal{G}_{0,\ldots,T}\rvert$	1463	115	1138	43
Init. t	256	25	256	20

6.2 Baseline Methods

We compare our proposed method to four representative state-of-the-art methods over dynamic graphs: **Graph-level method: AnomRank**[36] [4] uses the global Personalized PageRank and use 1-st and 2-nd derivative to quantify the anomaly score of graph-level. We use the node-wise AnomRank score for the node-level anomaly, **Edge-level method: SadenSpot** [12] approximates node-wise PageRank changes before and after inserting edges as the edge-level anomaly score. **Node-level method: NetWalk** [38] incrementally updates the node embeddings by extending random walks. However, since the *anomaly* defined in the original paper is the outlier in each static snapshot, we re-alignment [16, 31] NetWalk embeddings and

[4]We discovered potential bugs in the official code, which make the results different from the original paper. We have contacted authors for further clarification.

apply ℓ_2-distance for node-level anomaly. [5] **Node-level DynPPE** [19] used the local node embeddings based on PPVs for unweighted graph. We extend its core PPV calculation algorithm for weighted graphs. The detailed hyper-parameter settings are listed in Table 9 in appendix. In the following experiments, we demonstrate that our proposed algorithm outperforms over the strong baselines.

6.3 Exp1: Node-level Anomaly Localization

Experiment Settings: As detailed in Def.3, given a sequence of graph snapshots $\mathcal{G} = \{\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_T\}$ and a tracked node subset $S = \{u_0, u_1, ..., u_i\}$. We denote the set of anomalous snapshots of node u as ground-truth, such that Y_u, containing k_u timestamps where there is at least one new anomalous edge connected to node u in that time. We calculate the node anomaly score for each node. For example for node u: $\boldsymbol{\delta}_u = \{\hat{\delta}_u^1, \hat{\delta}_u^2, \dots, \hat{\delta}_u^T\}$, and rank its anomaly scores across time. Finally, we use the set of the top-k_u highest anomalous snapshots $\hat{Y_u}$ as the predicted anomalous snapshot [6] and calculate the averaged prediction precision over all nodes as the final performance measure as shown below:

$$Precision_{avg} = \frac{1}{|S|} \sum_{u \in S} \frac{\hat{Y_u} \cap Y_u}{|\hat{Y_u}|}$$

For DARPA dataset, we track 151 nodes[7] which have anomalous edges after initial snapshot. Similarly we track 190,170 anomalous nodes over EU-CORE-S and EU-Core-L. We exclude edge-level method SedanSpot for node-level task because it can not calculate node-level anomaly score. We present the precision and running time in Table 4 and Table 5.

Table 4: The average precision of node-level anomaly localization. Our proposed DynAnom outperforms other baselines.

	DARPA	EU-CORE-S	EU-CORE-L
Random	0.0075	0.0142	0.0088
AnomRank	0.2790	0.2173	0.4019
AnomRankW	0.2652	0.2213	0.4078
NetWalk	OOM	0.0421	0.0416
DynPPE	0.1701	0.2478	0.2372
DynAnom	**0.5425**	**0.4242**	**0.5215**

Effectiveness. In Table 4, the low scores of *Random*[8] demonstrate the task itself is very challenging. We note that our proposed DynAnom outperforms all other baselines, demonstrating its effectiveness for node-level anomaly tracking.

Scalability. NetWalk hits Out-Of-Memory (OOM) on DARPA dataset due to the Auto-encoder (AE) structure where the parameter size is related to $|\mathcal{V}|$. Specifically, the length of input/output one-hot vector is $|\mathcal{V}|$, making it computationally prohibitive when dealing with graphs of even moderately large node set. Moreover, since the AE structure is fixed once being trained, there is no easy way to incrementally expand neural network to handle the new nodes in dynamic graphs. While DynAnom can easily track with any new nodes by adding extra dimensions in $\boldsymbol{p}_s, \boldsymbol{r}_s$ and $\boldsymbol{d}$ and apply the same procedure in an online fashion.

Power of Node-level Representation. The core advantage of DynAnom over AnomRank(W) is the power of node-level representation. AnomRank(W) essentially represents individual node's status by one element in the PageRank vector. While DynAnom uses node representation derived from PPVs, thus having far more expressivity and better capturing node changes over time.

