Real-Time Anomaly Detection in Edge Streams

SIDDHARTH BHATIA, RUI LIU, and BRYAN HOOI, National University of Singapore, Singapore
MINJI YOON, Carnegie Mellon University, United States
KIJUNG SHIN, KAIST, United States
CHRISTOS FALOUTSOS, Carnegie Mellon University, United States

Given a stream of graph edges from a dynamic graph, how can we assign anomaly scores to edges in an online manner, for the purpose of detecting unusual behavior, using constant time and memory? Existing approaches aim to detect *individually surprising* edges.

In this work, we propose Midas, which focuses on detecting *microcluster anomalies*, or suddenly arriving groups of suspiciously similar edges, such as lockstep behavior, including denial of service attacks in network traffic data. We further propose Midas-F, to solve the problem by which anomalies are incorporated into the algorithm's internal states, creating a "poisoning" effect that can allow future anomalies to slip through undetected. Midas-F introduces two modifications: (1) we modify the anomaly scoring function, aiming to reduce the "poisoning"effect of newly arriving edges; (2) we introduce a conditional merge step, which updates the algorithm's data structures after each time tick, but only if the anomaly score is below a threshold value, also to reduce the "poisoning"effect. Experiments show that Midas-F has significantly higher accuracy than Midas.

In general, the algorithms proposed in this work have the following properties: (a) they detects microcluster anomalies while providing theoretical guarantees about the false positive probability; (b) they are online, thus processing each edge in constant time and constant memory, and also processes the data orders-of-magnitude faster than state-of-the-art approaches; and (c) they provides up to 62% higher area under the receiver operating characteristic curve than state-of-the-art approaches.

CCS Concepts: • **Computing methodologies** → **Anomaly detection**; • **Security and privacy** → *Intrusion detection systems;*

Additional Key Words and Phrases: Anomaly detection, streaming, real-time, dynamic graphs, edge streams, microcluster

ACM Reference format:
Siddharth Bhatia, Rui Liu, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2022. Real-Time Anomaly Detection in Edge Streams. *ACM Trans. Knowl. Discov. Data.* 16, 4, Article 75 (January 2022), 22 pages.
https://doi.org/10.1145/3494564

1 INTRODUCTION

Anomaly detection in graphs is a critical problem for finding suspicious behavior in innumerable systems, such as intrusion detection, fake ratings, and financial fraud. This has been a

Authors' addresses: S. Bhatia, R. Liu, and Bryan Hooi, National University of Singapore: 21 Lower Kent Ridge Road, Singapore 119077, Singapore; emails: siddharth@comp.nus.edu.sg, xxliuruiabc@gmail.com, bhooi@comp.nus.edu.sg; M. Yoon and C. Faloutsos, Carnegie Mellon University: 5000 Forbes Avenue, Pittsburgh, PA 15213; emails: {minjiy, christos}@cs.cmu.edu; K. Shin, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea; email: kijungs@kaist.ac.kr.

1556-4681/2022/01-ART75 $15.00
https://doi.org/10.1145/3494564

well-researched problem with majority of the proposed approaches [3, 13, 22, 23, 25, 41] focusing on static graphs. However, many real-world graphs are dynamic in nature, and methods based on static connections may miss temporal characteristics of the graphs and anomalies.

Among the methods focusing on dynamic graphs, most of them have edges aggregated into graph snapshots [17, 20, 26, 43–45]. However, to minimize the effect of malicious activities and start recovery as soon as possible, we need to detect anomalies in real time or near real time, i.e., to identify whether an incoming edge is anomalous or not, as soon as we receive it. In addition, since the number of vertices can increase as we process the stream of edges, we need an algorithm that uses constant memory in the graph size.

Moreover, fraudulent or anomalous events in many applications occur in microclusters, suddenly arriving groups of suspiciously similar edges, e.g., denial of service attacks in network traffic data and lockstep behavior. However, existing methods that process edge streams in an online manner, including [16, 36], aim to detect individually surprising edges, not microclusters, and can thus miss large amounts of suspicious activity. It is worth noting that in other literature, microcluster may have different meanings [1, 5, 27], while we specifically refer to a group of sudden arriving edges.

In this work, we propose Midas, which detect *microcluster anomalies*, or suddenly arriving groups of suspiciously similar edges, in edge streams. Consider the example in Figure 1 of a single source-destination pair (u, v), which shows a large burst of activity at time 10. This burst is the simplest example of a microcluster, as it consists of a large group of edges that are very similar to one another. The Midas algorithm uses **count-min sketches (CMS)** [15] to count the number of occurrences in each timestamp, then use the chi-squared test to evaluate the degree of deviation and produce a score representing the anomalousness. The higher the score, the more anomalous the edge is. The proposed method uses constant memory and has a constant time complexity processing each edge. Additionally, by using a principled hypothesis testing framework, Midas provides theoretical bounds on the false positive probability, which those methods do not provide.

We then propose a relational variant Midas-R, which incorporates temporal and spatial relations. In the base version of the Midas algorithm, the CMS is cleared after every timestamp change. However, some anomalies persist for multiple timestamps. Maintaining partial counts of previous timestamps to the next allows the algorithm to quickly produce a high score when the edge occurs again. This variant also considers the source and destination nodes as additional information that helps determine anomalous edges.

We also notice that in the base version, edges are merged into the CMS without considering whether the edge is anomalous. However, those anomalous edges will increase the historical mean level of counts, poisoning the scores of future edges. Therefore, we propose the condition merge to prevent this. Also, we simplify the formula of the anomaly score, which grants more efficiency to the detector. Then, we proposed the filtering Midas, or Midas-F, integrating those changes, and further improves the accuracy of the algorithm.

Our main contributions are as follows:

(1) Streaming Microcluster Detection: We propose a novel streaming approach combining statistical (chi-squared test) and algorithmic (count-min sketch) ideas to detect microcluster anomalies, requiring constant time and memory.
(2) Theoretical Guarantees: In Theorem 1, we show guarantees on the false positive probability of Midas.
(3) Effectiveness: Our experimental results show that Midas outperforms baseline approaches by up to 62% higher **area under the receiver operating characteristic curve (ROC-AUC)**, and processes the data orders-of-magnitude faster than baseline approaches.

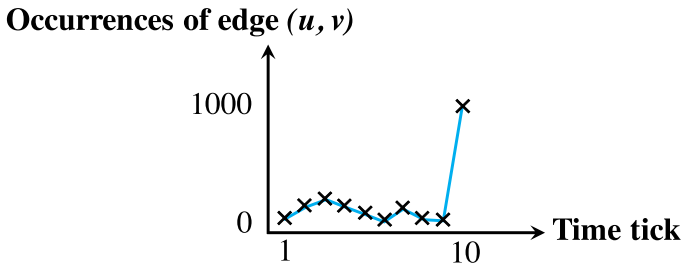

Fig. 1. Time series of a single source-destination pair (u, v), with a large burst of activity at time tick 10.

(4) Filtering Anomalies: We propose a variant, MIDAS-F, that introduces two modifications that aim to filter away anomalous edges to prevent them from negatively affecting the algorithm's internal data structures.

In the rest of this work, the related works in the area of anomaly detection will be briefly described in Section 2. The problem this work is trying to solve is given in Section 3. The base MIDAS algorithm, the relational variant MIDAS-R, and the theoretical guarantee are proposed or proved in Section 4. The filtering variant MIDAS-F and its underlying concepts are given in Section 5. The time and space complexity are analyzed in Section 6. Experiments and analysis on several real-world datasets are shown in Section 7.

Reproducibility: Our code and datasets are publicly available at https://github.com/Stream-AD/MIDAS.

2 RELATED WORK

Our work is closely related to areas like graph processing [18, 21, 40, 47], streaming algorithms [6, 24, 29, 33, 38], streaming graph analysis [12, 31, 34], and anomaly detection [10, 11, 28, 35, 39]. In this section, we limit our review only to previous approaches detecting anomalous signs on static and dynamic graphs. See [4] for an extensive survey on graph-based anomaly detection.

Anomaly detection in static graphs can be classified by which anomalous entities (nodes, edges, subgraph, etc.) are spotted.

- Anomalous node detection: ODDBALL [3] extracts egonet-based features and finds empirical patterns with respect to the features. Then, it identifies nodes whose egonets deviate from the patterns, including the count of triangles, total weight, and principal eigenvalues. CATCHSYNC [23] computes node features, including degree and authoritativeness [25], then spots nodes whose neighbors are notably close in the feature space.
- Anomalous subgraph detection: FRAUDAR [22] and k-cores [41] measure the anomalousness of nodes and edges, detecting a dense subgraph consisting of many anomalous nodes and edges.
- Anomalous edge detection: AutoPart [13] encodes an input graph based on similar connectivity among nodes, then spots edges whose removal reduces the total encoding cost significantly. NrMF [46] factorize the adjacency matrix and flag edges with high reconstruction error as outliers.

