ONLINE COMPUTATION OF REDUCED EGONET FEATURES FOR ANOMALY DETECTION IN BANK TRANSACTIONS GRAPHS

Cristian-Enache Zica, Bogdan Dumitrescu

University Politehnica of Bucharest
Department of Automatic Control and Computers
313 Spl. Independenţei, 060042 Bucharest, Romania

ABSTRACT

Anomaly detection in bank transactions graphs can be an effective tool for revealing potentially fraudulent activities. Online algorithms are the most interesting in this context, since they can lead to early analysis of dubious transactions. We propose an online algorithm for computing vertex features related to egonet and reduced egonet. Such features have been proved effective for anomaly detection in bank transactions graphs, but in a batch approach. The proposed online algorithm needs to explore only sets of common neighbors of two vertices, without isolating the egonet as a subgraph. Hence, it is very efficient and has execution times suited for practical use. Also, the performance in detecting anomalies is quite similar to that of an a posteriori algorithm that computes the features and the anomaly scores only at the end of each day.

Index Terms— Graphs, egonet, anomaly detection, bank transactions, online algorithm

1. INTRODUCTION

Fraudulent activities like money laundering can seriously hinder the economic system of a country or region. Their early detection with machine learning tools is beneficial for the sanity of the system. Our focus here is on the problem of finding money laundering in bank transactions, which are still used extensively by some individuals and firms for this purpose.

Bank transactions can be naturally modeled with a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. A vertex $v \in \mathcal{V}$ is a bank account, or a group of accounts belonging to the same person or entity. An edge $e = (v_1, v_2) \in \mathcal{E}$ represents money transfers from v_1 to v_2. We associate two attributes with the edge: the cumulated amount of the transfers (converted to the same currency) and the number of transactions over a given time window. So, the graph is simple: there is at most one edge from one vertex to another; the reverse edge may exist or not.

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS - UEFISCDI, project number PN-III-P4-PCE-2021-0154, within PNCDI III.

Detecting abnormal vertices in such a graph, with graph anomaly detectors, can seriously narrow the group of fraud suspects. In this paper, we present an online version of a method with good results [1].

Previous work. The inspiration for some of the existing methods is in the detection of structures near those specific to money laundering, among which most prominent are cliques, stars and rings, but also more complicated structures like tripartite graphs [2]. Other methods use a statistical approach, identifying the extreme points in the distribution of some features, related to egonets [3], local connectivity [4], or several categories [5, 6]. The reduced egonet is used in [1] as a structure that can provide relevant features, fed to an anomaly detector. There are also many learning approaches for anomaly detection in graphs: node2vec [7] (automated features extraction), Netwalk [8], auto-encoders [9, 10, 11] (the first with application to financial fraud detection). More references can be found in the review articles [12, 13, 14], the latter on methods using deep learning.

Contribution and contents. We have two main purposes. The first and main goal is to design and implement an efficient online algorithm for exactly computing the reduced egonet features without computing egonets. The features are reviewed in Section 2. The algorithm is derived and described in Section 3. The second is to check whether the online algorithm has similar anomaly detection performance as the batch a posteriori algorithm, in a scenario that will be detailed later in Section 4. In the same section, we present experimental evidence on the speed-up of our online algorithm and on its detection performance.

2. REDUCED EGONET FEATURES

We review here some relevant features related to the egonet structure. The egonet $E(v)$ of (central) vertex v is the subgraph of $\mathcal{G}$ whose vertices are v and its immediate neighbors, regardless of the directions of the edges between them.

The reduced egonet (egored) [1] $\hat{E}(v)$ is the subgraph of $E(v)$ that is obtained after removing all vertices that have a single edge (in or out). Figure 1 shows five simple egonets,

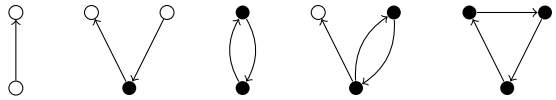


Fig. 1. Simple egonets and their egoreds. The bottom node is the center. Egoreds are made of the filled vertices.