Advantage of Aligned Space. Despite that NetWalk hits OOM on DARPA dataset, we hypothesize that the reason of its poor performance on EU-CORE{S,L} is the embedding misalignment. The incremental training makes the embedding space not directly comparable from time to time even after extra alignment. Although it's useful for classification or clustering within one snapshot, the misaligned embeddings may cause troublesome overhead for downstream tasks, such as training separate classifiers for every snapshot.

Importance of Edge Weights. Over all datasets, our proposed DynAnom consistently outperforms DynPPE, which is considered as the unweighted counterpart of DynAnom. It demonstrates the validity and versatility of our proposed framework.

Table 5: For comparing running time (in seconds), excluding the time for data loading and graph update.

	DARPA	EU-CORE-S	EU-CORE-L
AnomRank(W)	905.983	26.084	26.865
NetWalk	OOM	3649.14	3644.58
DynPPE	84.247	33.835	30.933
DynAnom	379.334	30.054	26.359

Efficiency. Table 5 presents the wall-clock time [9]. Deep learning based NetWalk is the slowest. Although it can incrementally update random walk for the new edges, it has to re-train or fine-tune for every snapshot. At the first glance, DynPPE achieves the shortest running time on DAPRA dataset, but it is an illusive victory caused by dropping numerous multi-edge updates at the expense of precision. Furthermore, the Power Iteration-based AnomRank(W) is slow on DARPA as we discussed in Section 3, but the speed difference may not be evident when the graph is small (e.g., DARPA has 25k nodes, and EU-CORE has less than 1k nodes). Moreover, our algorithm can independently track individual nodes as a result of its local characteristic, it could be further speed up by embarrassingly parallelization on a computing cluster with massive cores.

6.4 Exp2: Graph-level Anomaly Detection

Experiment Setting: Given a sequence of graph snapshots $\mathcal{G} = \{\mathcal{G}_0, \mathcal{G}_1, ..., \mathcal{G}_T\}$, we calculate the snapshot anomaly score and consider the top ones as anomalous snapshots, adopting the same experiment settings in [36]. We use both ℓ_1 and ℓ_2 as distance function to test the practicality of DynAnom. We modify edge-level SedanSpot for Graph-level anomaly detection by aggregating edge anomaly score with Mean() [10] in each snapshot. We describe detailed hyper-parameter settings in Appendix. D. For DAPRA dataset,

[5]We use Procrustes analysis to find optimal scale, translation and rotation to align

[6]In experiment, we assume that the number of anomalies is known for each node to calculate the prediction precision.

[7]We track totally 200 nodes, but 49 nodes have all anomalous edges before the initial snapshot, so we exclude them in precision calculation.

[8]For Random baseline, we randomly assign anomaly score for each node across snapshots

[9]We track function-level running time by CProfile, and remove those time caused by data loading and dynamic graph structure updating

[10]For a fair comparison, we tried $\{Mean(), Median(), Min(), Max()\}$ operators for SedanSpot, and found that $Mean()$ yields the best results.

as suggested by Yoon et al. [36], we take the precision at top-250 [11] as the performance measure in Table 6, and present Figure 3 of recall-precision curves as we vary the parameter $k \in \{50, 100, \ldots, 800\}$ for top-k snapshots considered to be anomalous.

Practicality for Graph-level Anomaly: The precision-recall curves show that DynAnom have a good trade-off, achieving the competitive second/third highest precision by predicting the highest top-150 as anomalous snapshots, and the precision decreases roughly at a linear rate when we includes more top-k snapshots as anomalies. Table 6 presents the precision at top-250, and we observe that DynAnom could outperform all other strong baselines, including its unweighted counterpart – DynPPE. It further demonstrates the high practicality of the proposed flexible framework even with the simplest $\ell_1$and ℓ_2 distance function.

