ABSTRACT

The problem of online fraud detection can often be formulated as mining a bipartite graph of users and objects for suspicious patterns. The edges in the bipartite graph represent the interactions between users and objects (e.g., reviewing or following). However, smart fraudsters use sophisticated strategies to influence the ranking algorithms used by existing methods. Based on these considerations, we propose FraudTrap, a fraud detection system that addresses the problem from a new angle. Unlike existing solutions, FraudTrap works on the object similarity graph (OSG) inferred from the original bipartite graph. The approach has several advantages. First, it effectively captures the loosely synchronized behavior in face of different types of camouflage. Second, it has two operating modes: unsupervised mode and semi-supervised mode, which are naturally incorporated when partially labeled data is available to further improve the performance. Third, all algorithms we design have the near-linear time complexities and apply on large scale real-world datasets. Aiming at each characteristic of FraudTrap, we design corresponding experiments that show FraudTrap outperforms other state-of-the-art methods on eleven real-world datasets.

KEYWORDS

Fraud Detection, Object Similarity Graph, Graph Partition

1 INTRODUCTION

Fraud has severely detrimental impacts on the business of social networks and other web online applications [26]. A user can become a fake celebrity by purchasing “zombie followers” on Twitter. A merchant can boost his reputation through fake reviews on Amazon. This phenomenon also conspicuously exists on Facebook, Yelp and TripAdvisor, etc. In all the cases, fraudsters try to manipulate the platform’s ranking mechanism by faking interactions between the fake accounts they control (fraud users) and the target customers (fraud objects).

These scenarios are often formulated as a bipartite graph of objects and users. We define an object as the target a user could interact with on a platform. Depending on the application, an object can be a followee, a product or a page. An edge corresponds to the interaction from a user to the object (e.g., reviewing or following). Detecting fraud in the bipartite graph has been explored by many methods. Since fraudsters rely on fraudulent user accounts, which are often limited in number, to create fraudulent edges for fraud objects’ gain [16], previous methods are mainly based on two observations: (1) fraud groups tend to form dense subgraphs in the bipartite graph (high-density signal), and/or (2) the subgraphs induced by fraud groups have unusually surprising connectivity structure (structure signal). These methods mine the bipartite graph directly for dense subgraphs or rare structure patterns. Their performances vary in real-world datasets.

Unfortunately, smart fraudsters use more sophisticated strategies to avoid such patterns. First, by multiplexing a larger pool of fraud users, a fraudster can effectively reduce the density of the subgraph induced by a fraud group. This is called loosely synchronized behavior and leads to the limited performance of the methods [13, 15, 28, 30, 35] depending on the high-density signal. Another commonly used technique is to create edges pointing to normal objects to disguise fraud users as normal ones. This strategy, often called camouflage, alters the connectivity structure of the bipartite graph and weakens the effectiveness of many approaches targeting such structure, such as HITS [16, 41], and belief propagation (BP) [1, 28]. Fig. 1 illustrates these two strategies.

The problem of fraud detection can also be handled using supervised or semi-supervised approaches when (partially) labeled data are available. [7, 42] provide better performance using a subset of labeled frauds. [8, 14, 20] build machine learning classifiers to detect anomalies. These approaches, however, have a number of limitations. Firstly, it is often very difficult to obtain enough labeled data in fraud detection due to the scale of the problem and the cost of investigation. Secondly, they require great effort in feature engineering which is tedious and demands high expertise level. Thirdly, they often fail to detect new fraud patterns. Finally, even though some labeled data can provide potentially valuable information for
In this paper, we propose FraudTrap, a graph-based fraud detection algorithm that overcomes these limitations with a novel change of the target of analysis. Instead of mining the bipartite graph directly, FraudTrap analyzes the Object Similarity Graph (OSG) that is derived from the original bipartite graph. There are two main advantages of our design: (1) fraud objects exhibit more similar behavior patterns since fraud objects are difficult to gain edges from normal users, which endows FraudTrap with inherent camouflage-resistance (Sec. 4.4); (2) since the number of objects is typically smaller than the number of users [18], working with OSG reduces computation cost while guaranteeing the effectiveness. In addition, although FraudTrap works well without any labels, we can easily switch to a semi-supervised mode and improve performance with partial labels.

In summary, our main contributions include:

1) [Metric C]. We build Object Similarity Graph (OSG) by a novel similarity metric, C-score, which transforms the sparse subgraphs induced by fraud groups in the bipartite graph into the much denser subgraphs in OSG, by merging information from unlabeled and labeled (if available) data.

2) [Algorithm LPA-TK]. We propose a similarity-based clustering algorithm, LPA-TK, that perfectly fits in OSG and outperforms the baseline (LPA) in face of noise edges (camouflage).

3) [Metric F]. Given candidate groups returned by C + LPA-TK, we propose an interpretable suspiciousness metric, F-score, meeting the all basic “axioms” proposed in [15].