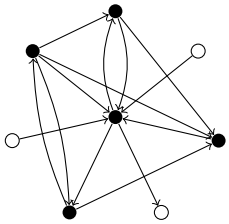


Fig. 2. An egonet. Its egored is the subgraph having the filled circles as vertices.

whose central vertex is the bottom one; the corresponding egoreds are the subgraphs having the filled circles as vertices; from left to right, the number of vertices in the egoreds are: 0, 1, 2, 2, 3. Another egonet is shown in Figure 2; it has eight vertices; its egored has five vertices and is a near-clique, which is a structure that may indicate a suspect behavior.

2.1. Basic features

The approach in [1] uses the following eight basic features of a vertex:

- in/out (I/O) degrees, denoted d_i and d_o
- I/O total amounts, a_i and a_o
- I/O number of transactions, t_i and t_o
- number of vertices in the egonet, n_v
- number of edges in the egonet (viewed as an undirected graph), n_e

For example, the vertex whose egonet is shown in Figure 2 has the indegree $d_i = 5$ and the outdegree $d_o = 3$; the number of vertices in the egonet is $n_v = 8$ and the number of edges is $n_e = 12$. The input total amount is the sum of all transfers to the central vertex from its neighbors; the input number of transactions is simply the number of such transfer operations. The output features are defined similarly.

The same basic features as above can be computed for the egored. We use the same notation as for the egonet, but with a hat. In Figure 2, considering now the egored, the central vertex has the indegree $\hat{d}_i = 3$ and the outdegree $\hat{d}_o = 2$; the number of vertices in the egored is $\hat{n}_v = 5$ and the number of edges is $\hat{n}_e = 9$. Since the egored is a subgraph of the egonet, the total amounts and the number of transactions are smaller: $\hat{a}_i \leq a_i$, $\hat{t}_i \leq t_i$, etc. When the vertex whose features are referred to is not clear, we use the notation, e.g., $d_i(v)$.

2.2. Features for anomaly detection

Anomaly detection was performed in [1] using features selected or derived from the basic ones. For the sake of completeness, we enumerate the features that proved the most successful for detecting anomalies in real and synthetic graphs.

There are seven egonet features:

- I/O degree, d_i and d_o
- I/O total amounts
- I/O average amount, a_i/t_i and a_o/t_o, total amount of money I/O divided by the number of I/O transactions
- egonet edge density, n_e/n_v, the ratio between the number of edges of the egonet and the number of its nodes.

The reduced egonet features are (also seven):

- I/O relative degree, $\hat{d}_i/d_i$ and $\hat{d}_o/d_o$, the ratios between degrees in the egored and degrees
- I/O relative total amount, $\hat{a}_i/a_i$ and $\hat{a}_o/a_o$, the ratios between the I/O total amount in the egored and the I/O total amount
- I/O average amount, $\hat{a}_i/\hat{t}_i$ and $\hat{a}_o/\hat{t}_o$
- egored edge density, $\hat{n}_e/\hat{n}_v$.

For all features that are a ratio, the value is set at zero if the denominator is zero. The computation of the above features is direct and very simple when the basic features are available. So, our focus in the next section will be on the online computation of the basic features.

3. ONLINE FEATURE UPDATE

We assume that we are in possession of a graph $\mathcal{G}$ and the basic features described in section 2.1 are available for all vertices. A new transaction appears, from v_1 to v_2, with amount a. Our problem is to update the basic features such that the new transactions is considered. In other words, we want to design an online algorithm for basic features update.

The inefficient algorithm is obvious. We simply recompute the egonets and egoreds of all vertices affected by the new transaction and extract their basic features. This algorithm may be good for vertices with low connectivity, but becomes slow for highly connected vertices. Since in an online algorithm we want a small execution time not only in average, but also in the worst case, this solution is not acceptable.

Let $C(u, w)$ be the set of common neighbors (no matter the directions of the edges) of the vertices u and w. We will describe an algorithm for basic feature update that computes only sets of common neighbors. No subgraphs, like the egonet or the egored, are necessary. Since a formal description of the algorithm would take too much space (and it can

be derived from the sources that we provide), we present here the main situations that appear and their treatment, with more or less details. In all considerations, the graph $\mathcal{G}$ is that before the arrival of the new transactions.