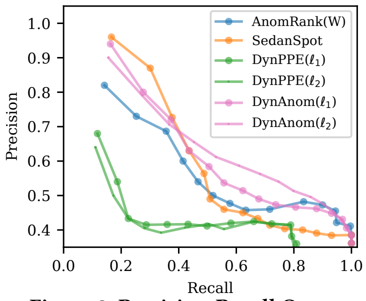


Figure 3: Precision-Recall Curve

Table 6: The precision of top-250 anomalies

Algorithms	Precision
NetWalk	OOM
AnomRankW	0.5400
SedanSpot	0.5640
DynPPE(ℓ_1)	0.4160
DynPPE(ℓ_2)	0.3920
DynAnom(ℓ_1)	0.5840
DynAnom(ℓ_2)	**0.6120**

ENRON graph records the emails involved in the notorious Enron company scandal, and it has been intensively studied in [9, 12, 36]. Although ENRON graph has no explicit anomaly annotation, a common practice [9, 37] is to visualize the detected anomaly score against time and discover the associated real-world events. Likewise, Figure 4 plots the detected peaks side-by-side with other strong baselines [12]. We find that DynAnom shares great overlaps with meaningful peaks and detects more prominent peak at Marker.1 when Enron got investigated for the first time, demonstrating its effectiveness for sensible change discovery . For better interpretation, we annotate the peaks associated to the events collected from the Enron Timeline [13], and present some milestones as followings:

1 :(2000/08/23) Investigated Enron when stock was high.
6 :(2001/08/22) CEO was warned of accounting irregularities.
10 :(2002/01/30) CEO was replaced after bankruptcy.

6.5 Case Study: Localize Changes in Person's life over Real World Graph

To demonstrate the scalability and usability of DynAnom, we present an interesting case study over our constructed large-scale PERSON graph, which records the structure and intensity of person-event interaction history. We apply DynAnom to track public figures (Joe Biden, Arnold Schwarzenegger, Al Franken) of the U.S.

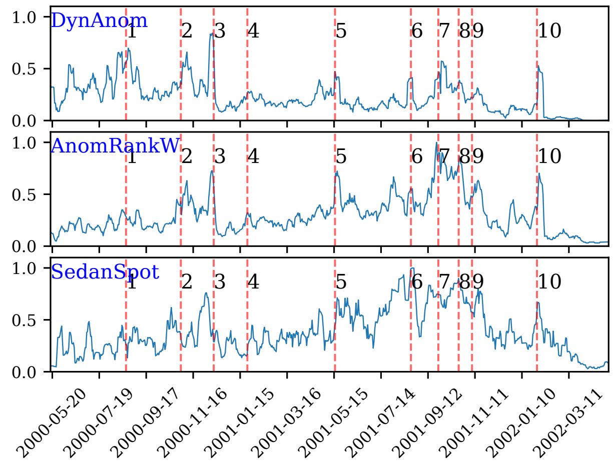


Figure 4: The anomaly scores of ENRON graph. DynAnom well correlates with other baselines, and detect novel peaks at beginning.

from 2000 to 2022 on a yearly basis, and visualize their changes by ℓ_1-distance, together with annotated peaks in Figure 5.

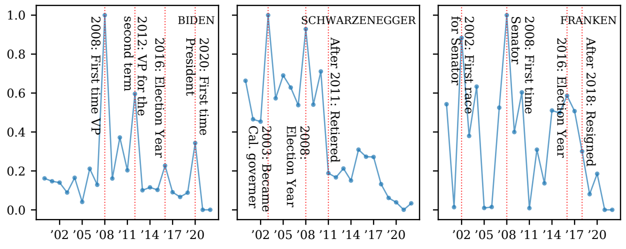


Figure 5: The yearly changes of public figures in PERSON graph from 2000-2022: The detected peaks are well correlated to the major events of the individuals as we annotated.

As we can see, *Biden* has his greatest peak in year 2008 when he became the 47-th U.S. Vice President for the very first time, which is considered to be his first major advancement. The second peak occurs in 2012 when he won the re-election. This time the peak has smaller magnitude because he was already the VP, thus the re-election caused less difference in him, compared to his transition in 2008. The third peak is in 2020 when he became the president, the magnitude is even smaller as he had been the VP for 8 years without huge context changed. Similarly, the middle sub-figure captures *Arnold Schwarzenegger*'s transition to be California Governor in 2003, and high activeness in election years. *Al Franken* is also a famous comedian who shifted career to be a politician in 2008, the peaks well capture the time when he entered political area and the election years. Table 8 lists full events in Appendix C. This resources could bring more opportunities for knowledge discovery and anomaly mining studies in the graph mining community.