4) [Effectiveness]. Our method FraudTrap (C + LPA-TK + F) can operate in two modes: unsupervised and semi-supervised. The unsupervised mode outperforms other state-of-the-art methods for catching synchronized behavior in face of camouflage. Semi-supervised mode naturally takes advantage of partially labeled data to further improve the performance.

2 RELATED WORK

To maximize their financial gains, fraudsters have to share or multiplex certain resources (e.g., phone numbers, devices). To achieve the “economy of scale”, fraudsters often use many fraudulent user accounts to conduct the same fraud [3, 5]. As a result, fraud users inevitably exhibit synchronized behavior on certain features, be it phone prefixes, or IP subnets. Group-based approaches that detect frauds by identifying such synchrony are surpassing content-based methods as the most effective anti-fraud solutions. There are three classes of methods.

Unsupervised. Unsupervised methods achieve various performance on fraud detection. There are two types of unsupervised detection methods in the literature.

The first type is based on high-density subgraphs formed by fraud groups. Mining dense subgraphs in the bipartite graph [30, 35, 37] is effective to detect the fraud group of users and objects connected by a massive number of edges. Fraudar [13] tries to find a subgraph with the maximal average degree using a greedy algorithm. CrossSpot [15] focuses on detecting dense blocks in a multi-dimensional tensor and gives several basic axioms that a suspiciousness metric should meet. People have also adopted singular-value decomposition (SVD) to capture abnormal dense user blocks [17, 33]. However, fraudsters can easily evade detection by reducing the synchrony in their actions (details in Sec. 3).

The second type is based on rare subgraph structures of fraud groups. Such structures may include the sudden creation of massive edges to an object, etc. BP [1, 28] and HITS [9, 10, 16] intend to catch such signals in the bipartite graph. FraudEagle [1] uses the loopy belief propagation to assign labels to the nodes in the network represented by Markov Random Field (MRF). [34] ranks abnormality of nodes based on the edge-attributed behavior pattern by leveraging minimum description length. [12, 19] use Bayesian approaches to address the rating-fraud problem. SynchroTrap [5] works on the user similarity graph. In all the cases, it is relatively easy for fraudsters to manipulate edges from fraud users to conceal such structural patterns (details in Sec. 3). The common requirement of parameter tuning is also problematic in practice, as the distribution of fraudsters changes often.

(Semi-)supervised. When partially labeled data are available, semi-supervised methods can be applied to anomaly detection. The fundamental idea is to use the graph structure to propagate known information to unknown nodes. [11, 22] model graphs as MRFs and label the potential suspiciousness of each node with BP. [4, 7, 24] use the random walk to detect Sybils. ADOA [42] clusters observed anomalies into k clusters and classifies unlabeled data into these k clusters according to both the isolation degree and similarity. When adequate labeled data are available, people have shown success with classifiers such as multi-kernel learning [14], support vector machines [39] and k-nearest neighbor [38]. However, it is rare to have enough fraud labels in practice.

3 DESIGN CONSIDERATIONS

We provide details why fraudsters can easily evade existing detection, and present the key ideas of FraudTrap design.

3.1 How Smart Fraudsters Evade Detection?

Reducing synchrony in fraud activities. One of the key signals that existing fraud detection methods rely on is the high-density of a subgraph. A naive fraud campaign may reuse some of the resources such as accounts or phone numbers, resulting in high-density subgraphs. However, experience shows that fraudsters now control larger resource pools and thus can adopt smarter strategies to reduce the synchrony by rotating the fraud users each time. For example, [16] reports that on Weibo a fraud campaign uses 3 million fraud accounts, a.k.a. zombie fans, to follow only 20 followees (fraud objects). Each followee gains edges from a different subset of the

| Table 1: FraudTrap v.s. existing methods |
|-----------------------------------------|
| FraudTrap [1] | Speakert [36] | Copy-Catch [3] | CopyCatcher [36] | CrossSpot [15] | Fraud [30] | FraudEagle [1] | M-zoom [36] | FraudTrap |
| Loosely synchronized? | × | × | × | × | × | × | × | × | × |
| Camouflage-resistant? | √ | × | × | ? | × | × | × | × | √ |
| Side information? | × | √ | × | × | × | × | √ | × | × |
| Semi-supervised? | × | × | × | × | × | × | × | × | × |
3.2 Our Key Ideas

The fundamental reason that the above two strategies succeed in deceiving existing detection methods is that they are based on analyzing the original bipartite graph. The fraudsters can easily manipulate the graph (both the density and structure) with a large number of fraud users. Unfortunately, the current black market makes the number of fraud accounts easy to obtain.

We propose to solve the problem from a different angle. Objects that pay for fraud activities are similar because the fraudsters must use their fraud user pool to serve many objects to make profits. Thus, instead of analyzing the user-object bipartite graph directly, we work on the similarity among different objects, which we capture as an object similarity graph (OSG) whose nodes are all objects and the edges represent the similarity among these objects. As we will show, with a carefully designed similarity score, a fraud object is much more similar with other fraud objects than normal ones and it is much harder for fraudsters to manipulate the OSG than the original bipartite graph. This is because, in the OSG, the subgraph $G$ formed by loosely synchronized behavior is much denser compared to the corresponding subgraph in the original user-object bipartite graph and the density of $G$ cannot be altered by camouflage. Figure 3 shows an intuitive example.

