The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

MIDAS: Microcluster-Based Detector of Anomalies in Edge Streams

Siddharth Bhatia,[1] **Bryan Hooi,**[1] **Minji Yoon,**[2] **Kijung Shin,**[3] **Christos Faloutsos**[2]
[1]National University of Singapore, [2]Carnegie Mellon University, [3]KAIST
{siddharth, bhooi}@comp.nus.edu.sg, {minjiy, christos}@cs.cmu.edu, kijungs@kaist.ac.kr

Abstract

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. MIDAS has the following properties: (a) it detects microcluster anomalies while providing theoretical guarantees about its false positive probability; (b) it is online, thus processing each edge in constant time and constant memory, and also processes the data $108 - 505$ times faster than state-of-the-art approaches; (c) it provides 46%-52% higher accuracy (in terms of AUC) than state-of-the-art approaches.

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 well-researched problem with majority of the proposed approaches (Akoglu, McGlohon, and Faloutsos 2010; Chakrabarti 2004; Hooi et al. 2017; Jiang et al. 2016; Kleinberg 1999; Shin, Eliassi-Rad, and Faloutsos 2018) 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 (Eswaran et al. 2018; Sun, Tao, and Faloutsos 2006; Sun et al. 2007; Koutra, Vogelstein, and Faloutsos 2013; Sricharan and Das 2014; Gupta et al. 2012). 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 which uses constant memory in graph size.

Moreover, fraudulent or anomalous events in many applications occur in microclusters or suddenly arriving groups of suspiciously similar edges e.g. denial of service attacks in network traffic data and lockstep behavior. However, existing methods which process edge streams in an online manner, including (Eswaran and Faloutsos 2018; Ranshous et al. 2016), aim to detect individually surprising edges, not microclusters, and can thus miss large amounts of suspicious activity.

In this work, we propose MIDAS, which detects *microcluster anomalies*, or suddenly arriving groups of suspiciously similar edges, in edge streams, using constant time and memory. In addition, by using a principled hypothesis testing framework, MIDAS provides theoretical bounds on the false positive probability, which these methods do not provide.

Our main contributions are as follows:

1. Streaming Microcluster Detection: We propose a novel streaming approach for detecting 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 46%-52% accuracy (in terms of AUC), and processes the data $108-505$ times faster than baseline approaches.

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

Related Work

In this section, we review previous approaches to detect anomalous signs on static and dynamic graphs. See (Akoglu, Tong, and Koutra 2015) 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: (Akoglu, McGlohon, and Faloutsos 2010) 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. (Jiang et al. 2016) computes node features, including degree and authoritativeness (Kleinberg 1999), then spots nodes whose neighbors are notably close in the feature space.

- Anomalous subgraph detection: (Hooi et al. 2017) and (Shin, Eliassi-Rad, and Faloutsos 2018) measure the anomalousness of nodes and edges, detecting a dense subgraph consisting of many anomalous nodes and edges.
- Anomalous edge detection: (Chakrabarti 2004) encodes an input graph based on similar connectivity among nodes, then spots edges whose removal reduces the total encoding cost significantly. (Tong and Lin 2011) 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: (Sun, Tao, and Faloutsos 2006) approximates the adjacency matrix of the current snapshot based on incremental matrix factorization, then spots nodes corresponding to rows with high reconstruction error.
- Anomalous subgraph detection: Given a graph with timestamps on edges, (Beutel et al. 2013) spots near-bipartite cores where each node is connected to others in the same core densly within a short time. (Jiang et al. 2016) detects groups of nodes who form dense subgraphs in a temporally synchronized manner.
- Anomalous event detection: (Eswaran et al. 2018) detects sudden appearance of many unexpected edges, and (Yoon et al. 2019) 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, (Yu et al. 2013) detects nodes whose egonets suddenly and significantly change.
- Anomalous subgraph detection: Given an edge stream, (Shin et al. 2017) identifies dense subtensors created within a short time.
- Anomalous edge detection: (Ranshous et al. 2016) focuses on sparsely-connected parts of a graph, while (Eswaran and Faloutsos 2018) identifies edge anomalies based on edge occurrence, preferential attachment, and mutual neighbors.

Only the 2 methods in the last category are applicable to our task, as they operate on edge streams and output a score per edge. However, as shown in Table 1, neither method aims to detect microclusters, or provides guarantees on false positive probability.

Table 1: Comparison of relevant edge stream anomaly detection approaches.

