EigenPulse: Detecting Surges in Large Streaming Graphs with Row Augmentation

Jiabao Zhang[1,2], Shenghua Liu[1,2](✉), Wenjian Yu[3](✉), Wenjie Feng[1,2],
and Xueqi Cheng[1,2]

[1] CAS Key Laboratory of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
liu.shengh@gmail.com
[2] University of Chinese Academy of Sciences, Beijing, China
zhangjiabao18@mails.ucas.edu.cn
[3] BNRist, Department of Computer Science and Technology, Tsinghua University,
Beijing, China
yu-wj@tsinghua.edu.cn

Abstract. How can we spot dense blocks in a large streaming graph efficiently? Anomalies such as fraudulent attacks, spamming, and DDoS attacks, can create dense blocks in a short time window, emerging a surge of density in a streaming graph. However, most existing methods detect dense blocks in a static graph or a snapshot of dynamic graphs, which need to inefficiently rerun the algorithms for a streaming graph. Moreover, some works on streaming graphs are either consuming much time on updating algorithm for every incoming edge, or spotting the whole snapshot of a graph instead of the attacking sub-block.

Therefore, we propose a row-augmented matrix with sliding window to model a streaming graph, and design the *AugSVD* algorithm for computation- and memory-efficient singular decomposition. *EigenPulse* is then proposed to spot the density surges in streaming graphs based on the singular spectrum. We theoretically analyze the robustness of our method. Experiments on real datasets with injections show our performance and efficiency compared with the state-of-the-art baseline.

Keywords: Surge detection · Streaming graphs · Sliding window

1 Introduction

The surges of density in some subgraph are a strong signal to detect anomalies in streaming graphs [2,13]. For example, the controlled user accounts rate fake and high scores to a set of target objects in a short period of time, in Amazon, Yelp, App stores, etc. The spamming phone calls/msgs are sent intensively from a group of phone numbers to another group. And the attacks to a set of servers of target websites from a large pool of IPs. Those cases will result a very dense

Q. Yang et al. (Eds.): PAKDD 2019, LNAI 11440, pp. 501–513, 2019.
https://doi.org/10.1007/978-3-030-16145-3_39

subgraph between some users and objects, phone numbers, and IPs in a short time window. Thus the question is raised:

How can we detect such a dense subgraph, and spot the density surge in a large streaming graph in an efficient and accurate way?

Many existing dense subgraph detection methods, such as M-zoom [11], D-Cube [12], HoloScope [9], have achieved satisfied accuracy in large static graphs. Re-running those methods once a batch of new data comes is very time-consuming and low efficiency, in a streaming graph. The recent work, SpotLight [2], can efficiently detect the sudden changes of a snapshot of the graph in a time period. It was not able to tell which specific part of the snapshot is attacked. DenseAlert [13] detects dense subgraph using an incremental and heuristic algorithm, which updates for every single incoming edge, in order to have a high accuracy. This slows down the algorithm, even though DenseAlert is faster than the batch methods.

Therefore, we reasonably model the streaming graph as a row-augmented matrix, and propose, EigenPulse, to detect surges in large streaming graphs, based on the singular spectrum of the matrix. To get the singular spectrum of a row-augmented matrix, we propose AugSVD for singular decomposition of the streaming graph in a sliding window. Even if attacks may cross windows, we can still detect them since the windows intersect. AugSVD outputs the singular spectrum of every stride, and EigenPulse algorithm calculate the density of a subgraph in first several singular vectors and detect anomalies. The experiments on 5 real data sets show that EigenPulse can detect the suspicious surges of density of some subgraph, achieving high accuracy as the baselines, but consuming much less computation time.

In summary, the main advantages of our algorithms are:

- **Incremental singular value decomposition:** we propose a scalable algorithm, AugSVD, to decompose large streaming graphs, which can output the spectral values of graph nodes at each time window, as long as the graphs can be formulated as matrices augmented in rows.
- **Robust and effective:** we theoretically analyze that the robust approximation of AugSVD to batch SVD. The experiments show that the EigenPulse generated by AugSVD can detect suspicious synchronized activities accurately in real-world graphs.
- **Scalable:** EigenPulse is computation- and memory-efficient. Compared with the state-of-the-art baseline, EigenPulse can be more than 5 times faster.

