# Midas: Microcluster-Based Detector of Anomalies in Edge Streams

Siddharth Bhatia
National University of Singapore
siddharth@comp.nus.edu.sg

Bryan Hooi
National University of Singapore
bhooi@comp.nus.edu.sg

Minji Yoon
Carnegie Mellon University
minjiy@cs.cmu.edu

Kijung Shin
KAIST
kijungs@kaist.ac.kr

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

## 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 162 − 644 times faster than state-of-the-art approaches; (c) it provides 42%-48% higher accuracy (in terms of AUC) than state-of-the-art approaches.

***Keywords:*** Edge Streams, Microcluster, Dynamic Graphs, Anomaly Detection

**ACM Reference Format:**
Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: Microcluster-Based Detector of Anomalies in Edge Streams. In *MLG, KDD, August 24, 2020, San Diego, USA.* ACM, New York, NY, USA, 8 pages.

## 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 well-researched problem with majority of the proposed approaches [1, 4, 9–11, 17] 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 [7, 8, 12, 19–21]. 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 [6, 14], 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 42%-48% accuracy (in terms of AUC), and processes the data 162 − 644 times faster than baseline approaches.

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

## 2 Related Work

In this section, we review previous approaches to detect anomalous signs on static and dynamic graphs. See [2] 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: [1] 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. [10] computes node features, including degree and authoritativeness [11], then spots nodes whose neighbors are notably close in the feature space.
- Anomalous subgraph detection: [9] and [17] measure the anomalousness of nodes and edges, detecting a dense subgraph consisting of many anomalous nodes and edges.
- Anomalous edge detection: [4] encodes an input graph based on similar connectivity among nodes, then spots edges whose removal reduces the total encoding cost significantly. [22] 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: [21] 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, [3] spots near-bipartite cores where each node is connected to others in the same core densly within a short time. [10] detects groups of nodes who form dense subgraphs in a temporally synchronized manner.
- Anomalous event detection: [7] detects sudden appearance of many unexpected edges, and [23] 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, [24] detects nodes whose egonets suddenly and significantly change.
- Anomalous subgraph detection: Given an edge stream, [18] identifies dense subtensors created within a short time.
- Anomalous edge detection: [14] focuses on sparsely-connected parts of a graph, while [6] 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 [6] | RHSS [14] | **Midas** |
|---|---|---|---|
| **Microcluster Detection** | | | ✔ |
| **Guarantee on False Positive Probability** | | | ✔ |
| **Constant Memory** | ✓ | ✓ | ✔ |
| **Constant Update Time** | ✓ | ✓ | ✔ |

## 3 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 $\epsilon$ (e.g. 1%), the algorithm should be adjustable so as to provide false positive probability of at most $\epsilon$ (e.g. by adjusting a

threshold that depends on $\epsilon$). 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 Proposed Algorithm

### 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

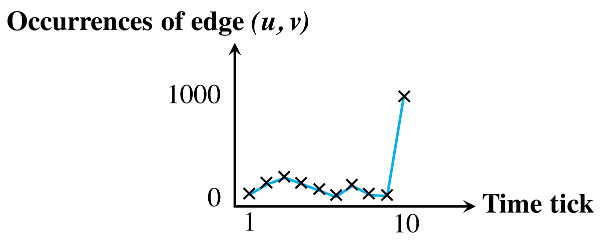

**Figure 1.** Time series of a single source-destination pair $(u, v)$, with a large burst of activity at time tick 10.

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 microcluster, 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**.

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

#### 4.2.1 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) [5] 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:

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

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:*

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

Algorithm 1 summarizes our MIDAS algorithm.

**Algorithm 1:** MIDAS: Streaming Anomaly Scoring

```
Input: Stream of graph edges over time
Output: Anomaly scores per edge
1 ▷ Initialize CMS data structures:
2 Initialize CMS for total count $s_{uv}$ and current count $a_{uv}$
3 while new edge $e = (u, v, t)$ is received: do
4     ▷ Update Counts:
5     Update CMS data structures for the new edge $uv$
6     ▷ Query Counts:
7     Retrieve updated counts $\hat{s_{uv}}$ and $\hat{a_{uv}}$
8     ▷ Anomaly Score:
9     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 $\epsilon$, the probability of a false positive should be at most $\epsilon$. 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 $\epsilon$ and $\nu$, for appropriately chosen CMS data structure sizes, with probability at least $1 - \frac{\epsilon}{2}$, the estimates $\hat{a_{uv}}$ satisfy:

$$\hat{a_{uv}} \leq a_{uv} + \nu \cdot N_t \tag{2}$$

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

$$s_{uv} \leq \hat{s_{uv}} \tag{3}$$

Define an adjusted version of our earlier score:

$$\tilde{a_{uv}} = \hat{a_{uv}} - \nu N_t \tag{4}$$

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

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

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:*

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

*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 $\epsilon$.*

*Proof.* Recall that

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

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

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

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

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

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

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

□

### 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 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.

### 4.5 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 $O(w)$.

## 5 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:

**Algorithm 2:** MIDAS-R: Incorporating Relations