Anomaly detection in graph streams use as input a series of graph snapshots over time. We categorize them similarly according to the type of anomaly detected:

- Anomalous node detection: DTA/STA [45] approximates the adjacency matrix of the current snapshot based on incremental matrix factorization, then spots nodes corresponding to rows

Table 1. Comparison of Relevant Edge Stream Anomaly Detection Approaches

	SedanSpot [16]	PENminer [7]	F-FADE [14]	Midas
Microcluster Detection				✓
Guarantee on False Positive Probability				✓
Constant Memory	✓		✓	✓
Constant Update Time	✓	✓	✓	✓

with high reconstruction error. [2] dynamically partitions the network graph to construct a structural connectivity model and detect outliers in graph streams.

- Anomalous subgraph detection: Given a graph with timestamps on edges, CopyCatch [8] spots near-bipartite cores where each node is connected to others in the same core densly within a short time.
- Anomalous event detection: SpotLight [17] detects sudden appearance of many unexpected edges, and AnomRank [48] spots sudden changes in 1st and 2nd derivatives of PageRank.

Anomaly detection in edge streams use as input a stream of edges over time. Categorizing them according to the type of anomaly detected:

- Anomalous node detection: Given an edge stream, HotSpot [49] detects nodes whose egonets suddenly and significantly change.
- Anomalous subgraph detection: Given an edge stream, DenseAlert [42] identifies dense subtensors created within a short time.
- Anomalous edge detection: Only the methods in this category are applicable to our task, as they operate on edge streams and output a score per edge. RHSS [36] focuses on sparsely-connected parts of a graph but was evaluated in [16] and was outperformed by SedanSpot[16]. SedanSpot uses a customized PageRank to detect edge anomalies based on edge occurrence, preferential attachment, and mutual neighbors in sublinear space and constant time per edge. PENminer [7] explores the persistence of activity snippets, i.e., the length and regularity of edge-update sequences' reoccurrences. F-FADE [14] aims to detect anomalous interaction patterns by factorizing the frequency of those patterns. These methods can effectively detect anomalies, but they require a considerable amount of time.

  We compare with SedanSpot, PENminer, and F-FADE, however, as shown in Table 1, neither method aims to detect microclusters, or provides guarantees on false positive probability.

3 PROBLEM

Let $\mathcal{E} = \{e_1, e_2, \cdots\}$ be a stream of edges from a time-evolving graph $\mathcal{G}$. Without loss of generality, we assume the graph is a directed multigraph. Each arriving edge is a tuple $e_i = (u_i, v_i, t_i)$ consisting of a source node $u_i \in \mathcal{V}$, a destination node $v_i \in \mathcal{V}$, and a time of occurrence t_i, which is the time at which the edge was added to the graph. We do not assume the set of vertices $\mathcal{V}$ is known a priori. We do not assume the length of the edge stream $\mathcal{E}$ or the node set $\mathcal{V}$ is known as a priori. At any moment, the algorithm only knows its internal states and an incoming edge from the stream.

As the algorithm primarily detects microclusters, we first give the definition of a microcluster.

Definition 3.1. Given an edge e, a detection period $T \geq 1$, and a threshold $\beta > 1$. We say there is a microcluster if it satisfies

$$\frac{c(e,(n+1)T)}{c(e,nT)} > \beta \text{ or } \frac{c(e,(n+1)T)}{c(e,nT)} < \frac{1}{\beta}, \tag{1}$$

where $c(e,nT)$ is the occurrence count of e within period nT.

A typical value of T is 1, that is, the algorithm detects if there is a burst between adjacent timestamps. β is a user-defined parameter, it depends on the actual use cases and applications. A higher β means the system is less sensitive to minor fluctuations.

Next, we give the formal problem statement.

Problem 1. *Given an evolving edge stream $\mathcal{E}$, return an anomaly score for each incoming edge e according to the contextual information. A higher anomaly score indicates the edge is more suspicious to part of a microcluster, or a burst of edges.*

The desired properties of our algorithm are as follows:

- **Microcluster Detection:** It should detect suddenly appearing bursts of activity that share many repeated nodes or edges, which we refer to as microclusters.
- **Guarantees on False Positive Probability:** Given any user-specified probability level ϵ (e.g. 1%), the algorithm should be adjustable so as to provide a false positive probability of at most ϵ (e.g., by adjusting a threshold that depends on ϵ). Moreover, while guarantees on the false positive probability rely on assumptions about the data distribution, we aim to make our assumptions as weak as possible.
- **Constant Memory and Update Time:** For scalability in the streaming setting, the algorithm should run in constant memory and constant update time per newly arriving edge. Thus, its memory usage and update time should not grow with the length of the stream or the number of nodes in the graph.

4 MIDAS AND MIDAS-R ALGORITHMS

4.1 Overview

Next, we describe our Midas and Midas-R approaches. The following provides an overview:

(1) **Streaming Hypothesis Testing Approach:** We describe our Midas algorithm, which uses streaming data structures within a hypothesis testing-based framework, allowing us to obtain guarantees on false positive probability.
(2) **Detection and Guarantees:** We describe our decision procedure for determining whether a point is anomalous, and our guarantees on false positive probability.
(3) **Incorporating Relations:** We extend our approach to the Midas-R algorithm, which incorporates relationships between edges temporally and spatially.[1]

4.2 Midas: Streaming Hypothesis Testing Approach

4.2.1 Streaming Data Structures. In an offline setting, there are many time-series methods that could detect such bursts of activity. However, in an online setting, recall that we want memory usage to be bounded, so we cannot keep track of even a single such time series. Moreover, there are many such source-destination pairs, and the set of sources and destinations is not fixed a priori.

[1]We use "spatially" in a graph sense, i.e., connecting nearby nodes, not to refer to any other continuous spatial dimension.

To circumvent these problems, we maintain two types of CMS [15] data structures. Assume we are at a particular fixed time tick t in the stream; we treat time as a discrete variable for simplicity. Let s_{uv} be the total number of edges from u to v up to the current time. Then, we use a single CMS data structure to approximately maintain all such counts s_{uv} (for all edges uv) in constant memory: at any time, we can query the data structure to obtain an approximate count $\hat{s}_{uv}$.

Secondly, let a_{uv} be the number of edges from u to v in the current time tick (but not including past time ticks). We keep track of a_{uv} using a similar CMS data structure, the only difference being that we reset this CMS data structure every time we transition to the next time tick. Hence, this CMS data structure provides approximate counts $\hat{a}_{uv}$ for the number of edges from u to v in the current time tick t.

4.2.2 Hypothesis Testing Framework. Given approximate counts $\hat{s}_{uv}$ and $\hat{a}_{uv}$, how can we detect microclusters? Moreover, how can we do this in a principled framework that allows for theoretical guarantees?

Fix a particular source and destination pair of nodes, (u, v), as in Figure 1. One approach would be to assume that the time series in Figure 1 follows a particular generative model: for example, a Gaussian distribution. We could then find the mean and standard deviation of this Gaussian distribution. Then, at time t, we could compute the Gaussian likelihood of the number of edge occurrences in the current time tick, and declare an anomaly if this likelihood is below a specified threshold.

However, this requires a restrictive Gaussian assumption, which can lead to excessive false positives or negatives if the data follows a very different distribution. Instead, we use a weaker assumption: that the mean level (i.e. the average rate at which edges appear) in the current time tick (e.g. $t = 10$) is the same as the mean level before the current time tick ($t < 10$). Note that this avoids assuming any particular distribution for each time tick, and also avoids a strict assumption of stationarity over time.

Hence, we can divide the past edges into two classes: the current time tick ($t = 10$) and all past time ticks ($t < 10$). Recalling our previous notation, the number of events at ($t = 10$) is a_{uv}, while the number of edges in past time ticks ($t < 10$) is $s_{uv} - a_{uv}$.