2 Related Work

2.1 Anomaly Detection in Static Graphs

Anomaly detection in static graphs is well studied in [1]. For example, methods based on spectral decomposition, e.g., EigenSpokes [10], which discovers that the induced sub-graph of the 20 nodes which had the highest magnitude projection along the singular vector almost contains near-cliques. Many existing methods rely on graph's density, e.g., Fraudar [6], several methods even taking into account rating variation and burst of attacks, e.g., CrossSpot [7] and HoloScope which is the only one considers temporal spikes and hyperbolic topology.

2.2 Anomaly Detection in Streaming Graphs

We summarize dense subgraph detection algorithms in streaming graphs. Traditional methods just compare the changes of adjacent graphs via a similarity function based on, e.g., belief propagation [8]. They do not consider evolutionary/periodic trends. Many existing methods, e.g., DenseAlert model dynamic graphs as streaming tensors and aim to approximately identify the top-k densest subblocks, i.e., maintained dense subtensors. DenseAlert divide time into bins and can detect sudden emerging dense subtensors, same with EigenPulse. In contrast, Spotlight detects only the sudden appearance or disappearance of dense blocks in real-time by using randomized sketching-based approach. [15] designs an algorithm MASCOT for counting local triangles to detect anomalies in graph streams. There are some methods based on graph decomposition and partitioning, such as [14] storing a summary of the graph structure based on tensor decomposition and identify change points as anomalies. Some randomized algorithms, i.e., [4] defines a robust random cut data structure that can be used as a sketch or synopsis of the input stream. Some other methods find patterns for anomaly detection, e.g., [3] investigates continuous pattern detection over large evolving graphs with snapshot isolation.

3 Proposed Method

We define a bipartite graph $\mathcal{G}$ to represent the relationships between users and rating objects, callers and callees, attacking IPs and target IPs, etc. The problem that detects surges of density in a large streaming graph $\mathcal{G}$ is described as follows:

Informal Problem 1 (Detecting Density Surges) ***Given:*** *a stream of triplets (user, object, timestamp), where timestamp is the time when a user creates an edge on an object, and a time window of width w,*

- ***find:*** *at each time step t, calculate the density D_t of the subgraph that is the densest one in streaming graph $\mathcal{G}$ within current time window;*
- *detect suspicious surges of density that are above a threshold.*

In our problem, a triplet $(user, object, timestamp)$ is a new edge created in streaming graph $\mathcal{G}$. The triplets come in an order of *timestamp*. A streaming graph $\mathcal{G}$ within a time window indicates that only the triplets coming in the time window are considered as the edges in graph $\mathcal{G}$. The time windows are sliding at each time step.

3.1 Our Model

To develop a fast algorithm for singular decomposition, we model large streaming graph $\mathcal{G}$ as a row-augmented matrix $\mathbf{A}$.

Row-Augmented Matrix: Matrix which is modified in a row augmented manner. For each new piece of data, its corresponding row is incremental or just same with the last row of current matrix.

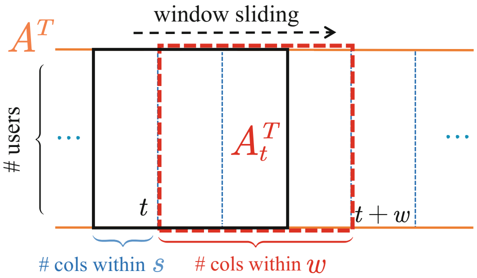


Fig. 1. The sliding window for the row-augmented matrix. w is the window size in a time unit, s is the stride size in a time unit. Note that the number of columns in different time windows may be different.

Figure 1 shows the sliding window for $\mathbf{A}^T$, which $\mathbf{A}$ is the row-augmented matrix. The columns of row-augmented matrix $\mathbf{A}$ represent the *user* ids in streaming graph $\mathcal{G}$. The rows are increasing, and each row is a combination of an *object* id and *timestamp*/*s*, where s is the stride to a next step. Such ids guarantee the rows coming in the next step are totally different, keeping the property of row-augmented matrix. Note that our model of $\mathbf{A}$ is actually batch-row augmented, and the batch size is decided by the number of new edges between stride s.