Furthermore, we want to leverage the side information available in different applications instead of letting the algorithm limit the choices. Specifically, we allow optionally including two types of information, (partial) fraud labels to offer a semi-supervised mode for the algorithm and side information of the activities, such as timestamp and star rating, etc. As we will show, the similarity score we design is additive for both labels and extra dimensions, so it is easy to incorporate all available information into the uniform framework.

4 METHODS

In this section, we first provide an overview of the workflow (Figure 2), and then we detail each of the three steps of the OSG construction (C), clustering on OSG (LPA-TK), and spot suspicious groups (F). Finally, we provide the intuitions and proofs.

Problem definition and workflow. Consider a bipartite graph $G$ of a user set $N$ and an object set $M$, and another bipartite graph $G'$ formed by a subset of labeled fraud users $N'$ and the same object set $M$. We use an edge $e$ pointing from a user to an object to represent an interaction between them, be it a follow, comment or purchase.

FraudTrap works in three stages:

1. (1) OSG construction: The OSG captures the object similarity, and we design a metric, $C$-score, to capture the similarity between two objects based on user interactions. If $G'$ is available, i.e., there are some labeled data, the $C$-score incorporates that data too.

2. (2) Clustering on OSG: We propose a similarity-based clustering algorithm that clusters each object into a group based on its most similar neighbors on OSG.

3. (3) Spot suspicious groups: Given candidate groups, it is important to use an interpretable metric to capture how suspicious an object/user group is, relative to other groups. We design the $F$-score metric for the purpose.

We elaborate these three stages in the rest of this section.

4.1 Stage I: OSG Construction (C)

OSG captures the similarity between object pairs, and thus the first step is to define the similarity metric, $C$-score. The $C$-score has two parts, similarity in $G$ (unlabeled) and in $G'$ (labeled). Formally, we define the similarity score $C_{ij}$ between object $m_i$ and object $m_j$ as

$$C_{ij} = S_{ij} + S'_{ij},$$

(1)
where $S_{ij}$ is the similarity score calculated from the unlabeled $G$, while $S'_{ij}$ is obtained from the labeled $G'$. In $G$, let $I = \{\varepsilon_{ij} : n_j \in N, (n_j, m_j) \in E\}$ be the set of edges pointing to $m_i$. Following the definition of the Jaccard similarity \[29\], we define the similarity between $m_i$ and $m_j$, $S_{ij}$, as

\[ S_{ij} = \frac{|I_i \cap I_j|}{|I_i \cup I_j|}. \]

(2)

In $G'$, let $I'_i = \{\varepsilon_{ij} : n_j \in N', (n_j, m_j) \in E'\}$ represent the set of edges pointing to $m_i$. Then the similarity score $S'_{ij}$ between $m_i$ and $m_j$ is given by:

\[ S'_{ij} = \frac{|I'_i \cap I'_j|}{mean(I')} \]

(3)

where $mean(I')$ is the mean of the set $I' = \{|I'_i \cap I'_j| : m_i, m_j \in M, |I'_i \cap I'_j| > 0\}$. **Leveraging side information.** If the side information describing additional properties of the user-object interaction is available, we want to include the information in the detection. For example, \[3\] reports that the time feature is essential for fraud detection. To do so, we can augment an edge $\varepsilon_{ij}$ both in $G$ and $G'$ using the following attribute tuple:

\[ \varepsilon_{ij} = (n_i, \text{Attr}_1, \text{Attr}_2, \cdots), \]

where $\text{Attr}_1$ can be a timestamp, star-rating, etc. We can append as many attributes as we need into the tuple and combine the synchronized behavior into a single score $C$. We give the following simple example.

**Example 1:** In a collection of reviews on Amazon, a review action $(n_i, m_j, \text{time}_1, \text{IP}_1)$ indicates that a user $n_i$ reviewed product $m_j$ at the time $\text{time}_1$ on $\text{IP}_1$. Then, we use $\varepsilon_{ij}$ to denote the review action, $\varepsilon_{ij} = (n_i, \text{time}_1, \text{IP}_1)$, and we discard $m_i$ for the comparisons in Eq.2 and Eq.3.

**Approximate comparisons.** Furthermore, we use a customizable $\cap$ operator to the set intersection and set union in Eq.2 and Eq.3. For example, considering two edge-attribute tuples $\varepsilon_{13} = (n_1, \text{time}_1, \text{IP}_1)$ and $\varepsilon_{14} = (n_1, \text{time}_2, \text{IP}_2)$ and let $\Delta$ denote a time range, then $\varepsilon_1 \equiv \varepsilon_2$ if $\text{time}_1 - \text{time}_2 < \Delta$. To make the computation fast, we quantize timestamps (e.g., hours) and use $= \text{operator}$.

**Reducing the C-score computation complexity.** In the worst case, it takes $O(|M|^2)$ to compute $C_{ij}$, $\forall (m_i, m_j) \in M$, during the OSG construction.