Under the chi-squared goodness-of-fit test, the chi-squared statistic is defined as the sum over categories of $\frac{(\text{observed}-\text{expected})^2}{\text{expected}}$. In this case, our categories are $t = 10$ and $t < 10$. Under our mean level assumption, since we have s_{uv} total edges (for this source-destination pair), the expected number at $t = 10$ is $\frac{s_{uv}}{t}$, and the expected number for $t < 10$ is the remaining, i.e. $\frac{t-1}{t} s_{uv}$. Thus the chi-squared statistic is:

$$
\begin{aligned}
X^2 &= \frac{(\text{observed}_{(t=10)} - \text{expected}_{(t=10)})^2}{\text{expected}_{(t=10)}} \\
&\quad + \frac{(\text{observed}_{(t<10)} - \text{expected}_{(t<10)})^2}{\text{expected}_{(t<10)}} \\
&= \frac{(a_{uv} - \frac{s_{uv}}{t})^2}{\frac{s_{uv}}{t}} + \frac{((s_{uv} - a_{uv}) - \frac{t-1}{t} s_{uv})^2}{\frac{t-1}{t} s_{uv}} \\
&= \frac{(a_{uv} - \frac{s_{uv}}{t})^2}{\frac{s_{uv}}{t}} + \frac{(a_{uv} - \frac{s_{uv}}{t})^2}{\frac{t-1}{t} s_{uv}} \\
&= \left(a_{uv} - \frac{s_{uv}}{t}\right)^2 \frac{t^2}{s_{uv}(t-1)}
\end{aligned}
$$

Note that both a_{uv} and s_{uv} can be estimated by our CMS data structures, obtaining approximations $\hat{a}_{uv}$ and $\hat{s}_{uv}$ respectively. This leads to our following anomaly score, using which we can evaluate a newly arriving edge with source-destination pair (u, v):

Definition 4.1 (Anomaly Score). Given a newly arriving edge (u, v, t), our anomaly score is computed as:

$$\text{score}(u, v, t) = \left(\hat{a}_{uv} - \frac{\hat{s}_{uv}}{t}\right)^2 \frac{t^2}{\hat{s}_{uv}(t-1)}. \tag{2}$$

Algorithm 1 summarizes our MIDAS algorithm.

ALGORITHM 1: MIDAS: Streaming Anomaly Scoring

Input: Stream of graph edges over time
Output: Anomaly scores per edge
▷ **Initialize CMS data structures:**
Initialize CMS for total count s_{uv} and current count a_{uv}
while *new edge* $e = (u, v, t)$ *is received:* **do**
  ▷ **Update Counts:**
  Update CMS data structures for the new edge uv
  ▷ **Query Counts:**
  Retrieve updated counts $\hat{s}_{uv}$ and $\hat{a}_{uv}$
  ▷ **Anomaly Score:**
  output score$((u, v, t)) = (\hat{a}_{uv} - \frac{\hat{s}_{uv}}{t})^2 \frac{t^2}{\hat{s}_{uv}(t-1)}$

4.3 Detection and Guarantees

While Algorithm 1 computes an anomaly score for each edge, it does not provide a binary decision for whether an edge is anomalous or not. We want a decision procedure that provides binary decisions and a guarantee on the false positive probability: i.e. given a user-defined threshold ϵ, the probability of a false positive should be at most ϵ. Intuitively, the key idea is to combine the approximation guarantees of CMS data structures with properties of a chi-squared random variable.

The key property of CMS data structures we use is that given any ϵ and ν, for appropriately chosen CMS data structure sizes ($w = \lceil ln\frac{2}{\epsilon} \rceil, b = \lceil \frac{e}{\nu} \rceil$) [15], with probability at least $1 - \frac{\epsilon}{2}$, the estimates $\hat{a}_{uv}$ satisfy:

$$\hat{a}_{uv} \leq a_{uv} + \nu \cdot N_t, \tag{3}$$

where N_t is the total number of edges in the CMS for a_{uv} at time tick t. Since CMS data structures can only overestimate the true counts, we additionally have

$$s_{uv} \leq \hat{s}_{uv}. \tag{4}$$

Define an adjusted version of our earlier score:

$$\tilde{a}_{uv} = \hat{a}_{uv} - \nu N_t. \tag{5}$$

To obtain its probabilistic guarantee, our decision procedure computes $\tilde{a_{uv}}$, and uses it to compute an adjusted version of our earlier statistic:

$$\tilde{X^2} = \left(\tilde{a}_{uv} - \frac{\hat{s}_{uv}}{t}\right)^2 \frac{t^2}{\hat{s}_{uv}(t-1)}. \tag{6}$$

Note that the usage of X^2 and $\tilde{X^2}$ are different. X^2 is used as the score of individual edges while $\tilde{X^2}$ facilitates making binary decisions.

Then our main guarantee is as follows:

Theorem 1 (False Positive Probability Bound). *Let $\chi^2_{1-\epsilon/2}(1)$ be the $1-\epsilon/2$ quantile of a chi-squared random variable with 1 degree of freedom. Then:*

$$P(\tilde{X^2} > \chi^2_{1-\epsilon/2}(1)) < \epsilon. \tag{7}$$

In other words, using $\tilde{X^2}$ as our test statistic and threshold $\chi^2_{1-\epsilon/2}(1)$ results in a false positive probability of at most ϵ.

Proof. Recall that

$$X^2 = \left(a_{uv} - \frac{s_{uv}}{t}\right)^2 \frac{t^2}{s_{uv}(t-1)}, \tag{8}$$

was defined so that it has a chi-squared distribution. Thus:

$$P\left(X^2 \le \chi^2_{1-\epsilon/2}(1)\right) = 1-\epsilon/2. \tag{9}$$

At the same time, by the CMS guarantees we have:

$$P(\hat{a}_{uv} \le a_{uv} + v \cdot N_t) \ge 1-\epsilon/2. \tag{10}$$

By union bound, with probability at least $1-\epsilon$, both these events (9) and (10) hold, in which case:

$$\begin{aligned}
\tilde{X^2} &= \left(\tilde{a}_{uv} - \frac{\hat{s}_{uv}}{t}\right)^2 \frac{t^2}{\hat{s}_{uv}(t-1)} \\
&= \left(\hat{a}_{uv} - v \cdot N_t - \frac{\hat{s}_{uv}}{t}\right)^2 \frac{t^2}{\hat{s}_{uv}(t-1)} \\
&\le \left(a_{uv} - \frac{s_{uv}}{t}\right)^2 \frac{t^2}{s_{uv}(t-1)} \\
&= X^2 \le \chi^2_{1-\epsilon/2}(1).
\end{aligned}$$

Finally, we conclude that

$$P(\tilde{X^2} > \chi^2_{1-\epsilon/2}(1)) < \epsilon. \tag{11}$$

□

4.4 Incorporating Relations

In this section, we describe our Midas-R approach, which considers edges in a *relational* manner: that is, it aims to group together edges that are nearby, either temporally or spatially.

Temporal Relations: Rather than just counting edges in the same time tick (as we do in Midas), we want to allow for some temporal flexibility: i.e., edges in the recent past should also count toward the current time tick, but modified by a reduced weight. A simple and efficient way to do this using our CMS data structures is as follows: at the end of every time tick, rather than resetting our CMS data structures for a_{uv}, we scale all its counts by a fixed fraction $\alpha \in (0,1)$. This allows past edges to count toward the current time tick, with a diminishing weight. Note that we do not consider 0 or 1, because 0 clears all previous values when the time tick changes and hence does not include any temporal effect; and 1 does not scale the CMS data structures at all.

Spatial Relations: We would like to catch large groups of spatially nearby edges: e.g., a single source IP address suddenly creating a large number of edges to many destinations, or a small group

of nodes suddenly creating an abnormally large number of edges between them. A simple intuition we use is that in either of these two cases, we expect to observe *nodes* with a sudden appearance of a large number of edges. Hence, we can use CMS data structures to keep track of edge counts like before, except counting all edges adjacent to any node u. Specifically, we create CMS counters $\hat{a}_u$ and $\hat{s}_u$ to approximate the current and total edge counts adjacent to node u. Given each incoming edge (u, v), we can then compute three anomaly scores: one for edge (u, v), as in our previous algorithm; one for source node u, and one for destination node v. Finally, we combine the three scores by taking their maximum value. Another possibility of aggregating the three scores is to take their sum and we discuss the performance of summing the scores in Section 7. Algorithm 2 summarizes the resulting MIDAS-R algorithm.