We now explain why we can model a streaming graph $\mathcal{G}$ with a row-augmented matrix $\mathbf{A}$. One reason is that since fraudsters and attackers create edges in a relatively short period of time, combining the object ids with binned time can still show a dense block in our matrix $\mathbf{A}$ for anomalies. Besides, an object in different time can mean differently, e.g. the same app may be different versions at different time, the same restaurant and product may have improvement or new generation at later time. With such a combination, one can consider those differences. In another case, a piece of twitter message or news as an object probably no users will read after a short while, which reduces the bias of our model. Finally, the most important of all, such a model, can help us achieve fast algorithm to detect density surges, which is described in the following sections.

Similarly, we can still introduce a sliding window for row-augmented matrix $\mathbf{A}$ as show in Fig. 1. With such a window, we can make algorithm focus on the most recent edges in graph $\mathcal{G}$. When assigning the width of window w as infinity, we can consider all the history at every time step. Or we can have non-overlapped dense subgraphs by setting $w = s$.

Algorithm 1. AugSVD with sliding window

```
Input: row-augmented matrix $\mathbf{A}$ with sliding window $w$, column size $n$, rank parameter $k$, block size $b$, two queues $glist$ and $hlist$.
1: Choose $l = tb$, where $t$ is an integer, so that $l$ is slightly larger than $k$
2: $\mathbf{\Omega} = randn(n, l)$;   $\mathbf{G} = [\ ]$;   set $\mathbf{H}$ to an $n \times l$ zero matrix
3: if $glist$ is not empty then
4:     $glist.dequeue()$;   $hlist.dequeue()$
5: end if
6: repeat
7:     Read rows $\mathbf{a}$ for next stride $s$
8:     $\mathbf{g} = \mathbf{a}\mathbf{\Omega}$;   $\mathbf{h} = \mathbf{a}^T\mathbf{g}$
9:     $glist.enqueue(\mathbf{g})$;   $hlist.enqueue(\mathbf{h})$
10: until the elements in $glist$ corresponds to a window $w$
11: for all $\mathbf{g}$ in $glist$, $\mathbf{h}$ in $hlist$ do
12:     $\mathbf{G} = [\mathbf{G}, \mathbf{g}]$;   $\mathbf{H} = \mathbf{H} + \mathbf{h}$
13: end for
14: $\mathbf{Q} = [\ ]$;   $\mathbf{B} = [\ ]$
15: for $i = 1, 2, \cdots, t$ do
16:     $\mathbf{\Omega}_i = \mathbf{\Omega}(:, (i-1)b+1 : ib)$; $\mathbf{Y}_i = \mathbf{G}(:, (i-1)b+1 : ib) - \mathbf{Q}(\mathbf{B}\mathbf{\Omega}_i)$
17:     $[\mathbf{Q}_i, \mathbf{R}_i] = qr(\mathbf{Y}_i)$
18:     $[\mathbf{Q}_i, \widetilde{\mathbf{R}}_i] = qr(\mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^T\mathbf{Q}_i))$
19:     $\mathbf{R}_i = \widetilde{\mathbf{R}}_i\mathbf{R}_i$
20:     $\mathbf{B}_i = \mathbf{R}_i^{-T}(\mathbf{H}(:, (i-1)b+1 : ib)^T - \mathbf{Y}_i^T\mathbf{Q}\mathbf{B} - \mathbf{\Omega}_i^T\mathbf{B}^T\mathbf{B})$
21:     $\mathbf{Q} = [\mathbf{Q}, \mathbf{Q}_i]$;   $\mathbf{B} = [\mathbf{B}^T, \mathbf{B}_i^T]^T$
22: end for
23: $[\widetilde{\mathbf{U}}, \mathbf{S}, \mathbf{V}] = svd(\mathbf{B})$
24: $\mathbf{U} = \mathbf{Q}\widetilde{\mathbf{U}}$
25: $\mathbf{U} = \mathbf{U}(:, 1 : k)$;   $\mathbf{V} = \mathbf{V}(:, 1 : k)$;   $\mathbf{S} = \mathbf{S}(1 : k, 1 : k)$
26: return $\mathbf{U}, \mathbf{S}, \mathbf{V}$
```

3.2 AugSVD Algorithm

AugSVD is designed for fast singular decomposition of row-augmented matrix $\mathbf{A}$ with sliding windows. It involves only one pass over the data and having accuracy guarantees. The algorithm is described as Algorithm 1.