	SedanSpot (2018)	RHSS (2016)	**Midas**
Microcluster Detection			✔
Guarantee on False Positive Probability			✔
Constant Memory	✓	✓	✔
Constant Update Time	✓	✓	✔

Problem

Let $\mathcal{E} = \{e_1, e_2, \cdots\}$ be a stream of edges from a time-evolving graph $\mathcal{G}$. 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. For example, in a network traffic stream, an edge e_i could represent a connection made from a source IP address u_i to a destination IP address v_i at time t_i. We do not assume that the set of vertices $\mathcal{V}$ is known a priori: for example, new IP addresses or user IDs may be created over the course of the stream.

We model $\mathcal{G}$ as a directed graph. Undirected graphs can simply be handled by treating an incoming undirected $e_i = (u_i, v_i, t_i)$ as two simultaneous directed edges, one in either direction.

We also allow $\mathcal{G}$ to be a multigraph: edges can be created multiple times between the same pair of nodes. Edges are allowed to arrive simultaneously: i.e. $t_{i+1} \geq t_i$, since in many applications t_i are given in the form of discrete time ticks.

The desired properties of our algorithm are as follows:

- **Microcluster Detection:** It should detect suddenly appearing bursts of activity which 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 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.

Proposed Algorithm

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].

MIDAS: Streaming Hypothesis Testing Approach

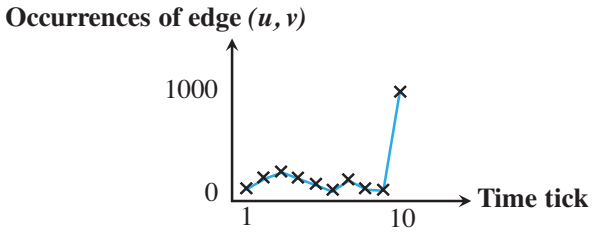

Figure 1: Time series of a single source-destination pair (u, v), with a large burst of activity at time tick 10.

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 micro-cluster, as it consists of a large group of edges which are very similar to one another (in fact identical), both **spatially** (i.e. in terms of the nodes they connect) and **temporally**.

Streaming Data Structures In an offline setting, there are many time-series methods which 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.

To circumvent these problems, we maintain two types of Count-Min-Sketch (CMS) (Cormode and Muthukrishnan 2005) 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 $\hat{s_{uv}}$.

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

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.

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:

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

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

$$score((u,v,t)) = (\hat{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)} \quad (1)$$

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$((u,v,t)) = (\hat{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)}$

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, with probability at least $1 - \frac{\epsilon}{2}$, the estimates $\hat{a_{uv}}$ satisfy:

$$\hat{a_{uv}} \leq a_{uv} + \nu \cdot N_t \quad (2)$$

where N_t is the total number of edges at time t. Since CMS data structures can only overestimate the true counts, we additionally have

$$s_{uv} \leq \hat{s_{uv}} \quad (3)$$

Define an adjusted version of our earlier score:

$$\tilde{a_{uv}} = \hat{a_{uv}} - \nu N_t \quad (4)$$

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

$$\tilde{X^2} = (\tilde{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)} \quad (5)$$

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:*

$$P(\tilde{X^2} > \chi^2_{1-\epsilon/2}(1)) < \epsilon \quad (6)$$

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

$$X^2 = (a_{uv} - \frac{s_{uv}}{t})^2 \frac{t^2}{s_{uv}(t-1)} \quad (7)$$

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

$$P(X^2 \leq \chi^2_{1-\epsilon/2}(1)) \geq 1 - \epsilon/2 \quad (8)$$

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

$$P(\hat{a_{uv}} \leq a_{uv} + \nu \cdot N_t) \leq 1 - \epsilon/2 \quad (9)$$

By union bound, with probability at least $1 - \epsilon$, both these events (8) and (9) hold, in which case:

$$\begin{aligned}
\tilde{X^2} &= (\tilde{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)} \\
&= (\hat{a_{uv}} - \nu \cdot N_t - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)} \\
&\leq (a_{uv} - \frac{s_{uv}}{t})^2 \frac{t^2}{s_{uv}(t-1)} \\
&= X^2 \leq \chi^2_{1-\epsilon/2}(1)
\end{aligned}$$

Finally, we conclude that

$$P(\tilde{X^2} > \chi^2_{1-\epsilon/2}(1)) < \epsilon. \quad (10)$$

□

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 which 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 a_{uv}, we reduce 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.