ALGORITHM 2: MIDAS-R: Incorporating Relations

Input: Stream of graph edges over time
Output: Anomaly scores per edge

▷ **Initialize CMS data structures:**
Initialize CMS for total count s_{uv} and current count a_{uv}
Initialize CMS for total count s_u, s_v and current count a_u, a_v
while *new edge* $e = (u, v, t)$ *is received:* **do**
  ▷ **Update Counts:**
  Update CMS data structures for the new edge uv, source node u and destination node v
  ▷ **Query Counts:**
  Retrieve updated counts $\hat{s}_{uv}$ and $\hat{a}_{uv}$
  Retrieve updated counts $\hat{s}_u, \hat{s}_v, \hat{a}_u, \hat{a}_v$
  ▷ **Compute Edge Scores:**
  $\text{score}(u, v, t) = (\hat{a}_{uv} - \frac{\hat{s}_{uv}}{t})^2 \frac{t^2}{\hat{s}_{uv}(t-1)}$
  ▷ **Compute Node Scores:**
  $\text{score}(u, t) = (\hat{a}_u - \frac{\hat{s}_u}{t})^2 \frac{t^2}{\hat{s}_u(t-1)}$
  $\text{score}(v, t) = (\hat{a}_v - \frac{\hat{s}_v}{t})^2 \frac{t^2}{\hat{s}_v(t-1)}$
  ▷ **Final Scores:**
  output $\max\{\text{score}(u, v, t), \text{score}(u, t), \text{score}(v, t)\}$

5 MIDAS-F: FILTERING ANOMALIES

In MIDAS and MIDAS-R, in addition to being assigned an anomaly score, all normal and anomalous edges are also always recorded into the internal CMS data structures, regardless of their score. However, this inclusion of anomalous edges creates a "poisoning"effect which can allow future anomalies to slip through undetected.

Let us consider a simplified case of a denial of service attack where a large number of edges arrive between two nodes within a short period of time. MIDAS and MIDAS-R analysis can be divided into three stages.

In the first stage, when only a small number of such edges have been processed, the difference between the current count, $\hat{a}_{uv}$, and the expected count, $\frac{\hat{s}_{uv}}{t}$, is relatively small, so the anomaly score is low. This stage will not last long as the anomaly score will increase rapidly with the number of occurrences of anomalous edges.

In the second stage, once the difference between these two counters becomes significant, the algorithm will return a high anomaly score for those suspicious edges.

In the third stage, as the attack continues, i.e., anomalous edges continue to arrive, the expected count of the anomalous edge will increase. As a result, the anomaly score will gradually decrease, which can lead to false negatives, i.e., the anomalous edges being considered as normal edges, which is the "poisoning"effect due to the inclusion of anomalies in the CMS data structures.

Therefore, to prevent these false negatives, we introduce the improved filtering MIDAS (MIDAS-F) algorithm. The following provides an overview:

(1) **Refined Scoring Function:** The new formula of the anomaly score only considers the information of the current time tick and uses the mean value of the previous time ticks as the expectation.
(2) **Conditional Merge:** The current count a for the source, destination and edge are no longer merged into the total count s immediately. We determine whether they should be merged or not at the end of the time tick conditioned on the anomaly score.

5.1 Refined Scoring Function

During a time tick, while new edges continue to arrive, we only assign them a score, but do not directly incorporate them into our CMS data structures as soon as they arrive. This prevents anomalous edges from affecting the subsequent anomaly scores, which can possibly lead to false negatives. To solve this problem, we refine the scoring function to delay incorporating the edges to the end of the current time tick using a conditional merge as discussed in Section 5.2.

As defined before, let a_{uv} be the number of edges from u to v in the current time tick (but not including past time ticks). But unlike MIDAS and MIDAS-R, in MIDAS-F, we define s_{uv} to be the total number of edges from u to v up to the previous time tick, not including the current edge count a_{uv}. By not including the current edge count immediately, we prevent a high a_{uv} from being merged into s_{uv} so that the anomaly score for anomalous edges is not reduced.

In the MIDAS-F algorithm, we still follow the same assumption: that the mean level in the current time tick is the same as the mean level before the current time tick. However, instead of dividing the edges into two classes: past and current time ticks, we only consider the current time ticks. Similar to the chi-squared statistic of [9], our statistic is as below.

$$
\begin{aligned}
X^2 &= \frac{(\text{observed} - \text{expected})^2}{\text{expected}} \\
&= \frac{\left(a_{uv} - \dfrac{s_{uv}}{t-1}\right)^2}{\dfrac{s_{uv}}{t-1}} \\
&= \frac{\left[a_{uv}^2 - \dfrac{2a_{uv}s_{uv}}{t-1} + \left(\dfrac{s_{uv}}{t-1}\right)^2\right](t-1)}{s_{uv}} \\
&= \frac{a_{uv}^2(t-1)^2 - 2a_{uv}s_{uv}(t-1) + s_{uv}^2}{s_{uv}(t-1)} \\
&= \frac{(a_{uv} + s_{uv} - a_{uv}t)^2}{s_{uv}(t-1)}
\end{aligned}
$$

Both a_{uv} and s_{uv} can be estimated by our CMS data structures, obtaining approximations $\hat{a}_{uv}$ and $\hat{s}_{uv}$ respectively. We will use this new score as the anomaly score for our MIDAS-F algorithm.

Definition 5.1 (MIDAS-F Anomaly Score). Given a newly arriving edge (u, v, t), our anomaly score for this edge is computed as:

$$score(u, v, t) = \frac{(\hat{a}_{uv} + \hat{s}_{uv} - \hat{a}_{uv}t)^2}{\hat{s}_{uv}(t-1)}. \tag{12}$$

5.2 Conditional Merge

At the end of the current time tick, we decide whether to add a_{uv} to s_{uv} or not based on whether the edge (u, v) appears normal or anomalous.

We introduce c_{uv} to keep track of the anomaly score. Whenever the time tick changes, if c_{uv} is less than the pre-determined threshold θ, then the corresponding a_{uv} will be added to s_{uv}; otherwise, the expected count, i.e., $\frac{s_{uv}}{t-1}$ will be added to s_{uv} to keep the mean level unchanged. We add a_{uv} only when the cached score c_{uv} is less than the pre-determined threshold θ to prevent anomalous instances of a_{uv} from being added to the s_{uv}, which would reduce the anomaly score for an anomalous edge in the future time ticks.

To store the latest anomaly score c_{uv}, we use a CMS-like data structure resembling the CMS data structure for a and s used in MIDAS and MIDAS-R. The only difference is that the updates to this data structure do not increment the existing occurrence counts, but instead override the previous values. In the remaining part of the article, we refer to this CMS-like data structure as CMS for convenience.

To efficiently merge the CMS data structure for a into the CMS data structure for s, we need to know which buckets in the same hash functions across the multiple CMS data structures correspond to a particular edge. However, the algorithm does not store the original edges after processing. Therefore it is necessary that for each entity (edge, source, destination), the three CMS data structures for a, s, c use the same layout and the same hash functions for each hash table so that the corresponding buckets refer to the same edge and we can do a bucket-wise merge. In practice, the nine CMS data structures can be categorized into three groups, corresponding to the edges, source nodes, and destination nodes, respectively. Only the three CMS data structures within the same group need to share the same structure.

The conditional merge step is described in Algorithm 3.

ALGORITHM 3: MERGE

Input: CMS for s, a, c, threshold θ

for $\hat{s}$, $\hat{a}$, $\hat{c}$ *from CMS buckets* **do**
  if $\hat{c} < \theta$ **then**
    $\hat{s} = \hat{s} + \hat{a}$
  else if $t \neq 1$ **then**
    $\hat{s} = \hat{s} + \frac{\hat{s}}{t-1}$ // $\hat{s}$ is up-to-date until $t-1$

We also incorporate temporal and spatial relations as done in MIDAS-R. For temporal relations, at the end of every time tick, rather than resetting our CMS data structures for a_{uv}, we scale all its counts by a fixed fraction $\alpha \in (0, 1)$. This allows past edges to count toward the current time tick, with a diminishing weight. For spatial relations, we use CMS data structures to keep track of the anomaly score of each edge like before, except considering all edges adjacent to any node u. Specifically, we create CMS counters $\hat{c}_u$ to keep track of the anomaly score for each node u across all its neighbors. Given each incoming edge (u, v), we can then compute three anomaly scores: one for edge (u, v), as in MIDAS and MIDAS-R; one for source node u, and one for destination node v.

Algorithm 4 summarizes the resulting MIDAS-F algorithm. It can be divided into two parts: (1) regular edge processing in lines 13 to 24, where we compute anomaly scores for each incoming edge and update the relevant counts, and (2) scaling and merging steps in lines 6 to 12, where at the end of each time tick, we scale the current counts by α and merge them into the total counts.