Initially, for the first window, the queues *glist* and *hlist* are empty. The AugSVD algorithm outputs the first k singular values and vectors for the data observed through the first window. At the second time invoking the algorithm, the window slides one stride forward to form the next window.In such a way, repeatedly calling AugSVD results in the singular vectors of row-augmented matrix $\mathbf{A}$ observed from the sliding windows. Such an algorithm outperforms the standard SVD and other existing randomized algorithms by largely reducing runtime and memory usage.

In Algorithm 1, Steps 3 through 13 prepares matrices $\mathbf{G}$ and $\mathbf{H}$ for the sliding window, while Steps 14 through 25 are just the same as those in the single-pass PCA algorithm [16]. Due to the accumulation of round-off errors, the orthonormality among the columns in $\{\mathbf{Q}_1, \mathbf{Q}_2, \cdots\}$ may lose. To fix this issue, $\mathbf{Q}_i$ is explicitly re-projected away from the span of the previously computed basis vectors (Step 19), just as what is done in [16].

Theorem 1. *The AugSVD algorithm is mathematically equivalent to the basic randomized SVD algorithm in [5] for the row augmented matrix* $\mathbf{A}$.

Proof. As stated before, the AugSVD algorithm is the same as the single-pass PCA algorithm for streaming data in the sliding window, i.e. the row augmented matrix $\mathbf{A}$. It has been proved in [16] that the single-pass PCA algorithm is mathematically equivalent to the basic randomized SVD algorithm in [5]. So, the theorem is proved.

Based on Theorem 1, the AugSVD algorithm inherits the theoretical error bound (if ignoring the round-off error) [5]:

$$\mathbb{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\| \le \left(1 + \sqrt{\frac{k}{s-1}}\right)\sigma_{k+1} + \frac{\mathrm{e}\sqrt{k+s}}{s}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \tag{1}$$

where $\mathbb{E}$ denotes expectation, $s = l - k$. If choosing $s = k + 1$, we have

$$\mathbb{E}\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T\| \le 2\sigma_{k+1} + \frac{\mathrm{e}\sqrt{2k+1}}{k}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \tag{2}$$

Here, we have substituted the computed SVD factors: $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$ and $\hat{\mathbf{V}}$ with the single-pass PCA algorithm. Applying a rough analysis, we have

$$\mathbb{E}\max_{i=1,\ldots,k} |\sigma_i - \hat{\sigma_i}| = \mathbb{E}\|\mathbf{\Sigma} - \hat{\mathbf{\Sigma}}\| \le 2\sigma_{k+1} + \frac{\mathrm{e}\sqrt{2k+1}}{k}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \tag{3}$$

where σ_i and $\hat{\sigma_i}$ are the accurate and computed the i-th singular value, respectively. Moreover, it can be shown that the likelihood of a substantial deviation from the expectation is extremely small; see Sec. 10.3 of [5] for a proof. This means the expectation symbol in (3) can be removed in an approximate sense. This results in:

$$|\sigma_i - \hat{\sigma_i}| \lesssim 2\sigma_{k+1} + \frac{\mathrm{e}\sqrt{2k+1}}{k}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}, \ i = 1, \ldots, k \tag{4}$$

where $\lesssim$ means less than approximately. The right-hand side of (4) means that the error on singular value depends not only on the (k+1)-th singular value but also the summation of its subsequent singular values. If $\mathbf{A}$'s singular values do not decay slowly, the second right-hand-side term in (4) would be relatively small, which means the computed singular value has sufficient accuracy.