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 anomalousness scores: one for edge (u, v), as in our previous algorithm; one for node u, and one for 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. 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 and current count a_u
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 $\hat{a_{uv}}$
  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 Node Scores:**
  output $\max\{\text{score}(u, v, t), \text{score}(u, t), \text{score}(v, t)\}$

Time and Memory Complexity

In terms of memory, both MIDAS and MIDAS-R 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 Algorithm 1 and 2 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 $O(w)$.

Experiments

In this section, we evaluate the performance of MIDAS and MIDAS-R 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?

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* (Lippmann et al. 1999) has $4.5M$ IP-IP communications between $9.4K$ source IP and $2.3K$ destination IP over $87.7K$ minutes. Each communication is a directed edge (srcIP, dstIP, timestamp, attack) where the ground truth attack label indicates whether the communication is an attack or not (anomalies are 23.8% of total).

TwitterSecurity (Rayana and Akoglu 2015; 2016) 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 daily basis (80 time ticks).

TwitterWorldCup (Rayana and Akoglu 2015; 2016) 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. Similar to TwitterSecurity, entity-entity co-mention temporal graphs are constructed on 5 minute sample rate (8640 time points).

Baseline: As described in our Related Work, only RHSS and SEDANSPOT operate on edge streams and provide a score for each edge. SEDANSPOT uses personalised PageRank to detect anomalies in sublinear space and constant time per edge. However, RHSS was evaluated in (Eswaran and Faloutsos 2018) on the DARPA dataset and found to have AUC of 0.17 (lower than chance). Hence, we only compare with SEDANSPOT.

Evaluation Metrics: All the methods output an anomaly score per edge (higher is more anomalous). We calculate the True Positive Rate (TPR) and False Positive Rate (FPR) and plot the ROC curve (TPR vs FPR). We also report the Area under the ROC curve (AUC) and Average Precision Score.

Experimental Setup

All experiments are carried out on a $2.7GHz$ Intel Core $i5$ processor, $16GB$ RAM, running OS X $10.14.6$. We implement MIDAS and MIDAS-R in C++. We use 2 hash functions for the CMS data structures, and we set the number of CMS buckets to 2719 to result in an approximation error of $\nu = 0.001$. For MIDAS-R, we set the temporal decay factor α as 0.5. We used an open-sourced implementation of SEDANSPOT, provided by the authors, following parameter settings as suggested in the original paper (sample size 500).

Q1. Accuracy

Figure 2 plots the ROC curve for MIDAS-R, MIDAS and SEDANSPOT. Figure 3(top) plots accuracy (AUC) vs. running time (log scale, in seconds, excluding I/O). We see that

MIDAS achieves a much higher accuracy ($= 0.94$) compared to the baseline ($= 0.64$), while also running significantly faster ($0.31s$ vs. $156s$). This is a 46% accuracy improvement at $505\times$ faster speed. MIDAS-R achieves the highest accuracy ($= 0.977$) which is 52% accuracy improvement compared to the baseline at $163\times$ faster speed.

Figure 3(bottom) plots the average precision score vs. running time. We see that MIDAS is more precise ($= 0.969$) compared to the baseline ($= 0.751$). This is a 29% precision improvement. MIDAS-R achieves the highest average precision score ($= 0.987$) which is 31% more precise than SEDANSPOT.

We see that MIDAS and MIDAS-R greatly outperform SEDANSPOT on both accuracy and precision metrics.

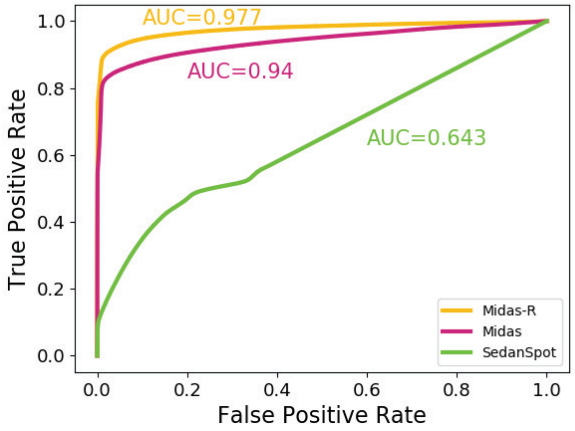

Figure 2: ROC for *DARPA* dataset

Q2. Scalability

Figure 4 shows the scalability of MIDAS and MIDAS-R. We plot the wall-clock time needed to run on the (chronologically) first $2^{12}, 2^{13}, 2^{14}, ..., 2^{22}$ edges of the *DARPA* dataset. This confirms the linear scalability of MIDAS and MIDAS-R with respect to the number of edges in the input dynamic graph due to its constant processing time per edge. Note that both MIDAS and MIDAS-R process $4M$ edges within 1 second, allowing real-time anomaly detection.