ALGORITHM 4: MIDAS-F

Input: Stream of graph edges over time, threshold θ
Output: Anomaly scores per edge

1 ▷ **Initialize CMS data structures:**
2 Initialize CMS data structure for total count s_{uv}, current count a_{uv}, anomaly score c_{uv}
3 Initialize CMS data structure for total count s_u, current count a_u, anomaly score c_u
4 Initialize CMS data structure for total count s_v, current count a_v, anomaly score c_v
5 **while** *new edge* $e = (u, v, t)$ *is received* **do**
6 **if** $t \neq t_{internal}$ **then** // Time tick changes
7 ▷ **Merge Counts:**
8 MERGE($\hat{s}_{uv}$, $\hat{a}_{uv}$, $\hat{c}_{uv}$, θ)
9 MERGE($\hat{s}_u$, $\hat{a}_u$, $\hat{c}_u$, θ)
10 MERGE($\hat{s}_v$, $\hat{a}_v$, $\hat{c}_v$, θ)
11 Scale CMS data structures for a_{uv}, a_u, a_v by α
12 $t_{internal} = t$
13 ▷ **Update Counts:**
14 Update CMS data structure for a for new edge uv and nodes u, v
15 ▷ **Query Counts:**
16 Retrieve updated counts $\hat{s}_{uv}$ and $\hat{a}_{uv}$
17 Retrieve updated counts $\hat{s}_u, \hat{s}_v, \hat{a}_u, \hat{a}_v$
18 ▷ **Compute Scores:**
19 $c_{uv} = \dfrac{(\hat{a}_{uv} + \hat{s}_{uv} - \hat{a}_{uv}t)^2}{\hat{s}_{uv}(t-1)}$
20 $c_u = \dfrac{(\hat{a}_u + \hat{s}_u - \hat{a}_u t)^2}{\hat{s}_u(t-1)}$
21 $c_v = \dfrac{(\hat{a}_v + \hat{s}_v - \hat{a}_v t)^2}{\hat{s}_v(t-1)}$
22 Update CMS data structure for c for edge uv and nodes u, v
23 ▷ **Final Scores:**
24 **output** $\max\{c_{uv}, c_u, c_v\}$

6 TIME AND MEMORY COMPLEXITY

In terms of memory, MIDAS, MIDAS-R, and MIDAS-F only need to maintain the CMS data structures over time, which are proportional to $O(wb)$, where w and b are the number of hash functions and the number of buckets in the CMS data structures; which is bounded with respect to the data size.

For time complexity, the only relevant steps in Algorithms 1, 2, and 4 are those that either update or query the CMS data structures, which take $O(w)$ (all other operations run in constant time). Thus, time complexity per update step is $O(w)$.

For MIDAS-F, additionally, at the end of each time tick, a is merged into s, as shown in Algorithm 3. At the end of each time tick, the algorithm needs to iterate over all hash functions and buckets. Thus, time complexity per time tick is $O(wb)$.

7 EXPERIMENTS

In this section, we evaluate the performance of Midas, Midas-R, and Midas-F compared to SedanSpot on dynamic graphs. We aim to answer the following questions:

- **Q1. Accuracy:** How accurately does Midas detect real-world anomalies compared to baselines, as evaluated using the ground truth labels? How will hyperparameters affect the accuracy?
- **Q2. Scalability:** How does it scale with input stream length? How does the time needed to process each input compare to baseline approaches?
- **Q3. Real-World Effectiveness:** Does it detect meaningful anomalies in case studies on *Twitter* graphs?

Datasets: *DARPA* [30] is an intrusion detection dataset created in 1998. It has $25K$ nodes, $4.5M$ edges, and $46K$ timestamps. The dataset records IP-IP connections from June 1 to August 1. Due to the relatively sparse time density, we use minutes as timestamps. *CTU-13* [19] is a botnet traffic dataset captured in the CTU University in 2011. It consists of botnet samples from 13 different scenarios. We mainly focus on those with denial of service attacks, i.e., scenario 4, 10, and 11. The dataset includes $371K$ nodes, $2.5M$ edges, and $33K$ timestamps, where the resolution of timestamps is one second. *UNSW-NB15* [32] is a hybrid of real normal activities and synthetic attack behaviors. The dataset contains only 50 nodes, but has $2.5M$ records and $85K$ timestamps. Each timestamp in the dataset represents an interval of one second. *TwitterSecurity* [37] has $2.6M$ tweet samples for four months (May–Aug 2014) containing Department of Homeland Security keywords related to terrorism or domestic security. Entity-entity co-mention temporal graphs are built on a daily basis. Ground truth contains the dates of major world incidents. *TwitterWorldCup* [37] has $1.7M$ tweet samples for the World Cup 2014 season (June 12-July 13). The tweets are filtered by popular/official World Cup hashtags, such as #worldcup, #fifa, #brazil, etc. Entity-entity co-mention temporal graphs are constructed on one hour sample rate.

Note that we use different time tick resolutions for different datasets, demonstrating our algorithm is capable of processing datasets with various edge densities.

Baselines: As described in the Related Work, we use SedanSpot[16], PENminer [7], and F-FADE [14] as our baselines.

Evaluation Metrics: All the methods output an anomaly score per edge (higher is more anomalous). We report the ROC-AUC (higher is better).

7.1 Experimental Setup

All experiments are carried out on a $2.4GHz$ Intel Core $i9$ processor, $32GB$ RAM, running OS X 10.15.2. We implement our algorithm in C++ and use the open-source implementations of SedanSpot, PENminer, and F-FADE provided by the authors, following parameter settings as suggested in the original articles.

We use two hash functions for the CMS data structures, and set the number of CMS buckets to $1,024$ to result in an approximation error of $\nu = 0.003$. For Midas-R and Midas-F, we set the temporal decay factor α as 0.5. For Midas-F, the default threshold θ is $1,000$. We discuss the influence of α and the threshold θ in the following section. Unless otherwise specified, all experiments are repeated 21 times and the median performance (ROC-AUC, running time, etc.) is reported to minimize the influence of randomization in hashing. Also, note that the reported running time does not include I/O.

7.2 Accuracy

Table 2 shows the ROC-AUC of SedanSpot, PENminer, F-FADE, Midas, Midas-R, and Midas-F on the *DARPA*, *CTU-13*, and *UNSW-NB15* datasets since only these three datasets have ground

Table 2. ROC-AUC (Standard Deviation)

Dataset	PENminer	F-FADE	SedanSpot	Midas	Midas-R	Midas-F
DARPA	0.8267	0.8451	0.6442	0.9042 (0.0032)	0.9514 (0.0012)	**0.9873** (0.0009)
CTU-13	0.6041	0.8028	0.6397	0.9079 (0.0049)	0.9703 (0.0009)	**0.9843** (0.0004)
UNSW-NB15	0.7028	0.6858	0.7575	0.8843 (0.0079)	**0.8952** (0.0028)	0.8517 (0.0013)

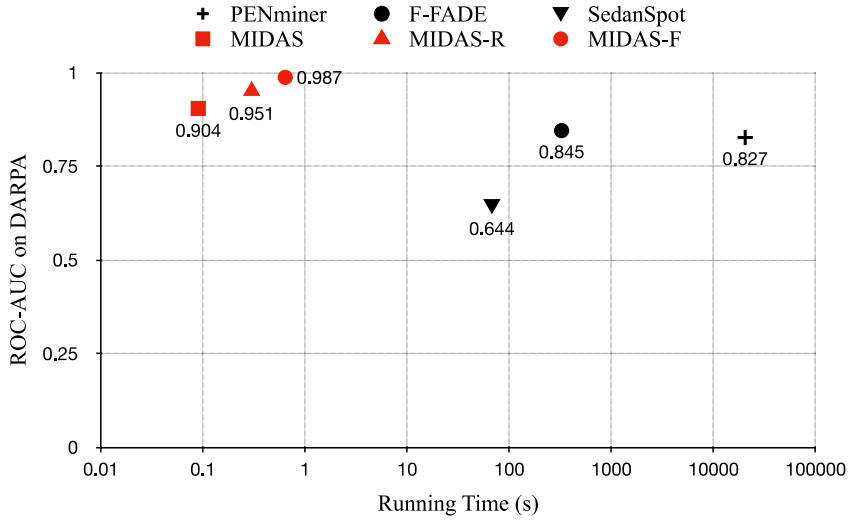

Fig. 2. ROC-AUC vs. time on *DARPA*.

truth available for each edge. On *DARPA*, compared to the baselines, Midas algorithms increase the ROC-AUC by 6%–53%, on *CTU-13* by 13%–62%, and on *UNSW-NB15* by 12%–30%.

Figures 2–4 plot the ROC-AUC vs. running time for the baselines and our methods on the *DARPA*, *CTU-13*, and *UNSW-NB15* datasets respectively. Note that Midas, Midas-R, and Midas-F achieve a much higher ROC-AUC compared to the baselines, while also running significantly faster.

Table 3 shows the influence of the temporal decay factor α on the ROC-AUC for Midas-R and Midas-F in the *DARPA* dataset. Note that instead of scaling the values in the CMS, Midas clears (or resets) values in the CMS data structure when the time tick changes; therefore, it is not included. We see that $\alpha = 0.9$ gives the maximum ROC-AUC for Midas-R (0.9657) and $\alpha = 0.8$ for Midas-F (0.9876).