Moreover, based on our current Python implementation, it took roughly 37.8 minutes on a 4-core CPU machine to track the subset of three nodes over the PERSON graph, which has more than 600k nodes and 8.7 million edges. Both presented results demonstrate that our proposed method has great efficiency and usability for practical subset node tracking.

[11] $k = 250$ reflects its practicality since it is the closest k without exceeding total 288 graph-level anomalies
[12] All values are post-processed by dividing the maximum value. For SedanSpot, we use $Mean()$ to aggregate edge anomaly score for the best visualization. Other aggregator (e.g., $Max()$) may produce flat curve, causing an unfair presentation.
[13] Full events are included in Table 7 in Appendix. Enron timeline: https://www.agsm.edu.au/bobm/teaching/BE/Enron/timeline.html

7 DISCUSSION AND CONCLUSION

In this paper, we propose an unified framework DynAnom for subset node anomaly tracking over large dynamic graphs. This framework can be easily applied to different graph anomaly detection tasks from local to global with a flexible score function customized for various application. Experiments show that our proposed framework outperforms current state-of-the-art methods by a large margin, and has a significant speed advantage (2.3 times faster) over large graphs. We also present a real-world PERSON graph with an interesting case study about personal life changes, providing a rich resource for both knowledge discovery and algorithm benchmarking. For the future work, it remains interesting to explore different type of score functions, automatically identify the interesting subset of nodes as better strategies for tracking global-level anomaly, and further investigate temporal-aware PageRank as better node representations.

ACKNOWLEDGMENTS

This work was partially supported by NSF grants IIS-1926781, IIS-1927227, IIS-1546113 and OAC-1919752.

REFERENCES

[1] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. 2011. Outlier detection in graph streams. In *International Conference on Data Engineering*. IEEE, 399–409.

[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 410–421.

[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *The IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 475–486.

[4] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

[5] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *International World Wide Web Conference*. 119–130.

[6] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: Microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 3242–3249.

[7] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. 2021. MSTREAM: Fast Anomaly Detection in Multi-Aspect Streams. In *Proceedings of the Web Conference 2021*. 3371–3382.

[8] Siddharth Bhatia, Mohit Wadhwa, Philip S Yu, and Bryan Hooi. 2021. Sketch-Based Streaming Anomaly Detection in Dynamic Graphs. *arXiv preprint arXiv:2106.04486* (2021).

[9] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. 2021. F-fade: Frequency factorization for anomaly detection in edge streams. In *ACM International Conference on Web Search and Data Mining*. 589–597.

[10] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and accurate network embeddings via very sparse random projection. In *International Conference on Information and Knowledge Management*. 399–408.

[11] W.W. Cohen. [n.d.]. Enron email dataset. http://www.cs.cmu.edu/ enron/. Accessed in 2009.

[12] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *IEEE International Conference on Data Mining (ICDM)*. IEEE, 953–958.

[13] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1378–1386.

[14] Simon Gottschalk and Elena Demidova. 2018. EventKG: A Multilingual Event-Centric Temporal Knowledge Graph. In *Proceedings of the Extended Semantic Web Conference (ESWC 2018)*. Springer.

[15] Simon Gottschalk and Elena Demidova. 2019. EventKG - the Hub of Event Knowledge on the Web - and Biographical Timeline Generation. *Semantic Web Journal (SWJ)* 10, 6, 1039–1070.

[16] John C Gower. 1975. Generalized procrustes analysis. *Psychometrika* 40, 1 (1975), 33–51.

[17] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273* (2018).

[18] Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. 2017. Parallel personalized pagerank on dynamic graphs. *VLDB Endowment* 11, 1 (2017), 93–106.

[19] Xingzhi Guo, Baojian Zhou, and Steven Skiena. 2021. Subset Node Representation Learning over Large Dynamic Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 516–526.

[20] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1269–1278.

[21] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. 2000. The 1999 DARPA off-line intrusion detection evaluation. *Computer networks* 34, 4 (2000), 579–595.

[22] Yuanfu Lu, Xiao Wang, Chuan Shi, Philip S Yu, and Yanfang Ye. 2019. Temporal network embedding with micro-and macro-dynamics. In *International Conference on Information and Knowledge Management*. 469–478.

[23] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[24] Ştefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. 2020. InstantEmbedding: Efficient Local Node Representations. *arXiv preprint arXiv:2010.06992* (2020).