Figure 5 plots the number of edges (in millions) and time to process each edge for *DARPA* dataset. MIDAS processes $2.38M$ edges within 1μs each and $1.77M$ edges within 2μs each. MIDAS-R processes $1.04M$ edges within 1μs each and $2.24M$ edges within 2μs each.

Table 2 shows the time it takes SEDANSPOT, MIDAS and MIDAS-R to run on the *TwitterWorldCup*, *TwitterSecurity* and *DARPA* datasets. For *TwitterWorldCup* dataset, we see that MIDAS-R is $108\times$ faster than SEDANSPOT ($0.48s$ vs. $52.17s$) and MIDAS is $306\times$ faster than SEDANSPOT($0.17s$ vs $52.17s$). For *TwitterSecurity* dataset, we see that MIDAS-R is $112\times$ faster than SEDANSPOT ($0.68s$ vs. $76.6s$) and MIDAS is $283\times$ faster than SEDANSPOT($0.27s$ vs $76.6s$). For the *DARPA* dataset, we see that MIDAS-R is $163\times$ faster

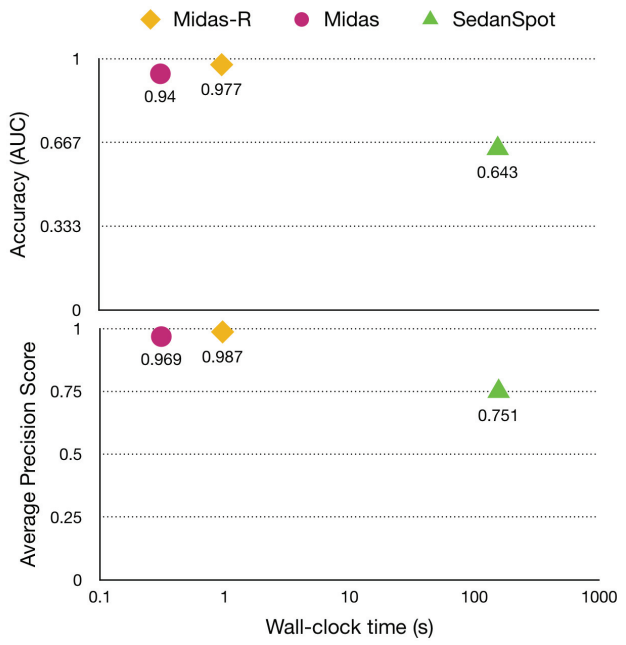

Figure 3: (top) Accuracy (AUC) vs time, (bottom) Average Precision Score vs time

than SEDANSPOT ($0.96s$ vs. $156s$) and MIDAS is $505\times$ faster than SEDANSPOT($0.31s$ vs $156s$).

SEDANSPOT requires several subprocesses (hashing, random-walking, reordering, sampling, etc), resulting in the large computation time. MIDAS and MIDAS-R are both both scalable and fast.

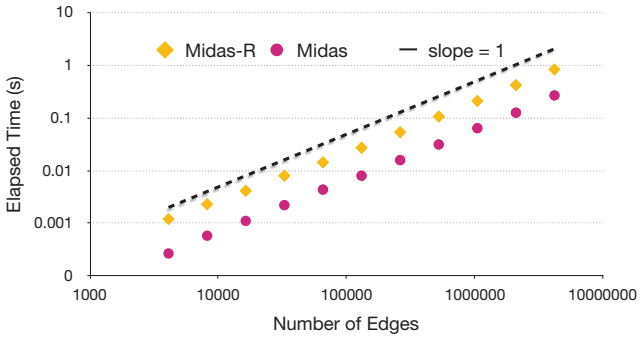

Figure 4: MIDAS and MIDAS-R scale linearly with the number of edges in the input dynamic graph.