In practice, we only need to compute the object pair $(m_i, m_j)$ with positive $C_{ij}$. We use the key-value approach to compute the $S'$-score of $C$-score, described in Algorithm 1 (We use the similar method to compute the $S'$-score).

Naively, it takes $O(|E|)$ to find all key-value pairs (lines 1-2) and takes $O(|E|)$ to build $G$ (lines 4-10). However, we expect $G$ to be sparse because an object only has positive $C$-scores with a very small subset of objects in the OSG. Empirically, we evaluate the edge density in several datasets and find the edge density quite low in all cases. Section 5.3 provides more details.

Furthermore, due to the Zipf-law, in many real datasets, there are a few objects with extremely high in-degrees in the bipartite graph. For example, a celebrity on Twitter (or a popular store on Amazon) has a vast number of followers (or customers). In our preprocessing step, we delete these nodes and their incoming edges, as the most popular objects are usually not fraudulent. This preprocessing significantly reduces $|E|$ and $|E|$, and thus the overall computation time for OSG construction.

**4.2 Stage II: Clustering on OSG (LPA-TK)**

We propose an algorithm, Label Propagation Algorithm based on Top-K Neighbors on Weighted Graph (LPA-TK), to cluster nodes of OSG into groups in face of camouflage. The algorithm is inspired by LPA[6, 31] that has proven effective in detecting communities with dense connectivity structure, while LPA only works on the unweighted graph and does not resist the noise/camouflage edges.

LPA-TK takes the OSG $G$ as input and outputs multiple groups of similar objects, based on the similarity. Algorithms 2-3 describe LPA-TK.

**Initialization (Line 1-3).** First, we assign each node in OSG a unique label. Second, we color all nodes so that no adjacent nodes

![Algorithm 1 Building OSG](image)

Figure 3: (a) and (b) are the OSG matrixes consisting of 100 objects. ID 1-75 are normal objects sampled from the real-world dataset \[18\]. ID 76-100 are fraud objects sampled from an injected fraud group which consists of 100 users and 50 objects (edge density within the group is 0.066). (a) Fraud objects form a dense region in OSG (edge density is 1.0). (2) We add camouflage edges which are 75 percent of fraudulent edges. Then the dense region formed by the fraud objects does not change, and camouflage edges only produce tiny edge weights in camouflage zone.
Algorithm 2 LPA-TK

Require: G = (M, E)
Ensure: MS
1: for each \( m_i \in M \) do \( L_i^0 = m_i \) \# Initialize labels
2: \{M1, ..., M8\} ← M \# Color partition: no two adjacent vertices share the same color.
3: \( t ← 0 \), MS ← ∅
4: while the stop criterion is not met do
5: \( t ← t + 1 \) \# The \( t \)-th iteration
6: for \( M \in \{M1, ..., M8\} \) do
7: for each \( m_i \in M \) do
8: \( L_i^t = f(m_i, t) \) \# Algorithm 2 defines \( f \)
9: for each \( l \in \{L_i : m_i \in M, L_i = l\} \) do
10: MS ← MS + \{\( m_i : m_i \in M, L_i = l\}\}
11: return MS

share the same color. The coloring process is efficient and parallelizable, which takes only \( O(deg(G)) \) synchronous parallel steps \[2\]. And the number of colors, denoted by \( \delta \), is upper bounded by \( deg(G) + 1 \), where \( deg(G) \) denotes the maximum degree over all nodes in \( G \).

Iterations (Line 4-8). In the \( t \)-th iteration, each node \( m_i \) updates its label based on the labels of its neighbors (we leave the update criterion to Algorithm 3). Since the update of a node’s label is only based on its neighbors, we can simultaneously update all the nodes sharing the same color. Thus, we need at most \( \delta \) updates per iteration. The iteration continues until it meets the stop condition:

\[
\forall m_i \in M : 1) L_i^t = L_i^{t-1} \quad \text{or} \quad 2) L_i^t \neq L_i^{t-1}, \text{due to a tie}
\]

where \( L_i^t \) is the label of \( m_i \) in the \( t \)-th iteration, and \( tie \) represents a condition that \( L_i^t \) changes because \( f \) returns more than one label choices (line 8).

Return Groups (Line 9-11). After the iteration terminates, we partition nodes sharing a same final label into a group.

The key difference of LPA-TK from the original algorithm LPA[6] is the designing of update criterion \( f \). We consider the three choices of \( f \).

[Update Criterion: Sum]. Obviously, it is significant to design \( f \) that determines the final results. Based on the update criterion in [6] that only works on unweighted graphs, we first define \( f \) as the following form:

\[
f(m_i, t) = \arg \max_{l \in \{L_j^t : m_j \in H(m_i)\}, m_j \in H(m_i)} \sum C_{ij} h(L_j^t, l).
\]

where \( H(m) \) is the set of neighbors of \( m_i \) and \( h(L_j^t, l) \) is an indicator function:

\[
h(L_j^t, l) = \begin{cases} 1 & \text{if } L_j^t == l, \\ 0 & \text{otherwise.} \end{cases}
\]

According to Eq.4, the label of \( m_i \) is determined by the sum of edge weights of each distinct label among its neighbors. Unfortunately, the results of clustering deteriorate as the camouflage edges increase. Fig.4(a) gives an intuitive example.