[25] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. In *International World Wide Web Conference*. 1509–1520.

[26] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. 2016. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *IEEE International Conference on Data Mining (ICDM)*. SIAM, 189–197.

[27] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the AAAI conference on artificial intelligence.*

[28] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *ACM International Conference on Web Search and Data Mining*. 519–527.

[29] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. Densealert: Incremental dense-subtensor detection in tensor streams. In *ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1057–1066.

[30] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. 2021. FREDE: anytime graph embeddings. *VLDB Endowment* 14, 6 (2021), 1102–1110.

[31] Chang Wang and Sridhar Mahadevan. 2008. Manifold alignment using procrustes analysis. In *International Conference on Machine Learning (ICML)*. 1120–1127.

[32] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 505–514.

[33] Teng Wang, Chunsheng Fang, Derek Lin, and S Felix Wu. 2015. Localizing temporal anomalies in large evolving graphs. In *IEEE International Conference on Data Mining (ICDM)*. SIAM, 927–935.

[34] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *International Conference on Information and Knowledge Management*. 441–456.

[35] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *Proceedings of the 2021 International Conference on Management of Data*. 1996–2008.

[36] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 647–657.

[37] Minji Yoon, Woojeong Jin, and U Kang. 2018. Fast and accurate random walk with restart on dynamic graphs with guarantees. In *International World Wide Web Conference*. 409–418.

[38] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2672–2681.

[39] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized pagerank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1315–1324.

[40] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale network embedding with iterative random projection. In *IEEE International Conference on Data Mining (ICDM)*. IEEE, 787–796.

[41] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

A PROOF OF THM. 4

Proof. By the invariant property in Lemma 2, we have

$$p_s(i) + \alpha r_s(i) = (1-\alpha) \sum_{x \in N^{in}(i)} \frac{w_{(x,i)} p_s(x)}{d(x)} + \alpha \times 1_{i=s}, \forall i \in \mathcal{V}, \tag{13}$$

Initially, this invariant holds and keeps $\frac{r_s(i)}{d(i)} \le \epsilon$ after applying Algo.1. When the new edge $(u, v, \Delta w_{(u,v)})$ arrives, it changes the out-degree of u, breaking up the balance for all invariant involving $d(u)$. Our goal is to recover such invariant by adjusting small amount of p_s, r_s. While such adjustments may compromise the quality of PPVs, we could incrementally invoke Algo.1 to update p_s and r_s for better accuracy afterwards. We denote the initial vectors as p_s, r_s, d and post-change vectors as p'_s, r'_s, d'.

As $d'(u)$ is involved in $p_s(u)/d'(u)$, one need to have $p'_s(u)$ such that $p_s(u)/d(u) = p'_s(u)/d'(u)$, i.e. the invariant maintenance of Equ. (13). The updated amount of weight is $\delta w_(u,v)$, which indicates $p'_s(u) = (d(u) + \Delta w_{(u,v)}) p_s(u)/d(u)$. So, we have

$$p'_s(u) = p_s(u) \frac{\sum_{v \in \text{Nei}(u)} w_{(u,v)} + \Delta w_{(u,v)}}{\sum_{v \in \text{Nei}(u)} w_{(u,v)}}. \tag{14}$$

Hence, we reach Equ. 14 implies that we should scale up $p_s(u)$ to recover balance. This strategy is the general case of Zhang et al. [39] for unweighted graphs where $\Delta w_{(u,v)} = 1$.

However, once we assign $p'_s(u)$, it breaks the invariant for u since $p_s(u) + \alpha r_s(u) \ne p'_s(u) + \alpha r_s(u)$. Likewise, we maintain this equality by introducing $r'_s(u)$:

$$\begin{aligned} p_s(u) + \alpha r_s(u) &= p'_s(u) + \alpha r'_s(u) \\ &\text{Substitute } p'_s(u) \text{ from eq.5} \\ &= p_s(u) + \frac{\Delta w_{(u,v)} p_s(u)}{d(u)} + \alpha r'_s(u), \\ \Longrightarrow \alpha r'_s(u) - \alpha r_s(u) &= -\frac{\Delta w_{(u,v)} p_s(u)}{d(u)}, \\ r'_s(u) &= r_s(u) - \frac{\Delta w_{(u,v)} p_s(u)}{\alpha d(u)} \end{aligned} \tag{15}$$

Equ. (15) implies that we should decrease $r_s(u)$. From a heuristic perspective, it's consistent to the behavior of BCA-algorithm which keeps taking mass from r_s to p_s. In this updating case, we artificially create mass for p'_s at the expense of r_s's decrements. When $\Delta w_{(u,v)} = 1$ in case of unweighted graphs, this update rule is also equivalent to Zhang et al. [39].