3.1. Simple cases

We discuss first the cases whose solution is immediate.

If the edge (v_1, v_2) already exists, so this is not the first transactions between the two vertices, then the egonets $E(v_1)$ and $E(v_2)$ are unchanged. Only the amounts and number of transactions must be modified via

$$\begin{array}{ll} a_o(v_1) \mathrel{+}= a, & a_i(v_2) \mathrel{+}= a, \\ {+}{+}t_o(v_1), & {+}{+}t_i(v_2). \end{array} \tag{1}$$

To shorten the relations, we use the standard C operators $++$ to increase the value by 1 and $+=$ to add the right hand value to the left hand variable.

Next, we have to examine whether $v_1 \in \hat{E}(v_2)$ and $v_2 \in \hat{E}(v_1)$. Note that both events are either true or false: the relationship is reciprocal; if v_1 is in the egored of v_2, then necessarily v_2 is in the egored of v_1. The condition for v_1 and v_2 to be in each other's egored is

$$C(v_1, v_2) \neq \emptyset \text{ or } (v_2, v_1) \in \mathcal{E}. \tag{2}$$

So, either v_1 and v_2 have a common neighbor or the inverse edge exists. (These situations are essentially those in the rightmost and middle graphs in Figure 1, respectively.) If (2) holds, then updates similar to those in (1) are applied for the egored amounts and number of transactions:

$$\begin{array}{ll} \hat{a}_o(v_1) \mathrel{+}= a, & \hat{a}_i(v_2) \mathrel{+}= a, \\ {+}{+}\hat{t}_o(v_1), & {+}{+}\hat{t}_i(v_2). \end{array}$$

If the edge (v_1, v_2) is new, then a simple situation is that when one or both vertices are new. The egored of the new vertex is empty (see leftmost case in Figure 1) and the egored of the existing vertex stays in general unchanged. For example, if v_1 exists but not v_2, then $\hat{E}(v_1)$ is unchanged unless v_1 had a single neighbor, in which case v_1 enters its own egonet, formerly empty (see, e.g., second graph in Figure 1; compare with second rightmost, where the egored is not modified). Also, $E(v_1)$ suffers trivial modifications:

$$+{+}d_o(v_1),\ +{+}n_v(v_1),\ +{+}n_e(v_1).$$

The case where v_2 exists but not v_1 is similar. The case where both v_1 and v_2 are new is trivial; the basic features of a new node should be obvious.

3.2. New edge between existing vertices

The interesting case is that of a new edge between existing vertices, because in this case the egonet always changes and the egored may or may not change. Moreover, changes in the egonets and egoreds of other vertices may appear. In all cases, amounts and number of transactions are modified as in (1).

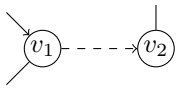
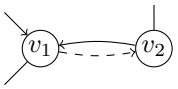
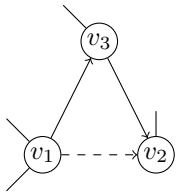
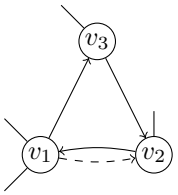
	$(v_2, v_1) \notin \mathcal{E}$	$(v_2, v_1) \in \mathcal{E}$
$C(v_1, v_2) = \emptyset$	A 	B 
$C(v_1, v_2) \neq \emptyset$	C 	D 

Table 1. Elementary cases; the new edge is dashed.

Relation (2) suggests which are the main conditions that can be used to explore the possible cases. Table 1 illustrate the four elementary cases, denoted A, B, C, D; the conditions for their definition are: i) whether the inverse edge (v_2, v_1) exists (B, D) or not (A, C); ii) whether v_1 and v_2 have common neighbors (C, D) or not (A, B).

We discuss first the direct effect of the edge (v_1, v_2) on the basic features of v_1 and v_2. After that, we will see the effects on the common neighbors of these vertices.