Algorithm 2. EigenPulse

```
Input: time $t$, matrix $\mathbf{A}_t$ within time window $[t-w, t]$, row size $m_t$ and column size
    $n$, a pair of left/right singular vectors $\mathbf{u}_t$, $\mathbf{v}_t$.
$rowset = [\,]$;   $colset = [\,]$
$\tau_u = \frac{1}{\sqrt{m_t}}$;   $\tau_v = \frac{1}{\sqrt{n}}$
for $i = 1, \cdots, m_t$ do
    if $abs(u_t[i]) >= \tau_u$ then
        $rowset.append(i)$
    end if
end for
for $j = 1, \cdots, n$ do
    if $abs(v_t[j]) >= \tau_v$ then
        $colset.append(j)$
    end if
end for
[optional] $rowset, colset = dense\_block\_detection(\mathbf{A}_t, rowset, colset)$
return $D_t(rowset, colset)$
```

3.3 EigenPulse Algorithm

As we known, the nodes in a dense subgraph probably correspond to larger absolute values in the first several singular vectors. The EigenPulse algorithm is used to detect such subgraph and calculate the density measure based the singular vectors computed with AugSVD. It is described as Algorithm 2.

In Algorithm 2, τ_u and τ_v are two thresholds for left and right singular vectors respectively. The density measure is calculated as:

$$D_t(rowset, colset) = \frac{\sum_{i \in rowset} \sum_{j \in colset} \mathbf{A}_t(i,j)}{|rowset| + |colset|} \tag{5}$$

We calculate this density measure for every time window. If it is obviously larger in a window than that in other windows, the window is very suspicious. Optionally, we can use an existing *dense_block_detection* algorithm, such as Fraudar and HoloScope (HS-α), to further shave *rowset* and *colset* to find a densest subblock, which is efficient for a reduced rows and columns (see step 2–12 in Algorithm 2), and benefit from the existing algorithms. To detect suspiciously surging window, one can simply combine mean value with standard deviation of historical density values as a threshold to take out suspicious windows. By this way, we greatly reduce the data needs to be detected than static methods.

4 Experiments

We design experiments to answer the following questions:

Q1.Efificiency: How fast does EigenPulse analyze the real world data compared with competitor?
Q2. Accuracy: How accurately does EigenPulse detect dense blocks?
Q3. Scalability: Does EigenPulse scale linearly with the size of tensor?

Table 1. Datasets statistic information

Name	Nodes	Edges	Span time
BeerAdvocate	26.5K × 50.8K	1.08M	Jan 2008 – Nov 2011
Yelp	686K × 85.3K	2.68M	Oct 2004 – Jul 2016
Amazon Cellphone	2.26M × 329K	3.45M	Jan 2007 – Jul 2014
Amazon Electronics	4.20M × 476K	7.82M	Dec 1998 – Jul 2014
Amazon Grocery	763K × 165K	1.29M	Jan 2007 – Jul 2014
Sina Weibo	2.74M × 8.08M	50.06M	Sep 2013 – Dec 2013

4.1 Experimental Settings

Machine: We ran all experiments on a machine with 2.7 GHz Intel Xeon E7-8837 CPUs and 512 GB memory.

Data: Table 1 lists the real-world datasets used in our experiments. All of the data are 4-way tensors (users, items, timestamps, ratings) where entry values are the number of reviews. In addition, the AugSVD can only decompose matrices augmented in rows, so we first filter the data with high rating scores, then concatenate these items by the timestamp as row, user as column, and thus the row of modified tensor grows with the forward of time.

Implementations: We chose the state-of-the-art streaming dense-subtensor detection algorithm, DenseAlert, as baseline. In all the experiments, we used sparse tensor format and only considered the first pair left/right singular vector.