Table 2: Running time for different datasets in seconds

	SEDANSPOT	MIDAS	MIDAS-R
TwitterWorldCup	52.17s	0.17s	0.48s
TwitterSecurity	76.60s	0.27s	0.68s
DARPA	156.66s	0.31s	0.96s

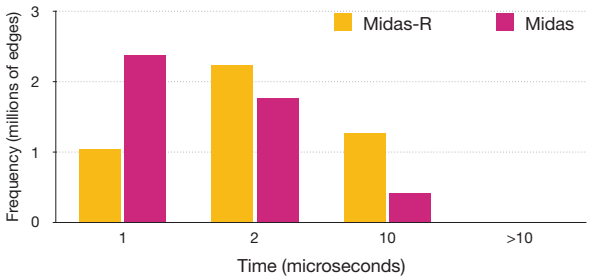

Figure 5: Distribution of processing times for $\sim 4.5M$ edges of *DARPA* dataset.

Q3. Real-World Effectiveness

We measure anomaly scores using MIDAS, MIDAS-R and SEDANSPOT on the *TwitterSecurity* dataset. Figure 6 plots anomaly scores vs. day (during the four months of 2014). To visualise, we aggregate edges occurring in each day by taking the max anomalousness score per day, for a total of 90 days. Anomalies correspond to major world news such as Mpeketoni attack (Event 6) or Soma Mine explosion (Event 1). MIDAS and MIDAS-R show similar trends whereas SEDANSPOT misses some anomalous events (Events $2, 7$), and outputs many high scores unrelated to any true events. This is also reflected in the low accuracy and precision of SEDANSPOT in Figure 3. The anomalies detected by MIDAS and MIDAS-R coincide with major 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.
   03-06-14. Suicide bombing
4. 09-06-14. Suicide/Truck bombings.
5. 10-06-2014. Iraqi Militants Seized Large Regions.
   11-06-2014. Kidnapping
6. 15-06-14. Attack
7. 26-06-14. Suicide Bombing/Shootout/Raid
8. 03-07-14. Israel Conflicts with Hamas in Gaza.
9. 18-07-14. Airplane with 298 Onboard was Shot Down over Ukraine.
10. 30-07-14. Ebola Virus Outbreak.

This shows the effectiveness of MIDAS and MIDAS-R for catching real-world anomalies.

Microcluster anomalies: Figure 7 corresponds to Event 7 in the *TwitterSecurity* dataset. All single edges are equivalent to 444 edges and double edges are equivalent to 888 edges between the nodes. This suddenly arriving (within 1 day) group of suspiciously similar edges is an example of a microcluster anomaly which MIDAS-R detects, but SEDANSPOT misses.

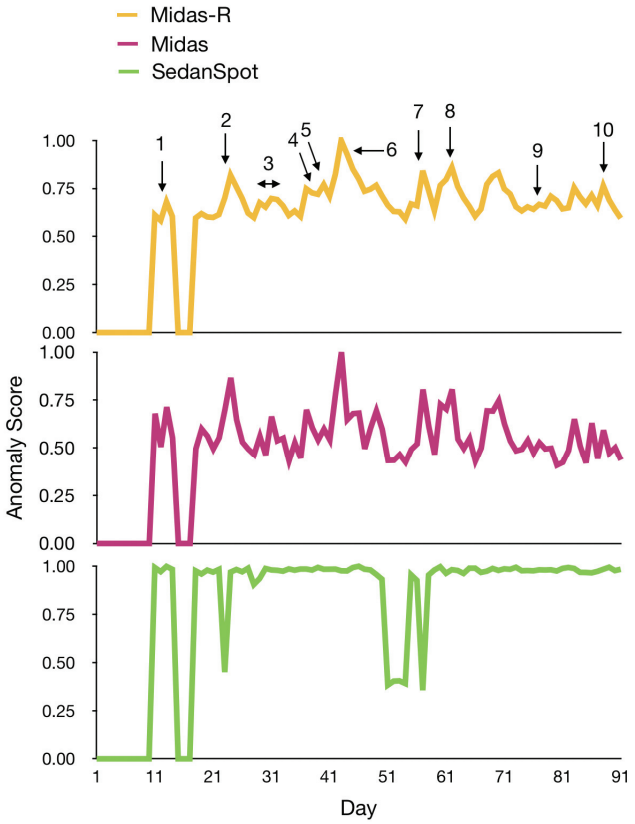

Figure 6: Anomalies detected by MIDAS and MIDAS-R correspond to major security-related events in *TwitterSecurity*.