Egonet direct changes. Since the edge is new, the (egonet) degrees always grow:

$$+{+}d_o(v_1),\ +{+}d_i(v_2). \tag{3}$$

Also, in cases A and C, the edge (v_1, v_2) adds a vertex and a connection to each egonet $E(v_1)$, $E(v_2)$:

$$\begin{array}{ll} {+}{+}n_v(v_1), & {+}{+}n_v(v_2), \\ {+}{+}n_e(v_1), & {+}{+}n_e(v_2). \end{array} \tag{4}$$

Egored direct changes. In case A, the egoreds do not change. The new edge is the only path from v_1 to v_2.

In cases B, C, D, the degrees grow, similarly to (3):

$$+{+}\hat{d}_o(v_1),\ +{+}\hat{d}_i(v_2). \tag{5}$$

Furthermore, in cases B, C, each vertex enters the egonet of the other and so, similarly to (4), we update

$$\begin{array}{ll} {+}{+}\hat{n}_v(v_1), & {+}{+}\hat{n}_v(v_2), \\ {+}{+}\hat{n}_e(v_1), & {+}{+}\hat{n}_e(v_2). \end{array} \tag{6}$$

Finally, in case B, the inverse edge (v_2, v_1) contributes now to the egored

$$\begin{array}{ll} \hat{a}_i(v_1) \mathrel{+}= a(v_2, v_1), & \hat{a}_o(v_2) \mathrel{+}= a(v_2, v_1), \\ \hat{t}_i(v_1) \mathrel{+}= t(v_2, v_1), & \hat{t}_o(v_2) \mathrel{+}= t(v_2, v_1), \end{array} \tag{7}$$

where $a(v_2, v_1)$ is the amount transferred from v_2 to v_1 and $t(v_2, v_1)$ is the corresponding number of transactions; this information can be retrieved from $\mathcal{G}$.

Indirect changes. We go now to changes caused to vertices that are common neighbors of v_1 and v_2, which may have also side effects on v_1 and v_2. We describe below the updates for an arbitrary $v_3 \in C(v_1, v_2)$. For illustration, we use the directions of the edges from Figure 1, i.e., (v_1, v_3) and (v_3, v_2); a complete algorithm would take into account all possible directions.

We note that in case D, v_1 and v_2 were already in $\hat{E}(v_3)$. So, only case C is furthermore concerned. Each of v_1, v_2, v_3 joins the egonets and egoreds of the two other vertices, if not already there. First of all, each egonet and egored receives a new edge: (v_1, v_2) joins $E(v_3)$ and $\hat{E}(v_3)$; (v_1, v_3) joins $E(v_2)$ and $\hat{E}(v_2)$; (v_3, v_2) joins $E(v_1)$ and $\hat{E}(v_1)$. The corresponding updates are

$$\begin{array}{lll} ++n_e(v_1), & ++n_e(v_2), & ++n_e(v_3) \\ ++\hat{n}_e(v_1), & ++\hat{n}_e(v_2), & ++\hat{n}_e(v_3). \end{array} \tag{8}$$

The condition that v_1 and v_3 join the egoreds of each other is that $C(v_1, v_3) = \emptyset$ (no common neighbors exist) and $(v_3, v_1) \notin \mathcal{E}$; otherwise said, v_2 becomes their first common neighbor. If so, the effects are

$$\begin{array}{ll} ++\hat{n}_v(v_1), & ++\hat{n}_v(v_3), \\ ++\hat{n}_e(v_1), & ++\hat{n}_e(v_3). \end{array} \tag{9}$$

Moreover, under the same condition, the edge (v_1, v_3) is new to $\hat{E}(v_1)$ and $\hat{E}(v_3)$, which produces the updates (see (7))

$$\begin{array}{ll} ++\hat{d}_o(v_1), & ++\hat{d}_i(v_3), \\ \hat{a}_o(v_1) \mathrel{+}= a(v_1, v_3), & \hat{a}_i(v_3) \mathrel{+}= a(v_1, v_3), \\ \hat{t}_o(v_1) \mathrel{+}= t(v_1, v_3), & \hat{t}_i(v_3) \mathrel{+}= t(v_1, v_3). \end{array} \tag{10}$$

The pair v_2, v_3 is treated similarly.