[Update Criterion: Max]. To minimize the influence of camouflage, we propose another form of \( f \):

\[
f(m_i, t) = \arg \max_{l \in \{L_j^t : m_j \in H(m_i)\}} h(l, m_i)
\]

where \( H(m_i) \) is the set of neighbors of \( m_i \) and

\[
h(l, m_i) = \max \{C_{ij} : m_j \in H(m_i), L_j^t = l\}
\]

Based on Eq.5, the label of \( m_i \) is determined by the maximal edge weight of each distinct label among its neighbors. Although Eq.5 can eliminate the influence of camouflage because the most similar neighbor of a fraud object should also be fraudulent, the result of clustering is not well and a group of objects is often divided into multiple parts. Fig.4(b) gives an example.

![Figure 4](image-url)

Figure 4: (a) and (b) describe two possible states when determine the label of a fraud object \( m \), in which the thickness of an edge indicates the C-score. In (a), normal objects labeled as ‘B’ are connected with \( m \) because of camouflage edges. According to Eq.4, \( m \) will be labeled as ‘B’, while we expect it to be ‘A’. In (b), according to Eq.5, \( m \) will be labeled as ‘B’, while we expect it to be ‘A’.

[Update Criterion: Top K]. Based on these considerations, we propose our final form of \( f \), which can eliminate the influence of camouflage and keep ideal clustering results, shown in the Algorithm 3.

Algorithm 3 \( f(m_i, t) \)

Require: \( K \)
1: for each \( l \in \{L_j^t : m_j \in H(m_i)\} \) do \# \( H(m_i) \) is the set of neighbors of \( m_i \)
2: \( h[l] ← 0 \)
3: \( C ← \{C_{ij} : m_j \in H(m_i), L_j^t = l\} \)
4: while \( K > 0 \) do
5: if \( |C| \geq K \) then
6: \( h[l] ← h[l] + \max C \)
7: remove max \( C \) from \( C \)
8: \( K ← K - 1 \)
9: return \( l \) if \( h[l] \) is the maximum in \( h \)

In Algorithm 2, the label of \( m_i \) is determined by the sum of Top-K maximal edge weights of each distinct label among its neighbors. Note that computing the sum of Top-K maximal edge weights
After generating all candidate groups, in this section, we propose an interpretable suspiciousness metric \( F \) to score each group and find top suspicious groups. Given a fraud group \( M \in \hat{M} \) returned by LPA-TK, let \( \mathcal{G} \) be the subgraph of \( G \) induced by \( M \), \( \mathcal{G} = (M,E) \). Then \( F \) follows the form:

\[
F_\mathcal{G} = \frac{\sum_{(m_i,m_j) \in E} C_{ij} \cdot \sum_{(m_i,m_j) \in E} |I_i \cap I_j|}{|M|(|M|-1)^2},
\]

where \( F_\mathcal{G} = F_\mathcal{G}^1 \cdot F_\mathcal{G}^2 \cdot |M| \),

\[
F_\mathcal{G}^1 = \frac{\sum_{(m_i,m_j) \in E} C_{ij}}{|M|(|M|-1)},
\]

and

\[
F_\mathcal{G}^2 = \frac{\sum_{(m_i,m_j) \in E} |I_i \cap I_j|}{|M|(|M|-1)}.
\]

Intuitively, \( F_\mathcal{G}^1 \) is the average value of \( C \)-score on all edges of \( \mathcal{G} \), \( F_\mathcal{G}^2 \) is the average number of edges pointed from same users on all object pairs of \( \mathcal{G} \).

The advantage of \( F \)-score is that the score obeys the following good properties including axioms proposed in [15] that all good algorithms should have. First, we present a well-known metric, edge density denoted by \( \rho_{edge} = \frac{|E|}{|M|(|M|-1)} \). And we use ‘+’, ‘↑’ and ‘=’ to represent ‘increase’, ‘decrease’, and ‘not change’.

(i) AXIOM 1. [Object Size]. Keeping \( \rho_{edge}, C_{ij} \), and \( |I_i \cap I_j| \) fixed, a larger \( \mathcal{G} \) is more suspicious than one with a smaller size.

\[
(F_\mathcal{G}^1 \uparrow) \land (F_\mathcal{G}^2 \uparrow) \land (|M| \uparrow) \Rightarrow F_\mathcal{G} \uparrow
\]

(ii) AXIOM 2. [Object Similarity]. Keeping \( \rho_{edge}, |I_i \cap I_j| \), and \(|M|\) fixed, a \( \mathcal{G} \) with more similar object pairs is more suspicious.

\[
C_{ij} \uparrow \Rightarrow (F_\mathcal{G}^1 \uparrow) \land (F_\mathcal{G}^2 \uparrow) \Rightarrow F_\mathcal{G} \uparrow
\]