```
   Input: Stream of graph edges over time
   Output: Anomaly scores per edge
1  ▷ Initialize CMS data structures:
2  Initialize CMS for total count $s_{uv}$ and current count $a_{uv}$
3  Initialize CMS for total count $s_u$ and current count $a_u$
4  while new edge $e = (u, v, t)$ is received: do
5      ▷ Update Counts:
6      Update CMS data structures for the new edge $uv$
7      ▷ Query Counts:
8      Retrieve updated counts $\hat{s_{uv}}$ and $\hat{a_{uv}}$
9      Retrieve updated counts $\hat{s_u}, \hat{s_v}, \hat{a_u}, \hat{a_v}$
10     ▷ Compute Edge Scores:
11     $\text{score}(u, v, t) = (\hat{a_{uv}} - \frac{\hat{s_{uv}}}{t})^2 \frac{t^2}{\hat{s_{uv}}(t-1)}$
12     ▷ Compute Node Scores:
13     $\text{score}(u, t) = (\hat{a_u} - \frac{\hat{s_u}}{t})^2 \frac{t^2}{\hat{s_u}(t-1)}$
14     $\text{score}(v, t) = (\hat{a_v} - \frac{\hat{s_v}}{t})^2 \frac{t^2}{\hat{s_v}(t-1)}$
15     ▷ Final Node Scores:
16     output $\max\{\text{score}(u, v, t), \text{score}(u, t), \text{score}(v, t)\}$
```

**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* [13] has 4.5$M$ IP-IP communications between 9.4$K$ source IP and 23.3$K$ destination IP over 87.7$K$ 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 60.1% of total).

*TwitterSecurity* [15, 16] has 2.6$M$ 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* [15, 16] has 1.7$M$ 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 [6] 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.

## 5.1 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 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 $\alpha$ 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).

## 5.2 Q1. Accuracy

Figure 2 plots the ROC curve for Midas-R, Midas and SedanSpot on the *DARPA* dataset. 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 (= 0.91) compared to the baseline (= 0.64), while also running significantly faster (0.13$s$ vs. 84$s$). This is a 42% accuracy improvement at 644× faster speed. Midas-R achieves the highest accuracy (= 0.95) which is 48% accuracy improvement compared to the baseline at 215× faster speed.

Figure 3(bottom) plots the average precision score vs. running time. We see that Midas is more precise (= 0.95) compared to the baseline (= 0.75). This is a 27% precision improvement. Midas-R achieves the highest average precision score (= 0.97) which is 29% more precise than SedanSpot.