Finally, we note that the inverse operation, that of downdating, which is necessary when a transaction is removed, can be easily implemented by following the same cases as above.

3.3. Complexity

The operations described in section 3.2 are gathered in Algorithm 1. We can safely evaluate the worst case complexity only for the case of a new edge between existing nodes. Steps 1–6 contain a constant number of operations. However, we need to compute $C(v_1, v_2)$, which has at most $\min(d_i(v_1) + d_o(v_1), d_i(v_2) + d_o(v_2))$ vertices. So, the operations performed for each v_3 in the loop 7–11 add up to a complexity that is at most linear in the degree of the node. Note that the evaluation of the conditions $C(v_1, v_3) = \emptyset$ and $C(v_2, v_3) = \emptyset$ is less demanding than computing the set of common neighbors; once a common neighbor is found, the search is finished.

By comparison, the full recomputation of the basic features for v_1, v_2 and all $v_3 \in C(v_1, v_2)$ also needs the exploration of all neighbors of v_1, v_2 and v_3, but it adds another level of complexity. All edges of the egonets and egoreds have to be explored and hence the number of operations may be quadratic in the degree, in the worst case. Also, the total amounts and numbers of transactions need to be computed from scratch, while in (1) a single addition is necessary.

Algorithm 1: Online egored basic features update for a new edge between existing nodes.

Data: graph $\mathcal{G}$ and associated basic features
vertices v_1, v_2, amount a
Result: updated basic features

1 Update amount and numbers of transactions with (1)
2 Update degrees with (3)
3 Case A. Update egonet number of vertices and edges: (4)
4 Case B. Update egored degrees: (5); update egored number of vertices and edges: (6); add contribution of inverse edge: (7)
5 Case C. Do (4); (5); (6)
6 Case D. Do (5)
7 **for** *all* $v_3 \in C(v_1, v_2)$*, only in case C* **do**
8 Update numbers of edges with (8)
9 **if** $C(v_1, v_3) = \emptyset$ *and* $(v_1, v_3) \notin \mathcal{E}$ **then**
10 Update number of vertices and edges: (9); add contribution of (v_1, v_3): (10)
11 Same as in steps 9–10 for (v_3, v_2)

4. EXPERIMENTAL RESULTS

We have implemented our algorithm in Python. The sources can be found at `http://asydil.upb.ro/software`. The tests were performed on a Macbook M1 Pro computer with an 8 core processor running at 2.06 - 3.22 GHz and 16 GB unified memory.

The dataset is a preprocessed and anonymized four-month long list of financial transactions between accounts provided by Libra Internet Bank from Romania. In order to enable online-like behavior, the four-month data were grouped in two parts: first part comprises the historical data[1] (three months) and second part contains the fourth month split into days. Thus, each day can be processed as an increment to the existing data. The historical dataset has 4558805 transactions and the incremental dataset has 1613005 transactions. The graph corresponding to the first three months has 385100 vertices and 597165 edges. The transactions were labeled by bank experts, based on a set of internal rules. Suspect transactions are named alerts. After further investigation, some of the alerts are sent to state institutions for a thorough analysis of their fraudulent character; these particular transactions

[1] `http://graphomaly.upb.ro/Date/Libra_bank_3months_graph.zip`

are called reports. In this paper we consider only alerts as anomalies. Each vertex is associated with weights equal to the number of anomalous transactions it is involved in.