So far, we have the updated $p'_s(u)$ and $r'_s(u)$ so that all nodes (without direct edge update, except for v) keep the invariant hold. However, due to the introduction of $\Delta w_{(u,v)}$, the shares of mass pushed to v is changes, breaking the invariant for v as shown below:

$$p_s(v) + \alpha r_s(v) \ne (1-\alpha)\Big(\frac{(w_{(u,v)} + \Delta w_{(u,v)}) p'_s(u)}{d'(u)} + \sum_{x \in N^{in}(v) \setminus \{u\}} \frac{w_{(u,x)} p_s(x)}{d(x)}\Big) + \alpha \times 1_{t=s}$$

In order to recover the balance with minimal effort, we should update $r_s(v)$ instead of $p_s(v)$. The main reason is that any change in $p_s(v)$ will break the balance for v's neighbors, similar to the breaks incurred by the change of $p_s(u)$. We present the updated $r'_s(v) = r_s(v) + \Delta$ as following:

$$p_s(v) + \alpha\Big(r_s(v) + \Delta\Big) = (1-\alpha)\Big(\frac{\Delta w_{(u,v)} p'_s(u)}{d'(u)} + \sum_{x \in N^{in}(v)} \frac{w_{(u,x)} p_s(x)}{d(x)}\Big) + \alpha 1_{t=s}. \tag{16}$$

Note that $p'_s(u)/d'(u) = p_s(u)/d(u)$, and

$$p_s(v) + \alpha r_s(v) = (1-\alpha) \sum_{x \in N^{in}(v)} \frac{w_{(x,v)} p_s(x)}{d(x)}.$$

Reorganize Equ. (16) and cancel out $p_s(v), r_s(v)$, we have

$$\begin{aligned} \Delta &= \frac{(1-\alpha)}{\alpha} \frac{\Delta_{w_{(u,v)}} p_s(u)}{d(u)} \\ r'_s(v) &= r_s(v) + \frac{(1-\alpha)}{\alpha} \frac{\Delta_{w_{(u,v)}} p_s(u)}{d(u)} \end{aligned} \tag{17}$$

Combining Equ. (14),(15), and (17), we prove the theorem. □

Remark 6. *The above proof mainly follows from [39] where unweighted graph is considered while we consider weighted graph in our problem setting.*

B PROOF OF THM. 5

Before we proof the theorem, we present the known time complexity of IncrementPush as in the following lemma.

Lemma 7 (Time complexity of IncrementPush [19]). *Suppose the teleport parameter of obtaining PPR is α and the precision parameter is ϵ. Given current weighted graph snapshot $\mathcal{G}_t$ and a set of edge events ΔE_t with $|\Delta E_t| = m$, the time complexity of IncrementPush is $O(m/\alpha^2 + \bar{d}^t/(\epsilon\alpha^2) + 1/(\epsilon\alpha))$ where $\bar{d}^t$ is the average node degree of current graph $\mathcal{G}_t$.*

Proof. The total time complexity of our instantiation DynAnom has components, which correspond to three steps in Algo. 2. The run time of step 1, IncrementPush is $O(m/\alpha^2 + \bar{d}^t/(\epsilon\alpha^2) + 1/(\epsilon\alpha))$ by Lemma 7. The run time of step 2, DynNodeRep is bounded by $O(n)$ as the component of DynNodeRep is the procedure of the dimension reduction by using two hash functions. The calculation of two has functions given $\boldsymbol{p_s}$ is linear on $|\operatorname{supp}(\boldsymbol{p_s})|$, which is $|\operatorname{supp}(\boldsymbol{p_s})| \le n$. Finally, the instantiation of our score function is also linear on n. Therefore, the whole time complexity is dominated by $O(km/\alpha^2 + k\bar{d}^t/(\epsilon\alpha^2) + k/(\epsilon\alpha) + kTs)$ where s is the maximal allowed sparsity defined in the theorem. We proof the theorem. □

C TIMELINES OF REAL WORLD GRAPHS

We list the real-world events for the ENRON and PERSON graph in table 7 and 8.