4.2 Q1.Efficiency

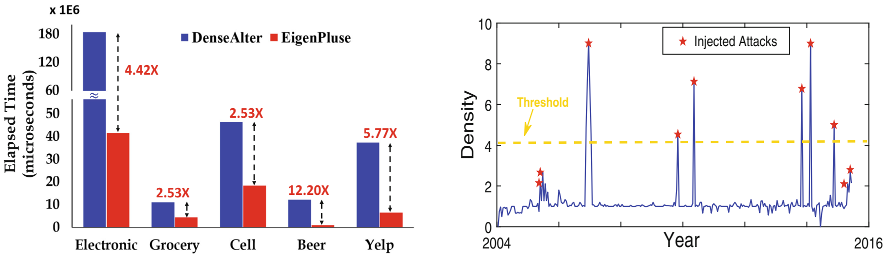
As we see, EigenPulse chooses suspicious windows based on AugSVD, and then combines other shaving algorithms to obtain smaller dense blocks, finally identifies the fraudulent blocks with density measure. Other streaming methods, however, e.g., DenseAlert needs to update dense subtensor every time when coming a new tensor and SpotLight maintains a streaming tensor about graph sketch information in real-time. So EigenPluse is faster than those algorithms.

We measured the wall-clock time taken by EigenPluse and DenseAlert for analyzing the first 5 datasets and showed the results in the Fig. 2(a). The EigenPluse achieves **2.53**× **faster** than DenseAlert, or even achieves **12.2**× speed up in BeerAdvocate dataset. According to the performance results of DenseAlert, which is a million times faster than the fastest batch algorithms, e.g., M-Zoom or CP Decomposition. We can draw the conclusion that EigenPluse significantly outperforms most of the state-of-the-art competitors.

In addition, EigenPulse is memory-efficient for only calculating one window's data each time, which up to **2.33 GB** memory consumed.

4.3 Q2.Accuracy

This experiment demonstrates the accuracy of EigenPulse for detecting dense blocks in different datasets.



(a) **Runtime performance**

(b) **Detection for injected attacks**

Fig. 2. EigenPluse performance: (a). EigenPulse consistently outperforms DenseAlert on 5 datasets, and achieves more than **2.53**× speed up. ('Beer' denotes BeerAdvocate); (b). EigenPluse successfully detects most of the injected attacks on Yelp dataset.

Detection of Injected Attacks: Here, we set $w = 30$ and $s = 10$ in days.

We injected dense blocks with different densities and different speeds to identify the lowest detection density(LDD) and the lowest detection speed(LDS). The unit of detection speed is ($\#edges/days$), referring to the maximum number of injected edges in one day which we can detect. To identify the LDS, we keep the injected density unchanged and change the time span until the F-measure value is less than 90%. We randomly choose 20 products whose in-degrees are no more than 100 because they are more likely to buy fake reviews. Since data in windows is part of all the data, so we should inject small blocks into windows. For 0.1 may be a suitable density, we sample out 200 fraudsters as a whole to randomly create 20 edges on each of the 20 products, and also create biased camouflage on other products. Then, we just vary the time span across the data to find out the LDS for each dataset. Having identified the LDS, We choose a proper time span, e.g., 30 days, then inject blocks with different densities until the F-measure value is less than 90% to identify the LDD. We randomly choose 20 products as mentioned above and sample out fraudsters ranges from 20 to 2000 to generate fraudulent blocks with densities ranges from 1.0 to 0.01 for testing to find the lowest fraudulent density. The time was generated for each fraudulent edge: randomly choosing a time in the window range.

In order to give a comparison with DenseAlert, we compare the LDD and LDS on the first 5 datasets in Table 2. As we can see that EigenPulse has the lower LDD than DenseAlert except on Amazon Electronics dataset and has the lower LDS than DenseAlert. In detail, EigenPluse has the LDD which can be as small as 250 on source nodes on Yelp dataset and Amazon Cellphone dataset with the minimum density of 0.08 on sink nodes, which means we can detect fraudsters even if they use 250 accounts to create 20 × 20 edges across 30 days.