We used Isolation Forest (IF) [15] in the PyOD [16] implementation for anomaly detection on the features listed in section 2; IF gave very good results in [1] and was preferred to other detectors. We fitted IF on the historical data, for which the transactions graph and all features were already available. The fitting produces anomaly scores, that are sorted decreasingly (the higher the score, the more abnormal the vertex). Note that the whole operation is unsupervised. The transactions of the first day were processed one by one with our proposed algorithm, each time updating the basic features. After the update, the IF score was computed for the updated features of all vertices that are modified. The position of the score in the sorted historical scores is called online ranking; for example a value of 0.01 means that 1% vertices have higher score and 99% have lower score. If the ranking is below a given threshold, an alert can be raised. At the end of the day, the transactions of the day are appended to the historical data, IF is refitted and new historical scores are computed. This modus operandi is meant to produce quick online decisions; the anomaly detector is fitted at night, when activity is low. The next day, the same procedure is applied.

We measured the time for online processing of each transaction over the fourth month. It includes the execution time for updating the basic features (Algorithm 1 in complete form), the computation of the features listed in Section 2.2 and the score calculation by IF. The distribution of the execution times is shown in Figure 3. It can be seen that, for most transactions, the time is between 0.07 and 0.11 seconds. The average time is 0.0803. The minimum time is 0.044, while the maximum is 0.55. There are only 84 transactions for which the computation time is greater than 0.15 seconds. Hence, the online algorithm is suitable for practical uses, at least in Libra bank.

Figure 4 shows a histogram of the ratio between the execution times of the full recomputation algorithm and of the proposed online algorithm. Due to the high complexity of full recomputation, we present only the results for the first day of the fourth month. The speed-up of the online algorithm is usually below 5, due to existence of many vertices with low connectivity. However, in the worst case, the speed-up can attain values over 300; there are two transactions for which the full recomputation algorithm needs about 29 seconds, while the online algorithm takes less than 0.1 seconds. Although the times for the first day are slightly lower, as seen in Figure 3, we can quite safely say that the ratios given in Figure 4 are representative, since the graph grows in size and both algorithms will have higher complexities; note also that the largest overall time (0.55) happens to appear in the first day.

To evaluate the quality of detection, we compare the online algorithm with an a posteriori algorithm, where the features are computed at the end of the day, IF is fitted on all

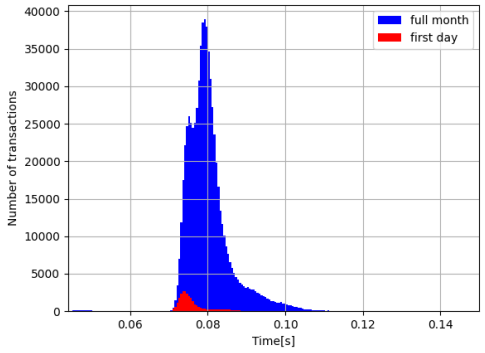


Fig. 3. Histogram of execution times of the online algorithm.

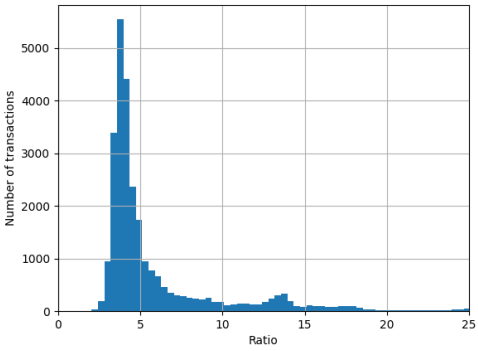


Fig. 4. Ratio of the execution times of full recomputation and online algorithms, for the first day.

available data (including the current day) and the scores are computed. Figure 5 shows a histogram of the differences Δ between the a posteriori ranking and the online ranking for the nodes that have become abnormal in the fourth month and were involved directly in transactions. A vertex can be involved across a day in multiple transactions, which results in multiple online rankings. When comparing to the a posteriori score, we use the smallest ranking, i.e., the most dubious snapshot of the vertex. A positive difference means that the online ranking is smaller, hence the vertex is more abnormal. The average value of $|\Delta|$ is $9.97 \cdot 10^{-4}$. So, in general, the difference between the a posteriori and online rankings is negligible; note that IF is not fully deterministic, hence such differences cannot be avoided. The mininum Δ is -0.0183, while the maximum is 0.0399. Only 12 cases out of 711 were observed to have a difference greater than 0.01 in absolute value. Moreover, whenever $|\Delta| > 0.01$, the ranking is in an area that is not in the usual anomaly range. For example, when $\Delta = 0.0399$, the a posteriori ranking is 0.074 and the online ranking is 0.034; when $\Delta = -0.0183$, the a posteriori ranking is 0.038 and the online ranking is 0.054. We