Table 4 shows the influence of the threshold θ on the ROC-AUC for Midas-F in the *DARPA* dataset. If the threshold is too low, even normal edges can be rejected. On the other end, if the threshold is too high ($\theta = 10^7$), very few anomalous edges will be rejected, and Midas-F (ROC-AUC = 0.9572) performs similar to Midas-R (ROC-AUC = 0.95). We see that $\theta = 10^3$ achieves the maximum ROC-AUC of 0.9853.

Table 5 shows the ROC-AUC vs. number of buckets (b) in CMSs on the *UNSW-NB15* dataset. We can observe the increase in the performance, which indicates that increasing the buckets helps alleviate the effect of conflicts, and further reduce the false positive rate of the resulting scores. Also, note that the ROC-AUC does not change after 10,000 buckets, one possible reason is that the number of columns is sufficiently high to negate the influence of conflicts. This also simulates the "no-CMS" situation, i.e., the edge counts are maintained in an array of infinite size.

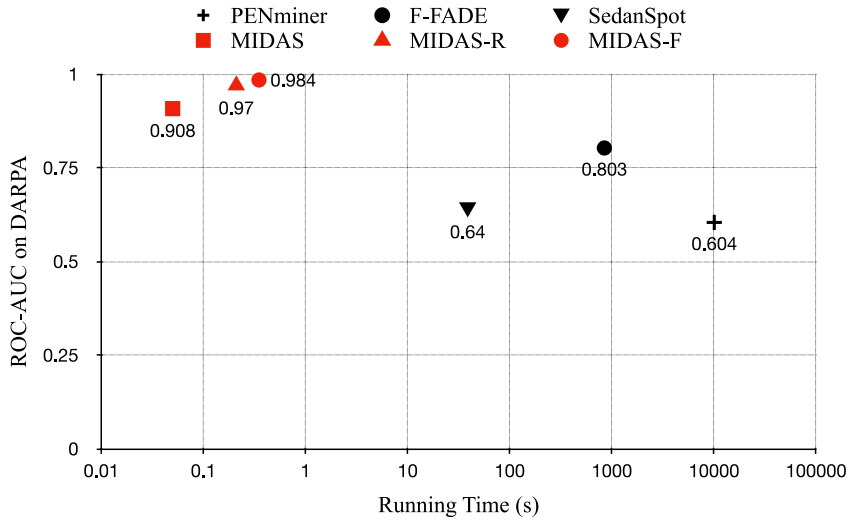

Fig. 3. ROC-AUC vs. time on *CTU-13*.

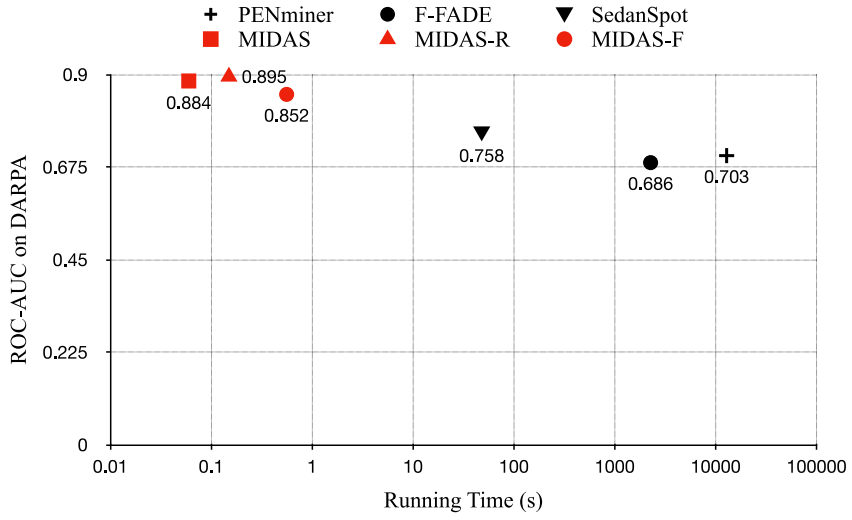

Fig. 4. ROC-AUC vs. time on *UNSW-NB15*.

For MIDAS-R and MIDAS-F, we also test the effect of summing the three anomaly scores, one for the edge (u, v), one for node u, and one for node v. The scores are not significantly different: with default parameters, the ROC-AUC is 0.95 for MIDAS-R (vs. 0.95 using maximum) and 0.98 for MIDAS-F (vs. 0.99 using maximum).

7.3 Scalability

Table 6 shows the running time for the baselines and MIDAS algorithms. Compared to SEDANSPOT, on all the five datasets, MIDAS speeds up by 623 − 800×, MIDAS-R speeds up by 183 − 326×, and MIDAS-F speeds up by 85 − 286×. Compared to F-FADE, on all the five datasets, MIDAS speeds up

Table 3. Influence of Temporal Decay Factor α on the ROC-AUC in MIDAS-R and MIDAS-F

α	MIDAS-R	MIDAS-F
0.1	0.9346	0.9779
0.2	0.9429	0.9801
0.3	0.9449	0.9817
0.4	0.9484	0.9837
0.5	0.9504	0.9852
0.6	0.9526	0.9863
0.7	0.9542	0.9863
0.8	0.9590	0.9883
0.9	0.9657	0.9876

Table 4. Influence of Threshold θ on the ROC-AUC in MIDAS-F

θ	ROC-AUC
10^0	0.9838
10^1	0.9840
10^2	0.9839
10^3	0.9853
10^4	0.9807
10^5	0.9625
10^6	0.9597
10^7	0.9572

Table 5. Influence of the Number of Buckets on the ROC-AUC in MIDAS, MIDAS-R, and MIDAS-F

b	MIDAS	MIDAS-R	MIDAS-F
10^2	0.7978	0.8161	0.8653
10^3	0.8732	0.8418	0.8863
10^4	0.8842	0.8517	0.8952
10^5	0.8842	0.8517	0.8952
10^6	0.8842	0.8517	0.8952
10^7	0.8842	0.8517	0.8952

by 806 – 37782×, MIDAS-R speeds up by 366 – 15112×, and MIDAS-F speeds up by 366 – 4047×. Compared to PENminer, on all the five datasets, MIDAS speeds up by 101419 – 214282×, and MIDAS-R speeds up by 46099 – 85712×, MIDAS-F speeds up by 22958 – 47324×.

SEDANSPOT requires several subprocesses (hashing, random-walking, reordering, sampling, etc.), resulting in a large computation time. For PENminer and F-FADE, while the python implementation is a factor, the algorithm procedures also negatively affect their running speed.

Table 6. Running Time for Different Datasets in Seconds

Dataset	PENminer	F-FADE	SedanSpot	Midas	Midas-R	Midas-F
DARPA	20,423 s	325.1 s	67.54 s	0.09 s	0.30 s	0.64 s
CTU-13	10,065 s	844.2 s	38.73 s	0.05 s	0.21 s	0.35 s
UNSW-NB15	12,857 s	2,267 s	48.03 s	0.06 s	0.15 s	0.56 s
TwitterWorldCup	3,786 s	141.7 s	22.92 s	0.03 s	0.07 s	0.08 s
TwitterSecurity	5,071 s	40.34 s	31.18 s	0.05 s	0.11 s	0.11 s

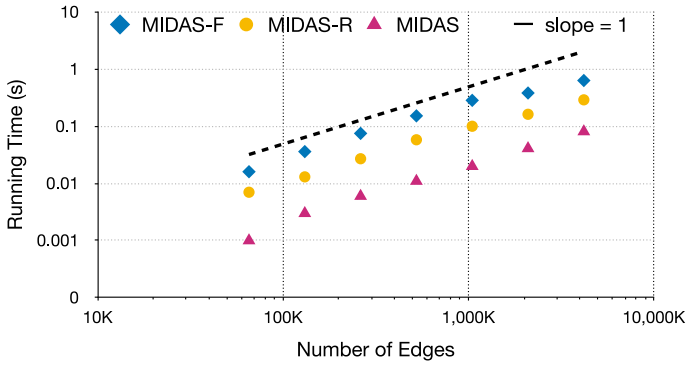

Fig. 5. Midas, Midas-R, and Midas-F scale linearly with the number of edges in the input dynamic graph.

PENminer requires active pattern exploration and F-FADE needs expensive factorization operations. For Midas, the improvement of running speed is through both, the algorithm procedure as well as the implementation. The algorithm procedure is less complicated than baselines; for each edge, the only operations are updating CMSs (hashing) and computing scores, and both are within constant time complexity. The implementation is well optimized and utilizes techniques like auto-vectorization to boost execution efficiency.