We see that Midas and Midas-R greatly outperform SedanSpot on both accuracy and precision metrics.

## 5.3 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 0.5 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 4.4$M$ edges within 1$\mu s$ each and 0.15$M$ edges within 2$\mu s$ each. Midas-R processes 4.3$M$ edges within 1$\mu s$ each and 0.23$M$ edges within 2$\mu s$ each.

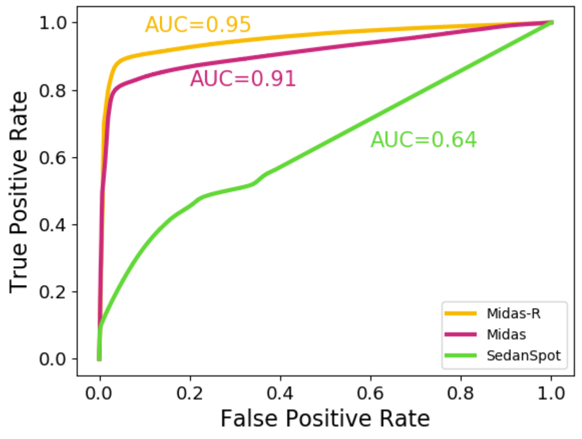

**Figure 2.** ROC for *DARPA* dataset

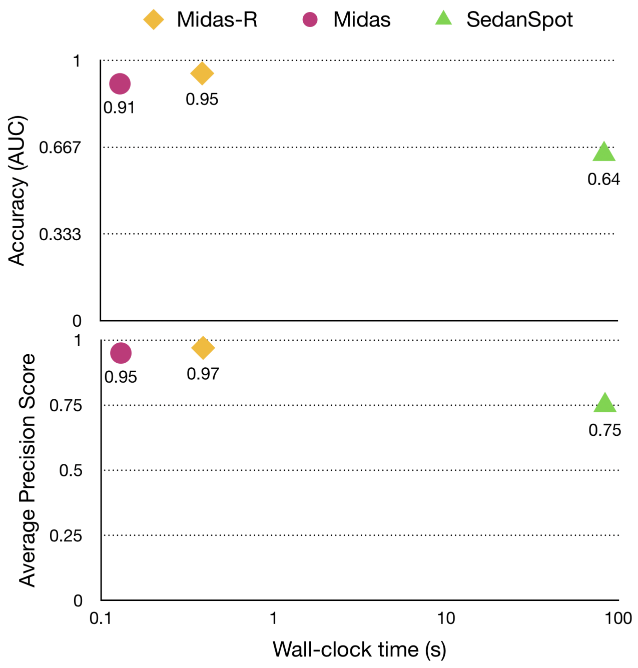

**Figure 3.** (top) Accuracy (AUC) vs time, (bottom) Average Precision Score vs time

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 162× faster than SedanSpot (0.17$s$ vs. 27.58$s$) and Midas is 460× faster than SedanSpot(0.06$s$ vs 27.58$s$). For *TwitterSecurity* dataset, we see that Midas-R is 177× faster than SedanSpot (0.23$s$ vs. 40.71$s$) and Midas is 509× faster than SedanSpot(0.08$s$ vs 40.71$s$). For the *DARPA* dataset, we see that Midas-R is 215× faster than SedanSpot

($0.39s$ vs. $83.66s$) and Midas is 644× faster than SedanSpot ($0.13s$ vs $83.66s$).

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

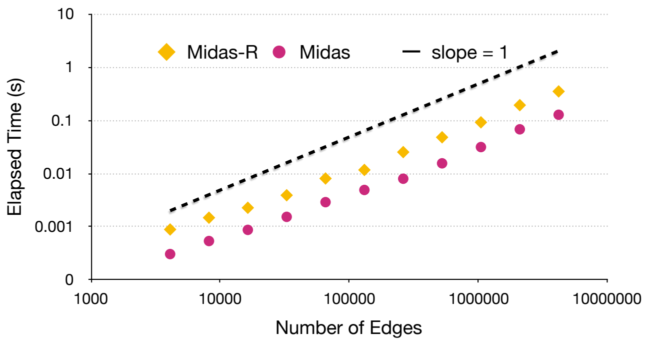

**Figure 4.** Midas and Midas-R scale linearly with the number of edges in the input dynamic graph.

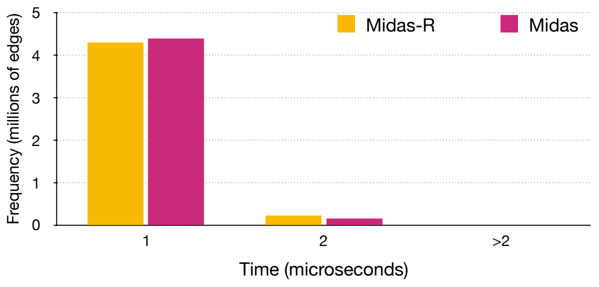

**Figure 5.** Distribution of processing times for $\sim 4.5M$ edges of *DARPA* dataset.

**Table 2.** Running time for different datasets in seconds

| | SedanSpot | Midas | Midas-R |
|---|---|---|---|
| **TwitterWorldCup** | 27.58s | 0.06s | 0.17s |
| **TwitterSecurity** | 40.71s | 0.08s | 0.23s |
| **DARPA** | 83.66s | 0.13s | 0.39s |

### 5.4 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.

## 6 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 42%-48% accuracy (in terms of AUC), and processes the data 162 − 644 times faster than baseline approaches.

## 7 Acknowledgments

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

## References

[1] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*.
[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* (2015).

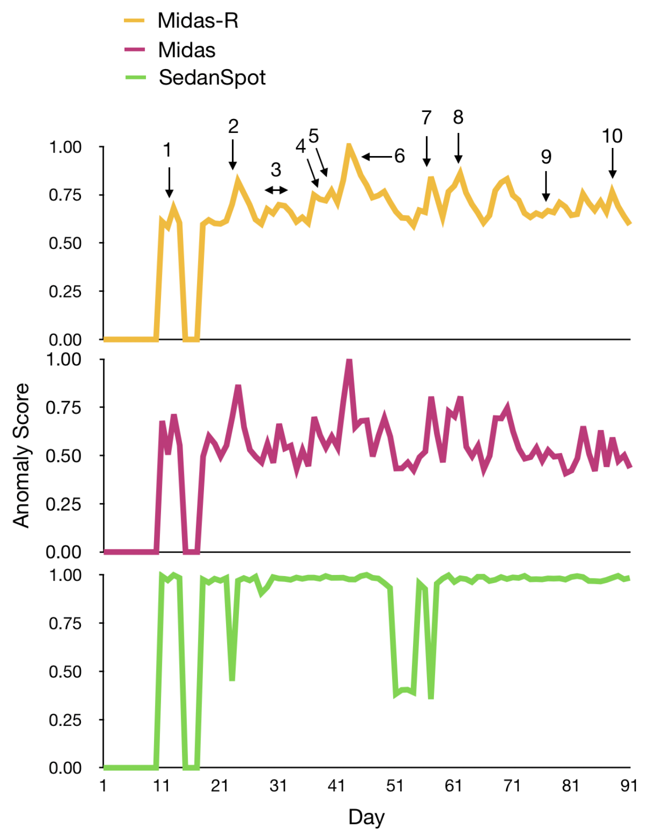

**Figure 6.** Anomalies detected by MIDAS and MIDAS-R correspond to major security-related events in *TwitterSecurity*.

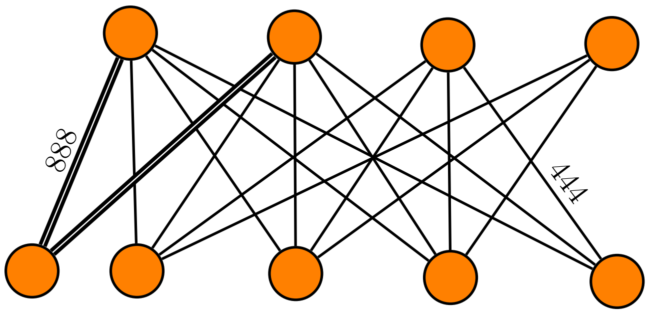

**Figure 7.** Microcluster Anomaly in *TwitterSecurity*

[3] 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*.