Table 2. Experimental results on real data with injected labels

Data Name	Metrics	DenseAlert	EigenPulse
BeerAdvocate	LDD	0.1	0.1
	LDS	13.33	**6.67**
Yelp	LDD	0.2	0.1
	LDS	26.67	13.33
Amazon Cellphone	LDD	0.2	**0.08**
	LDS	26.67	13.33
Amazon Electronics	LDD	**0.2**	1
	LDS	26.67	**6.67**
Amazon Grocery	LDD	0.2	**0.08**
	LDS	26.67	**6.67**

Besides, we injected 10 dense blocks with density vary from 0.01 to 1 for the Yelp dataset. The Fig. 2(b) shows the densities of all the windows on the EigenPulse. We can see that the injected dense blocks cause significant density surges. By assuming the density follows a Normal distribution, we successfully detect 6 injected blocks after simply set the density detection threshold as $\mu+3\sigma$, where μ and σ are the mean and standard deviation of all windows' density measures.

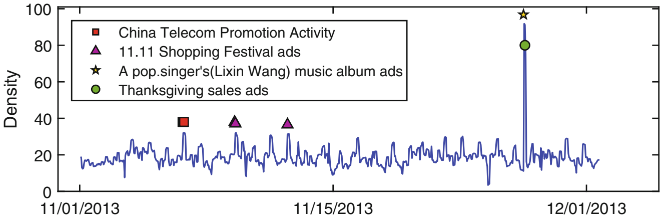


Fig. 3. EigenPulse detects anomalies dense blocks on Sina weibo dataset.

Anomaly Detection on Real Data: For the social network, i.e.,following relationship or message retweets, the dense blocks usually contain anomalous items or correspond to suspicious behaviors, and the sudden surges of density measure can be a significant signal for anomalies. So we applied the EigenPluse to Sina weibo dataset to detect the suspicious dense blocks, and also crawled the detailed content of msgs for verification.

The Fig. 3 illustrates the density change of dense blocks in the sliding windows with $w = 2$ hours, $s = 1$ hour. The Table 3 reports the suspicious features and content of detected blocks. We spot some significant spikes in the Fig. 3, and the message content all gives the tell-tale sign of suspicious behaviors, that is, as the

'Message Topic' shown, most of the messages about advertisements or products promotion information. In particular, We can notice that there are **953** edges for the only 7 users × 8 messages in two hours, which means a user retweeted 20 times for one message in average, and it's very suspicious intuitively. In summary, EigenPluse can detect anomalies dense blocks in real dataset.

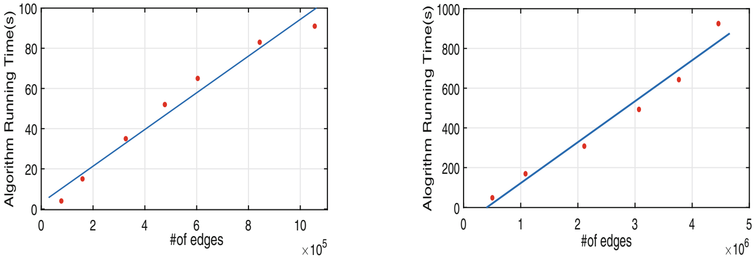
Table 3. Dense blocks detected by EigenPulse on Sina weibo dataset

Message topic	Size	Time range	#Edges
China Telecom Promotion Activity	39 × 57	6:00–8:00, Nov 7	2,004
	78 × 58	7:00–9:00, Nov 7	4,051
	151 × 119	8:00–10:00, Nov 7	8,295
11.11 Shopping Festival ads	201 × 139	6:00–8:00, Nov 10	7,012
	196 × 111	7:00–9:00, Nov 10	9,668
	126 × 93	8:00–10:00, Nov 13	638
A pop. singer's (Lixin Wang) music album ads	**7** × **8**	22:00–24:00, Nov 26	**953**
Thanksgiving sale ads	26 × 36	23:00, Nov 26–1:00, Nov 27	629
	43 × 34	1:00–3:00, Nov 27	263

4.4 Q3.Scalability

We demonstrate the linearly scalability with of EigenPluse by measuring how rapidly its update time increases as a tensor grows.

We choose two representative datasets: BeerAdvocate with the highest volume density, and Amazon Electronics with the most edges, and randomly sample sub-tensors with different size of edges. As shown in Fig. 4, the running time of our algorithm increases linearly with the number of the edges.