Figure 5 shows the scalability of Midas, Midas-R, and Midas-F algorithms. We plot the time required to process the first $2^{16}, 2^{17}, \ldots, 2^{22}$ edges of the *DARPA* dataset. This confirms the linear scalability of Midas algorithms with respect to the number of edges in the input dynamic graph due to its constant processing time per edge. Note that Midas, Midas-R and Midas-F can process $4.5M$ edges within 1 second, allowing real-time anomaly detection.

Figure 6 plots the number of edges and the time to process each edge in the *DARPA* dataset. Due to the limitation of clock accuracy, it is difficult to obtain the exact time of each edge. But we can approximately divide them into two categories, i.e., less than $1\mu s$ and greater than $1\mu s$. All three methods process majority of the edges within $1\mu s$.

Figure 7 shows the dependence of the running time on the threshold for Midas-F. We observe that the general pattern is a line with slope close to 0. Therefore, the time complexity does not depend on the threshold.

Figure 8 shows the dependence of the running time on the number of hash functions and linear scalability.

Figure 9 shows the dependence of the running time on the number of buckets. In general, the time increases with the number of buckets, but Midas-F is more sensitive to the number of buckets. This is because Midas-F requires updating the CMS data structure, which, due to the nested

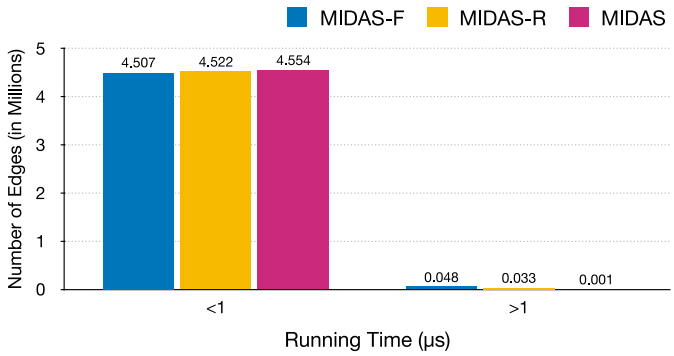

Fig. 6. Distribution of processing times for $\sim 4.5M$ edges of *DARPA* dataset.

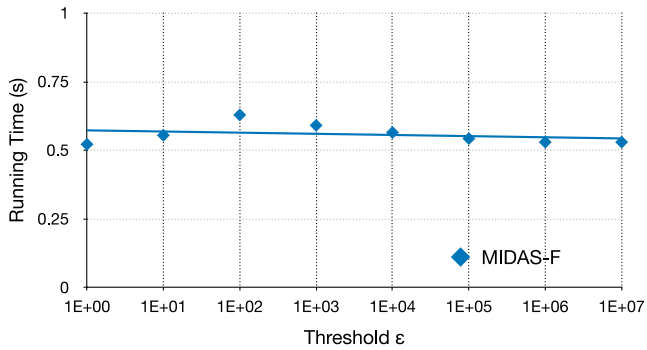

Fig. 7. Running time of MIDAS-F does not depend on the threshold θ.

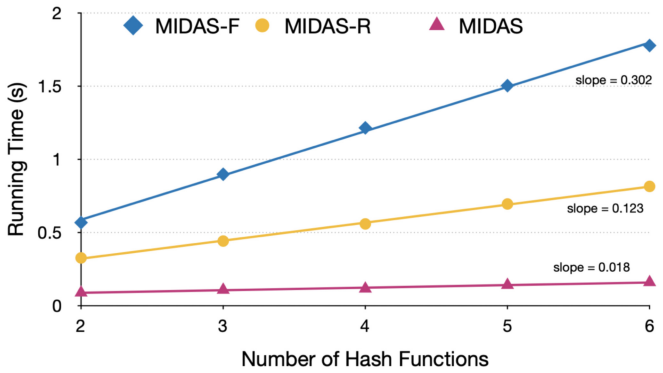

Fig. 8. MIDAS, MIDAS-R, and MIDAS-F scale linearly with the number of hash functions.

selection operation, cannot be vectorized. On the other hand, in MIDAS and MIDAS-R, the clearing and α reducing operations can be efficiently vectorized.

7.4 Real-World Effectiveness

We measure anomaly scores using MIDAS, MIDAS-R, MIDAS-F, SEDANSPOT, PENminer, and F-FADE on the *TwitterSecurity* dataset. Figure 10 plots the normalized anomaly scores vs. day (during the four months of 2014). We aggregate edges for each day by taking the highest anomaly

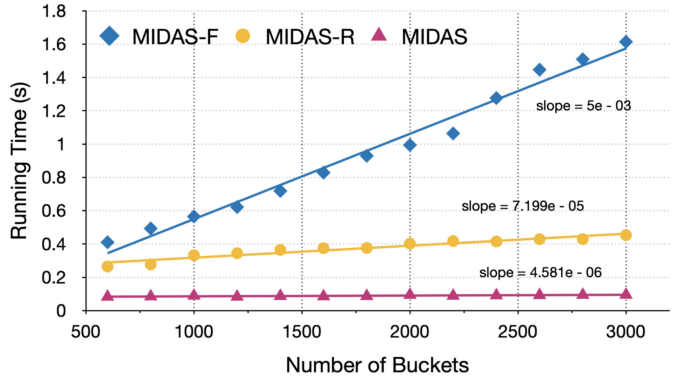

Fig. 9. MIDAS, MIDAS-R, and MIDAS-F scale linearly with the number of buckets.

score. Anomalies correspond to major world news such as the Mpeketoni attack (event 6) or the Soma Mine explosion (event 1).

SEDANSPOT gives relatively high scores for all days making it difficult to spot anomalies (events). F-FADE produces the highest score near event 6 and peaks at events 2 and 8. However, for other days, scores are maintained around a static level, which provides no useful information in detecting rest events. Also note that as F-FADE requires initial learning, thus there are no scores around event 1. PENminer's scores keep fluctuating during the four months. It would be hard to learn anomalies from the produced scores. For MIDAS and its variants, we can see four apparent peaks near major events like 2, 6, 7, and 8, and at events 1 and 10, small peaks are also noticeable, though less obvious. Hence, we can see our proposed algorithm can extract out more anomalous events from real-world social networks compared with baselines.

The anomalies detected by MIDAS, MIDAS-R, and MIDAS-F coincide with the ground events in the *TwitterSecurity* timeline as follows:

(1) 13-05-2014. Turkey Mine Accident, Hundreds Dead.
(2) 24-05-2014. Raid.
(3) 30-05-2014. Attack/Ambush.
    03-06-2014. Suicide bombing.
(4) 09-06-2014. Suicide/Truck bombings.
(5) 10-06-2014. Iraqi Militants Seized Large Regions.
    11-06-2014. Kidnapping.
(6) 15-06-2014. Attack.
(7) 26-06-2014. Suicide Bombing/Shootout/Raid.
(8) 03-07-2014. Israel Conflicts with Hamas in Gaza.
(9) 18-07-2014. Airplane with 298 Onboard was Shot Down over Ukraine.
(10) 30-07-2014. Ebola Virus Outbreak.

Microcluster anomalies: Figure 11 corresponds to event 7 in the *TwitterSecurity* dataset. Single edges in the plot denote 444 actual edges, while double edges in the plot denote 888 actual edges between the nodes. This suddenly arriving (within 1 day) group of suspiciously similar edges is an example of a microcluster anomaly which MIDAS, MIDAS-R, and MIDAS-F detect, but SEDANSPOT misses.

8 CONCLUSION

In this article, we proposed MIDAS, MIDAS-R, and MIDAS-F for microcluster based detection of anomalies in edge streams. Future work could consider more general types of data, including heterogeneous graphs or tensors. Our contributions are as follows:

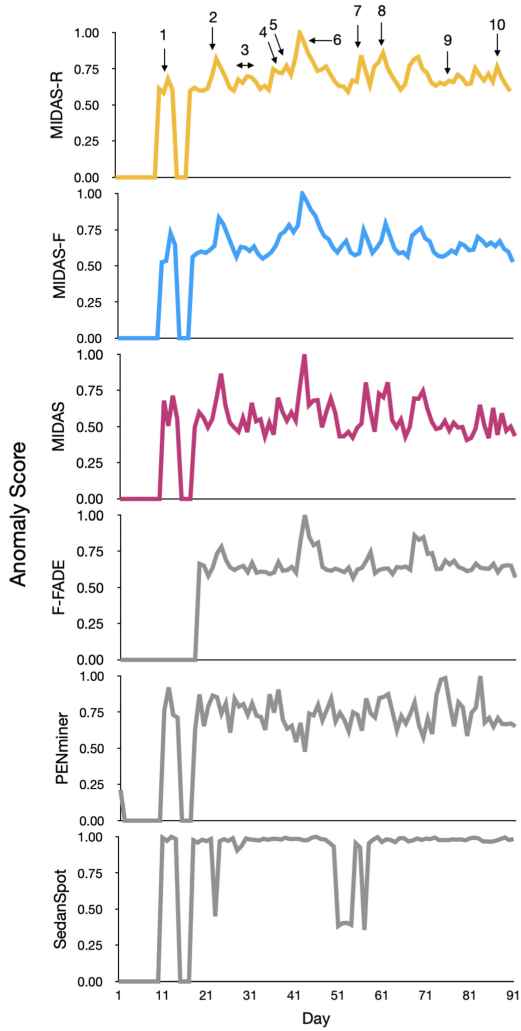

Fig. 10. Anomalies detected by MIDAS, MIDAS-R, and MIDAS-F correspond to major security-related events in *TwitterSecurity*.