[4] Deepayan Chakrabarti. 2004. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*.

[5] 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.

[6] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *ICDM*. IEEE, 953–958.

[7] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. SpotLight: Detecting Anomalies in Streaming Graphs. In *KDD*.

[8] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. 2012. Integrating Community Matching and Outlier Detection for Mining Evolutionary Community Outliers. In *KDD*.

[9] Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. 2017. Graph-based fraud detection in the face of camouflage. *TKDD* 11, 4 (2017), 44.

[10] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2016. Catching synchronized behaviors in large networks: A graph mining approach. *TKDD* 10, 4 (2016), 35.

[11] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *JACM* 46, 5 (1999), 604–632.

[12] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. 2013. Deltacon: A principled massive-graph similarity function. *arXiv preprint arXiv:1304.4657* (2013).

[13] 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*, Vol. 99. 829–835.

[14] 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*. SIAM, 189–197.

[15] Shebuti Rayana and Leman Akoglu. 2015. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *SDM*.

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

[17] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *KAIS* 54, 3 (2018), 677–710.

[18] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. DenseAlert: Incremental Dense-Subtensor Detection in Tensor Streams. *KDD* (2017).

[19] Kumar Sricharan and Kamalika Das. 2014. Localizing Anomalous Changes in Time-evolving Graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. 1347–1358.

[20] J. Sun, C. Faloutsos, S. Papadimitriou, and P.S. Yu. 2007. GraphScope: parameter-free mining of large time-evolving graphs. In *KDD*.

[21] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *KDD*.

[22] Hanghang Tong and Ching-Yung Lin. 2011. Non-negative residual matrix factorization with application to graph anomaly detection. In *SDM*.

[23] 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*. ACM.

[24] Weiren Yu, Charu C Aggarwal, Shuai Ma, and Haixun Wang. 2013. On anomalous hotspot discovery in graph streams. In *ICDM*.