(a) **BeerAdvocate dataset** (b) **Amazon Electronic dataset**

Fig. 4. EigenPulse runs in near-linear time.

5 Conclusion

In this paper, we proposed a surge detection method, EigenPulse, which can spot the density surge in a large streaming graph in a efficient and accurate way. We use row-augmented matrix and Sliding Window to model streaming graph and design the AugSVD algorithm for efficient singular decomposition which is the input of EigenPulse. In conclusion, our algorithm has the following advantages:

- **Incremental singular value decomposition:** we propose a scalable algorithm, AugSVD,which combines Sliding Window to do streaming graph decomposition.
- **Robust and effective:** AugSVD has good robustness and EigenPulse generated by AugSVD can detect suspicious synchronized activities accurately.
- **Scalable:** EigenPulse is near-linear in running time and memory-efficient because it only detects one window's data each time.
- **Reproducibility**: The code and data are available at https://github.com/shenghua-liu/EigenPulse/invitations.

Acknowledgments. This material is based upon work supported by the Strategic Priority Research Program of CAS (XDA19020400), NSF of China (61772498, 61872206, 61425016, 91746301), and the Beijing NSF (4172059).

References

1. Akoglu, L., Tong, H., Koutra, D.: Graph based anomaly detection and description: a survey. Data Min. Knowl. Discov. **29**(3), 626–688 (2015)
2. Eswaran, D., Faloutsos, C., Guha, S., Mishra, N.: Spotlight: detecting anomalies in streaming graphs. In: SIGKDD, pp. 1378–1386. ACM (2018)
3. Gao, J., Zhou, C., Yu, J.X.: Toward continuous pattern detection over evolving large graph with snapshot isolation. In: VLDB (2016)
4. Guha, S., Mishra, N., Roy, G., Schrijvers, O.: Robust random cut forest based anomaly detection on streams. In: ICML (2016)
5. Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**, 217–288 (2011)
6. Hooi, B., Song, H.A., Beutel, A., Shah, N., Shin, K., Faloutsos, C.: Fraudar: bounding graph fraud in the face of camouflage. In: KDD. ACM (2016)
7. Jiang, M., Beutel, A., Cui, P., Hooi, B., Yang, S., Faloutsos, C.: A general suspiciousness metric for dense blocks in multimodal data. In: ICDM. IEEE (2015)
8. Koutra, D., Shah, N., Vogelstein, J.T., Gallagher, B., Faloutsos, C.: Deltacon: principled massive-graph similarity function with attribution. ACM Trans. Knowl. Discov. Data (TKDD) **10**, 28 (2016)
9. Liu, S., Hooi, B., Faloutsos, C.: Holoscope: topology-and-spike aware fraud detection. In: CIKM, pp. 1539–1548. ACM (2017)
10. Prakash, B.A., Sridharan, A., Seshadri, M., Machiraju, S., Faloutsos, C.: Eigen-Spokes: surprising patterns and scalable community chipping in large graphs. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010. LNCS (LNAI), vol. 6119, pp. 435–448. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13672-6_42

11. Shin, K., Hooi, B., Faloutsos, C.: M-Zoom: fast dense-block detection in tensors with quality guarantees. In: Frasconi, P., Landwehr, N., Manco, G., Vreeken, J. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9851, pp. 264–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46128-1_17
12. Shin, K., Hooi, B., Kim, J., Faloutsos., C.: D-cube: dense-block detection in terabyte-scale tensors. In: WSDM (2017)
13. Shin, K., Hooi, B., Kim, J., Faloutsos, C.: Densealert: incremental dense-subtensor detection in tensor streams. In: KDD. ACM (2017)
14. Sun, J., Tao, D., Faloutsos, C.: Beyond streams and graphs: dynamic tensor analysis. In: KDD. ACM (2006)
15. Yongsub Lim, M.J., Kang, U.: Memory-efficient and accurate sampling for counting local triangles in graph streams: from simple to multigraphs. In: TKDD. ACM (2018)
16. Yu, W., Gu, Y., Li, J., Liu, S., Li, Y.: Single-pass PCA of large high-dimensional data. In: IJCAI, pp. 3350–3356 (2017)