(iii) AXIOM 3. [User Size]. Keeping \( \rho_{edge}, C_{ij} \), and \(|M|\) fixed, a fraud object group \( \mathcal{G} \) connected with more fraud users is more suspicious.

\[
|I_i \cap I_j| \uparrow \Rightarrow (F_\mathcal{G}^1 \uparrow) \land (F_\mathcal{G}^2 \uparrow) \Rightarrow F_\mathcal{G} \uparrow
\]

(iv) AXIOM 4. [Edge Density]. Keeping \( C_{ij}, |I_i \cap I_j| \), and \(|M|\) fixed, a denser \( \mathcal{G} \) is more suspicious.

\[
\rho_{edge} \uparrow \Rightarrow (F_\mathcal{G}^1 \uparrow) \land (F_\mathcal{G}^2 \uparrow) \Rightarrow F_\mathcal{G} \uparrow
\]

(v) AXIOM 5. [Concentration]. With the same total suspiciousness, a smaller \( \mathcal{G} \) is more suspicious. We define the total suspiciousness as \( \sum_{(m_i,m_j) \in E} (C_{ij} + |I_i \cap I_j|) \).

\[
|M| \downarrow \Rightarrow F_\mathcal{G} \uparrow
\]

Note that naive metrics do not meet all axioms. For example, the edge density is not a good metric because it does not satisfy AXIOM 1-3 and 5.

Therefore, leveraging \( F \), we can sort groups in descending order of suspiciousness and catch top suspicious groups.

Given suspicious \( \mathcal{G} \), we catch fraud users \( N \) from \( \mathcal{G} \) comprised of fraud objects \( M \). The approach follows the form:

\[
N = \bigcup_{\forall m_i, m_j \in M} H_{i \cap H_{j}}
\]

where \( H_{i} = \{n : \forall n \in N, (n,m_i) \in E\} \) is the set of users having edges to \( m_i \).

To reduce false alarms in \( N \), we filter out users with low out-degrees in the subgraph induced by \( N \) and \( M \) of \( \mathcal{G} \), because a normal user may interact with a few fraud objects by accident while it is unlikely that it interacts with many fraud objects.

4.4 Analysis

There are four advantages of \( \text{FRAUDTRAP} \) (\( C + \text{LPA-TK} + F \)):

1. [Camouflage-resistance], \( C + \text{LPA-TK} \) is inherent to resist camouflage (see Theorem 4.2). However, for LPA[6, 31], its group detection results can be easily destroyed by camouflage (demonstrated in Sec.5.1).

2. [Capturing Loose Synchrony], \( C + \text{LPA-TK} + F \) focuses on catching loosely synchronized behavior, because its top-K most similar neighbors do not change in OSG for a fraud object. However, The density signal can be decreased significantly by synchrony reduction [13, 15, 30] (demonstrated in Sec.5.2).

3. [Clustering global similarities], Using \( C + \text{LPA-TK} \), we cluster nodes based on their similarity. However, [13, 36, 37] group nodes based on their degree or density features. Fig.7 in Sec. 5 shows an intuitive example of the clustering quality.

4. [Scalability]. \( C + \text{LPA-TK} \) cluster all objects into groups in one run with near-linear time complexity (Sec.4.4). Leveraging \( F \), we can obtain Top-k suspicious groups, while [13, 36, 37] only detect a single group per run.

Time complexity. In the OSG construction stage, it takes \( O(|E| + |E|) \) time, based on the optimization (Sec.4.1). In Stage II, the time cost is the product of the number of iterations and the number of colors, where the former value has been experimentally indicated to grow logarithmically with graph size [6] and the latter value is
bounded by $\text{deg}(G) + 1$. In Stage III, it takes $O(|E|)$ to compute $F^*$-score and catch fraud users of $G$, where $|E| << |E|$. Thus, FraudTrap has near-linear time complexity.

Capturing loosely synchronized behavior. We use a concrete example to show why the algorithm can handle loosely synchronized behaviors.

Consider a fraud group with 100 fraud users and 50 fraud objects, and each fraud user creates 30 edges to random fraud objects. Let $G_{\text{org}}$ denote the induced subgraph induced in the original user-object bipartite graph, and let $G_{\text{OSG}}$ denote the subgraph formed by fraud objects in OSG. We compute the edge density $\rho_{\text{edge}}$ and $F^*$ in Eq.(6) for both $G_{\text{org}}$ and $G_{\text{OSG}}$. We have

$$\rho_{\text{edge}}(G_{\text{org}}) = 0.134 \text{ VS } \rho_{\text{edge}}(G_{\text{OSG}}) = 1 \land F^*_1\gg G_{\text{OSG}} = 0.506.$$

Obviously, the subgraph in OSG is much denser than the original bipartite graph. Then, let us reduce the synchrony of fraud group by doubling the number of fraud users and keep the same number of edges. Then we have

$$\rho_{\text{edge}}(G_{\text{org}}) = 0.049 \text{ VS } \rho_{\text{edge}}(G_{\text{OSG}}) = 1 \land F^*_1\gg G_{\text{OSG}} = 0.251.$$  

It shows that $G_{\text{OSG}}$ is affected slightly by the reduction of synchrony, compared to $G_{\text{org}}$. Furthermore, as normal users hardly exhibit synchronized behavior, the C-score of normal object pairs are close to zero. Thus, FraudTrap ($C^* + \text{LPA-TK} + F^*$) is inherently more effective than approaches relying on density [13, 30, 33].