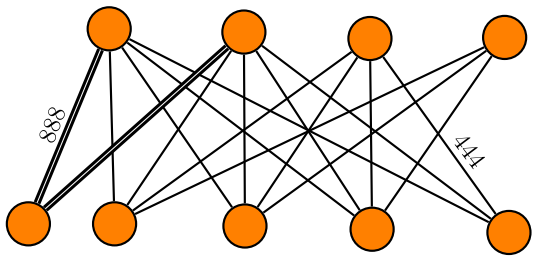

Fig. 11. Microcluster anomaly in *TwitterSecurity*.

(1) Streaming Microcluster Detection: We propose a novel streaming approach combining statistical (chi-squared test) and algorithmic (count-min sketch) ideas to detect microcluster anomalies, requiring constant time and memory.
(2) Theoretical Guarantees: In Theorem 1, we show guarantees on the false positive probability of Midas.
(3) Effectiveness: Our experimental results show that Midas outperforms baseline approaches by up to 62% higher ROC-AUC, and processes the data orders-of-magnitude faster than baseline approaches.
(4) Filtering Anomalies: We propose a variant, Midas-F, that introduces two modifications that aim to filter away anomalous edges to prevent them from negatively affecting the algorithm's internal data structures.

REFERENCES

[1] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. 2010. On clustering graph streams. In *SDM*.
[2] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. 2011. Outlier detection in graph streams. In *ICDE*.
[3] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*.
[4] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.
[5] Mohamed Jaward Bah, Hongzhi Wang, Mohamed Hammad, Furkh Zeshan, and Hanan Aljuaid. 2019. An effective minimal probing approach with micro-cluster for distance-based outlier detection in data streams. *IEEE Access* 7 (2019), 154922–154934.
[6] Maroua Bahri, Silviu Maniu, and Albert Bifet. 2018. A sketch-based naive Bayes algorithms for evolving data streams. In *IEEE Big Data*.
[7] Caleb Belth, Xinyi Zheng, and Danai Koutra. 2020. Mining persistent activity in continually evolving networks. In *KDD*.
[8] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: Stopping group attacks by spotting lockstep behavior in social networks. In *WWW*.
[9] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. MIDAS: Microcluster-based detector of anomalies in edge streams. In *AAAI*.
[10] Elnaz Bigdeli, Mahdi Mohammadi, Bijan Raahemi, and Stan Matwin. 2018. Incremental anomaly detection using two-layer cluster-based structure. *Information Sciences* 429 (2018), 315–331.
[11] Petko Bogdanov, Christos Faloutsos, Misael Mongiovì, Evangelos E. Papalexakis, Razvan Ranca, and Ambuj K. Singh. 2013. NetSpot: Spotting significant anomalous regions on dynamic networks. In *SDM*.
[12] Paul Boniol and Themis Palpanas. 2020. Series2graph: Graph-based subsequence anomaly detection for time series. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1821–1834.
[13] Deepayan Chakrabarti. 2004. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*.
[14] Yen-Yu Chang, Pan Li, Rok Sosic, M. H. Afifi, Marco Schweighauser, and Jure Leskovec. 2021. F-FADE: Frequency factorization for anomaly detection in edge streams. In *WSDM*.
[15] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.
[16] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *ICDM*.
[17] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. SpotLight: Detecting anomalies in streaming graphs. In *KDD*.
[18] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.
[19] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *Computers & Security* 45 (2014), 100–123.
[20] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. 2012. Integrating community matching and outlier detection for mining evolutionary community outliers. In *KDD*.
[21] Liang He, Bin Shao, Yatao Li, and Enhong Chen. 2015. Distributed real-time knowledge graph serving. In *BIGCOMP*.
[22] Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. 2017. Graph-based fraud detection in the face of camouflage. *ACM Transactions on Knowledge Discovery from Data* 11, 4 (2017), 1–26.
[23] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2016. Catching synchronized behaviors in large networks: A graph mining approach. *ACM Transactions on Knowledge Discovery from Data* 10, 4 (2016), 1–27.

[24] Arijit Khan and Sixing Yan. 2018. Composite hashing for data stream sketches. *ArXiv* abs/1808.06800 (2018).
[25] Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46, 5 (1999), 604–632.
[26] Danai Koutra, Joshua T. Vogelstein, and Christos Faloutsos. 2013. Deltacon: A principled massive-graph similarity function. In *SDM*.
[27] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. 2011. The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems* 29, 2 (2011), 249–272.
[28] Adarsh Kulkarni, Priya Mani, and Carlotta Domeniconi. 2017. Network-based anomaly detection for insider trading. *ArXiv* abs/1702.05809 (2017).
[29] Panagiotis Liakos, Katia Papakonstantinopoulou, Alexandros Ntoulas, and Alex Delis. 2020. Rapid detection of local communities in graph streams. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1.
[30] Richard Lippmann, Robert K. Cunningham, David J. Fried, Isaac Graf, Kris R. Kendall, Seth E. Webster, and Marc A. Zissman. 1999. Results of the DARPA 1998 offline intrusion detection evaluation. In *Recent Advances in Intrusion Detection*.
[31] Wenjuan Luo, Han Zhang, Xiaodi Yang, Lin Bo, Xiaoqing Yang, Zang Li, Xiaohu Qie, and Jieping Ye. 2020. Dynamic heterogeneous graph neural network for real-time event prediction. In *KDD*.
[32] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *MilCIS*.
[33] Xin Mu, Feida Zhu, Juan Du, Ee-Peng Lim, and Zhi-Hua Zhou. 2017. Streaming classification with emerging new class by class matrix sketching. In *AAAI*.
[34] Takaaki Nakamura, Makoto Imamura, Ryan Mercer, and Eamonn Keogh. 2020. MERLIN: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In *ICDM*.
[35] Caleb C. Noble and Diane J. Cook. 2003. Graph-based anomaly detection. In *KDD*.
[36] 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 *SDM*.
[37] Shebuti Rayana and Leman Akoglu. 2016. Less is more: Building selective anomaly ensembles. *ACM Transactions on Knowledge Discovery from Data* 10, 4 (2016), 1–33.
[38] Florin Rusu and Alin Dobra. 2009. Sketching sampled data streams. In *ICDE*.
[39] Mandana Saebi, Jian Xu, Lance M. Kaplan, Bruno Ribeiro, and Nitesh V. Chawla. 2020. Efficient modeling of higher-order dependencies in networks: from algorithm to application for anomaly detection. *EPJ Data Science* 9, 1 (2020), 15.
[40] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2019. Finding lasting dense subgraphs. *Data Mining and Knowledge Discovery* 33 (2019), 1417–1445.
[41] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems* 54, 3 (2018), 677–710.
[42] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. DenseAlert: Incremental dense-subtensor detection in tensor streams. In *KDD*.
[43] Kumar Sricharan and Kamalika Das. 2014. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*.
[44] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. 2007. GraphScope: Parameter-free mining of large time-evolving graphs. In *KDD*.
[45] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: Dynamic tensor analysis. In *KDD*.
[46] Hanghang Tong and Ching-Yung Lin. 2011. Non-negative residual matrix factorization with application to graph anomaly detection. In *SDM*.
[47] Da Yan, Guimu Guo, Md Mashiur Rahman Chowdhury, M. Tamer Özsu, Wei-Shinn Ku, and John C. S. Lui. 2020. G-thinker: A distributed framework for mining subgraphs in a big graph. In *ICDE*.
[48] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *KDD*.
[49] Weiren Yu, Charu C. Aggarwal, Shuai Ma, and Haixun Wang. 2013. On anomalous hotspot discovery in graph streams. In *ICDM*.

Received June 2021; revised September 2021; accepted October 2021