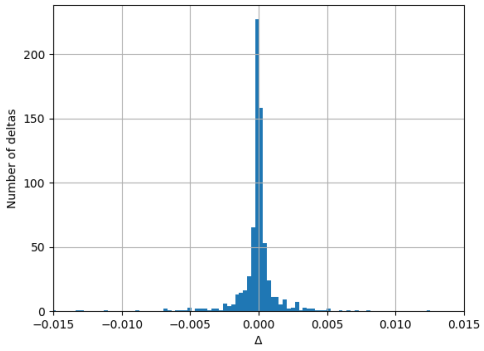


Fig. 5. Difference between a posteriori and online ranking.

conclude that the online algorithm keeps very close to the a posteriori nightly run; when the differences are large, they do not change the anomaly decision.

5. CONCLUSIONS

We have presented an online algorithm for updating the egonet and reduced egonet features, with application to a graph of bank transactions. Although not yet implemented to achieve maximum speed, the algorithm is fast enough for real time use and much faster than the full recomputation of the features of the updated graph. For a real dataset of transactions, the proposed online procedure has similar performance with a batch a posteriori algorithm that refits the anomaly detector daily.

6. REFERENCES

[1] B. Dumitrescu, A. Bălţoiu, and Ş. Budulan, "Anomaly detection in graphs of bank transactions for anti money laundering applications," *IEEE Access*, vol. 10, pp. 47699–47714, 2022.

[2] X. Li, S. Liu, Z. Li, X. Han, C. Shi, B. Hooi, H. Huang, and X. Cheng, "Flowscope: Spotting money laundering based on graphs," in *Proc. AAAI Conference on Artificial Intelligence*, 2020, vol. 34, pp. 4731–4738.

[3] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2010, pp. 410–421.

[4] I. Molloy, S. Chari, U. Finkler, M. Wiggerman, C. Jonker, T. Habeck, Y. Park, F. Jordens, and R. van Schaik, "Graph analytics for real-time scoring of cross-channel transactional fraud," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 22–40.

[5] A.E. Wegner, L. Ospina-Forero, R.E. Gaunt, C.M. Deane, and G. Reinert, "Identifying networks with common organizational principles," *Journal of Complex Networks*, vol. 6, no. 6, pp. 887–913, 2018.

[6] A. Elliott, M. Cucuringu, M.M. Luaces, P. Reidy, and G. Reinert, "Anomaly detection in networks with application to financial transaction networks," *arXiv:1901.00402*, 2019.

[7] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

[8] W. Yu, W. Cheng, C.C. Aggarwal, K. Zhang, H. Chen, and Wei Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, 2018, pp. 2672–2681.

[9] A.M. Mubalaike and E. Adali, "Deep learning approach for intelligent financial fraud detection system," in *3rd Int. Conf. Computer Science and Engineering (UBMK)*, 2018, pp. 598–603.

[10] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proc. Int. Conf. Data Mining*. SIAM, 2019, pp. 594–602.

[11] S. Bandyopadhyay, S.V. Vivek, and M.N. Murty, "Outlier resistant unsupervised deep architectures for attributed network embedding," in *Proc. 13th Int. Conf. Web Search and Data Mining*, 2020, pp. 25–33.

[12] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[13] W. Hilal, S.A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert Systems with Applications*, vol. 193, pp. 116429, 2022.

[14] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q.Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Trans. Knowledge and Data Engineering*, 2021.

[15] F. T. Liu, K.M. Ting, and Z.-H. Zhou, "Isolation forest," in *8th IEEE Int. Conf. Data Mining*. IEEE, 2008, pp. 413–422.

[16] Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A Python Toolbox for Scalable Outlier Detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019.