Camouflage-resistance. FraudTrap is robust to resist different types of camouflage. There are two reasons. First, the $F^*$-scores of subgraphs induced by fraud object groups do not decrease while adding camouflage edges. Formally, we give the following theorem.

**Theorem 4.2.** Let $G$ denote a subgraph induced by fraud objects $M$, and $N$ denote the fraud users. $M$ and $N$ are from a single fraud group. $G$ does not change when users in $N$ add camouflage edges to non-fraud objects.

**Proof.** Let $m_i$ and $m_j$ denote two fraud objects, $(m_i, m_j) \in M$. Camouflage only introduces edges between $N$ and normal objects. It does not add or remove edges pointing to $M$, which demonstrates that $I_i$ and $I_j$ in Eq.(1) do not change. Thus $C_{ij} \in G$ does not change, $\forall(m_i, m_j) \in M$. $\square$

Second, in OSG, a camouflage edge between a fraud user and a normal object only produces a quite small value of $C$-score due to the denominator of Eq. (1). Fig. 3 (b) provides a typical case. For a fraud user, this indicates that camouflage edges do not change its the most top-$K$ similar neighbors. Thus, the subgraphs induced by fraud groups can be effectively detected by LPA-TK.

**Effectiveness of the semi-supervised mode.** Given a subset of labeled fraud users, FraudTrap switches to the semi-supervised mode. Because of the design of $C$-score, the partially labeled data does enhance the similarities between fraud objects in a group and increase the density of induced subgraph on OSG. Thus, unsurprisingly, LPA-TK will more accurately cluster fraud objects into groups. The experiments in Section 5.2 show the fact.

5 EXPERIMENTS AND RESULTS

We want to answer the following questions in the evaluation:

(1) How does FraudTrap handle loosely synchronized behavior?
(2) Is FraudTrap robust with different camouflage?
(3) Does the semi-supervised mode improve the performance?
(4) Is FraudTrap scalable to large real-world datasets?

Table 3 gives the details of datasets used in the paper.

**Table 3: Datasets used in experiments.**

| datasets         | edges | datasets         | edges |
|------------------|-------|------------------|-------|
| AmazonOffice     | 53K   | YelpChi          | 67K   |
| AmazonBaby       | 160K  | YelpNYC          | 359K  |
| AmazonTools      | 134K  | YelpZip          | 1.14M |
| AmazonFood       | 1.3 M | DARPA            | 4.55M |
| AmazonVideo      | 583K  | Registration     | 26k   |
| AmazonPet        | 157K  |                  |       |

Implementation and existing methods in comparison. We implemented FraudTrap by Python and we run all experiments on a server with two 10-core 2.2 GHz Intel Xeon E5 CPUs and 64 GB memory. We compared FraudTrap with the following three state-of-the-art methods that focus on synchronized behavior with application to fraud detection.

(1) Fraudar[13] finds the subgraph with the maximal average degree in the bipartite graph using an approximated greedy algorithm. It is designed to be camouflage-resistance.
(2) CatchSync[16] specializes in catching rare connectivity structures of fraud groups that exhibit the synchronized behavior, it proposes the synchronicity and normality features based on the degree and HITS score of the user.
(3) CrossSpot[15] detects the dense blocks which maximize the suspicousness metric in the multi-dimensional dataset.

We did our best to fine-tune the hyper-parameters to achieve their best performances. For CrossSpot, we set the random initialization seeds as 400, 500 and 600, and chose the one with the maximal F1-score. Fraudar detects the subgraph with the maximal average degree and multiple subgraphs by deleting previously detected nodes and their edges. For all methods, we test the performance according to the rank of the suspicousness scores. We compared the performance using the standard metric, F1 score (the harmonic mean of precision and recall) across all algorithms.

**FraudTrap and FraudTrap+.** We run FraudTrap in two modes. The unsupervised mode (FraudTrap) and the semi-supervised mode (FraudTrap+) assuming 5% fraud users are randomly labeled. And in all experiments, we set $K = 3$ for LPA-TK. In the experiments regarding [Amazon] datasets, assume $M$ is a fraud object group returned by FraudTrap, $N$ is a fraud user group returned by Eq.7. Then we filtered out $n$ if the out-degree of $n$ is less than 3 in the subgraph induced by $(N, M)$ of $G$, $\forall n \in N$.

**Fraud Group Formulation.** To simulate the attack models of smart fraudsters, we used the same method as [13, 16] to generate labels: inject fraud groups into Amazon datasets [Amazon] datasets contain six collections of reviews for different types of commodities.
on Amazon, listed in Table 3). To accurately depict the injection, we formulate the fraud group as the following.