D HYPER-PARAMETER SETTINGS

we (re)implement the algorithms in Python to measure comparable running time. We list the hyper-parameter in Table 9.

Table 9: The hyper-parameter configurations of algorithms in experiments.

Algorithm	Hyper-parameter Settings
AnomRank	alpha = 0.5 (default) , epsilon = 1e-2 (default) or 1e-6 (for better accuracy)
SedanSpot	sample-size = 500 (default settings)
NetWalk	epoch_per_update=10, dim=128, default settings for the rest parameters
DynAnom & DynPPE	For exp1 and exp2: alpha = 0.15, minimum epsilon = 1e-12 , dim=1024; For exp2: we keep track of the top-100 high degree nodes for graph-level anomaly. For case studies: alpha = 0.7, the rest are the same.

Table 7: The events in Enron scandal timeline

Index	Date	Event Description
1	2000/08/23	Stock hits all-time high of $90.56. the Federal Energy Regulatory Commission orders an investigation.
2	2000/11/01	FERC investigation exonerates Enron for any wrongdoing in California.
3	2000/12/13	Enron announces that president and chief operating officer Jeffrey Skilling will take over as chief executive in February. Kenneth Lay will remain as chairman.
4	2001/01/25	Analyst Conference in Houston, Texas. Skilling bullish on the company. Analysts are all convinced.
5	2001/05/17	"Secret" meeting at Peninsula Hotel in LA – Schwarzenegger, Lay, Milken.
6	2001/08/22	Ms Watkins meets with Lay and gives him a letter in which she says that Enron might be an "elaborate hoax."
7	2001/09/26	Employee Meeting. Lay tells employees: Enron stock is an "incredible bargain." "Third quarter is looking great."
8	2001/10/22	Enron acknowledges Securities and Exchange Commission inquiry into a possible conflict of interest related to the company's dealings with the partnerships.
9	2001/11/08	Enron files documents with SEC revising its financial statements to account for $586 million in losses. The company starts negotiations to sell itself to head off bankrutcy.
10	2002/01/30	Stephen Cooper takes over as Enron CEO.

Table 8: The person events of *Schwarzenegger* and *Franken*

Year	Major Event
	Arnold Schwarzenegger
2003	Arnold Schwarzenegger won the California gubernatorial election as his first politician role
2008	Arnold began campaigning with McCain as the key endorsement for McCain's presidential campaign
2010	mid-term election
2011	Arnold reached his term limit as Governor and returned to acting
	Al Franken
2002	Al Franken consider his first race for office due to the tragedy of Minnesota Sen. Paul Wellstone.
2004	The Al Franken Show aired
2007	Al Franken announced for candidacy for Senate.
2008	Al Franken won election.
2012	Al Franken won re-election.
2018	Al Franken resigned

E EXPERIMENT DETAILS

Infrastructure: We conduct our experiment on machine with 4-core Intel i5-6500 3.20GHz CPU, 32 GB memory, and GeForce GTX 1070 GPU (8 GB memory) on Ubuntu 18.04.6 LTS.

Datasets: For DARPA dataset: we totally track 200 anomalous nodes, and 151 of them have at least one anomalous edge after initial snapshot, the ground-truth of node-level anomaly is derived from the annotated edge as aforementioned. For EU-CORE dataset: Since EU-CORE dataset does not have annotated anomalous edges, we randomly select 20 snapshots to inject artificial anomalous edges using two popular injection methods, which are similar to the approach used in [36]. For each selected snapshot in *EU-CORE-S*, we uniformly select one node,u_{high}, from the top 1% high degree nodes, and injected 70 multi-edges connecting the selected node to other 10 random nodes which were not connected to u_{high} before, which simulates the structural changes. In each selected snapshot in *EU-CORE-L*, we uniformly select 5 pairs of nodes, and injected edge connect each pair with totally 70 multi-edges as anomaly, which simulates the sudden peer-to-peer communication. We include all datasets in our supplementary materials.