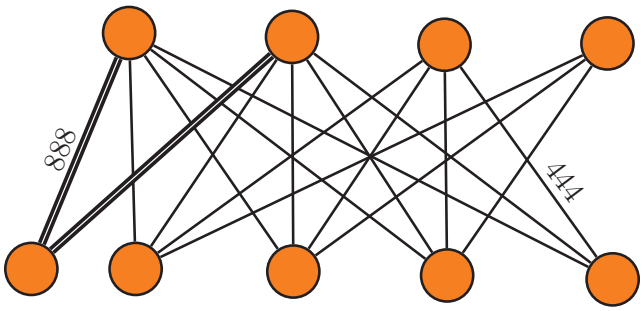

Figure 7: Microcluster Anomaly in *TwitterSecurity*

Conclusion

In this paper, we proposed MIDAS and MIDAS-R 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:

1. Streaming Microcluster Detection: We propose a novel streaming approach for detecting 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 46%-52% accuracy (in terms of AUC), and processes the data $108-505$ times faster than baseline approaches.

Acknowledgments

This work was supported in part by NUS ODPRT Grant R-252-000-A81-133.

References

Akoglu, L.; McGlohon, M.; and Faloutsos, C. 2010. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*.

Akoglu, L.; Tong, H.; and Koutra, D. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29(3):626–688.

Beutel, A.; Xu, W.; Guruswami, V.; Palow, C.; and Faloutsos, C. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*.

Chakrabarti, D. 2004. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*.

Cormode, G., and Muthukrishnan, S. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55(1):58–75.

Eswaran, D., and Faloutsos, C. 2018. Sedanspot: Detecting anomalies in edge streams. In *2018 IEEE International Conference on Data Mining (ICDM)*, 953–958. IEEE.

Eswaran, D.; Faloutsos, C.; Guha, S.; and Mishra, N. 2018. Spotlight: Detecting anomalies in streaming graphs. In *KDD*.

Gupta, M.; Gao, J.; Sun, Y.; and Han, J. 2012. Integrating community matching and outlier detection for mining evolutionary community outliers. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, 859–867. New York, NY, USA: ACM.

Hooi, B.; Shin, K.; Song, H. A.; Beutel, A.; Shah, N.; and Faloutsos, C. 2017. Graph-based fraud detection in the face of camouflage. *TKDD* 11(4):44.

Jiang, M.; Cui, P.; Beutel, A.; Faloutsos, C.; and Yang, S. 2016. Catching synchronized behaviors in large networks: A graph mining approach. *TKDD* 10(4):35.

Kleinberg, J. M. 1999. Authoritative sources in a hyperlinked environment. *JACM* 46(5):604–632.

Koutra, D.; Vogelstein, J. T.; and Faloutsos, C. 2013. Deltacon: A principled massive-graph similarity function. *arXiv preprint arXiv:1304.4657*.

Lippmann, R.; Cunningham, R. K.; Fried, D. J.; Graf, I.; Kendall, K. R.; Webster, S. E.; and Zissman, M. A. 1999. Results of the darpa 1998 offline intrusion detection evaluation. In *Recent advances in intrusion detection*, volume 99, 829–835.

Ranshous, S.; Harenberg, S.; Sharma, K.; and Samatova, N. F. 2016. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, 189–197. SIAM.

Rayana, S., and Akoglu, L. 2015. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, 622–630. SIAM.

Rayana, S., and Akoglu, L. 2016. Less is more: Building selective anomaly ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10(4):42.

Shin, K.; Hooi, B.; Kim, J.; and Faloutsos, C. 2017. Densealert: Incremental dense-subtensor detection in tensor streams. *KDD*.

Shin, K.; Eliassi-Rad, T.; and Faloutsos, C. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *KAIS* 54(3):677–710.

Sricharan, K., and Das, K. 2014. Localizing anomalous changes in time-evolving graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, 1347–1358. New York, NY, USA: ACM.

Sun, J.; Faloutsos, C.; Papadimitriou, S.; and Yu, P. 2007. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 687–696.

Sun, J.; Tao, D.; and Faloutsos, C. 2006. Beyond streams and graphs: dynamic tensor analysis. In *KDD*.

Tong, H., and Lin, C.-Y. 2011. Non-negative residual matrix factorization with application to graph anomaly detection. In *SDM*.

Yoon, M.; Hooi, B.; Shin, K.; and Faloutsos, C. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 647–657. ACM.

Yu, W.; Aggarwal, C. C.; Ma, S.; and Wang, H. 2013. On anomalous hotspot discovery in graph streams. In *ICDM*.