**Definition 5.1 ($\rho$-Synchrony).** Given a subgraph $G([N], |M|, \rho, \theta)$ induced by a group $(N, M)$ in $G$ where $N$ is a set of users, $M$ is a set of objects. (1) For each $n \in N$, $\exists W \subseteq M$ where for each $m \in W$, the edge $(n, m)$ exists. We define $\rho$ as

$$\rho = \frac{|W|}{|M|},$$

where $|W|$ is the mean for all $|W|$s. (2) For each $n \in N$, $\exists W$ and $W \cap M = \emptyset$, where for each $m \in W$, the edge $(n, m)$ exists. We set $\theta = |W|$, and $|W|$ is the mean for all $|W|$s.

Thus, we use $G([N], |M|, \rho, \theta)$ to represent a fraud group, where $\rho$ represents how loosely its synchronized behavior is and $\theta$ denotes the number of camouflage edges of each user on average. Naturally, $N$ and $M$ are labeled as ‘fraudulent’. Before we evaluate the performance of FraudTrap, we first verify the effectiveness of LPA-TK.

### 5.1 Performance of LPA-TK

We recall that LAP-TK has the best clustering performance and camouflage-resistance. We design this experiment to demonstrate its performance. We injected a fraud group $G([N] = 200, |M| = 50, \rho = 0.3, \theta)$ into AmazonOffice dataset, where $\rho = 0.3$ indicates that each fraud user of $N$ reviews 15 fraud objects of $M$ and $\theta$ represents the number of camouflage edges of each fraud user on average. We varied $\theta$ to specifically examine the resistance to camouflage of each clustering algorithm. Let $G$ denote the bipartite graph formed by injected AmazonOffice and we built the OSG of $G$ using the method in section 4.1. G. We run each algorithm on $G$ and evaluated the clustering performance and the performance of detecting fraud objects, and we used metric $\mathcal{F}$ to compute suspiciousness scores of detected groups. Note that we only injected one group and thus $M$ should be clustered into one group. LPA denotes the algorithm [6] that treats each edge weight equally, LPA-Sum denotes the Algorithm 2 + Eq.4 and LPA-Max denotes the Algorithm 2 + Eq.5.

Table 4: Clustering performance on the [Amazon] datasets. ‘Num’ represents ‘the number of groups $M$ is divided into’, $\theta$ denotes the number of camouflage edges of each fraud user.

|            | $\theta = 0$ | $\theta = 5$ | $\theta = 10$ | $\theta = 20$ |
|------------|--------------|--------------|---------------|---------------|
|            | Num/AUC      | Num/AUC      | Num/AUC       | Num/AUC       |
| LPA        | 1 | 1.0          | 1 | 0.787        | 1 | 0.727        | 1 | 0.731        |
| LPA-Sum    | 1 | 1.0          | 1 | 1.0          | 1 | 0.787        | 1 | 0.761        |
| LPA-Max    | 14 | 0.998        | 14 | 0.996        | 13 | 0.998        | 10 | 0.991        |
| LPA-TK     | 1 | 1.0          | 1 | 1.0          | 1 | 0.999        | 1 | 0.998        |

Table 4 presents the clustering performance of each algorithm. In the setup, we expect ‘AUC’ $= 1.0$ for the perfect performance of detecting fraud objects and ‘Num’ $= 1$ for the perfect clustering result. Then we have the following observations: (1) without camouflage ($\theta = 0$), LPA has an ideal performance. However, once camouflage is added ($\theta \geq 0$), its performance is destroyed and LPA clustered all objects into one group (thus ‘num’ $= 1$). (2) LPA-Sum shows weak camouflage-resistance, and its performance deteriorates as camouflage edges increases. (3) LPA-Max resists camouflage obviously. However, it divided $M$ into multiple groups, which is not good to group analysis and inspection. (4) Our algorithm LPA-TK has perfect performance. It clustered all fraud objects into one group and separated the group from legit objects even in face of camouflage. Thus the experiments demonstrate the advantages of LPA-TK.

### 5.2 Overall Performance of FraudTrap

To [Amazon] datasets, we designed two fraud group injection schemes: the first is to examine in detail the performances for detecting loosely synchronized behavior and resisting camouflage; the second is for more general performance evaluation.

**[Injection Scheme 1].** We chose AmazonOffice as the representative and injected a fraud group into it with varying configurations. We set the fraud group as $G([N] = 200, |M| = 50, \rho, \theta = |M| \times \rho)$. We introduced two perturbations according to strategies of smart
fraudsters: (1) reduce synchrony by decreasing $\rho$; (2) add camouflage edges obeying $\theta$. In $G$, $\theta = |M| \times \rho$ indicates that the number of a node’s camouflage edges is equal to the number of its edges within the fraud group. And we used all four types of camouflage as in Sec. 3.

Figure 6 and Figure 5 summarize the performance of detection fraud objects and fraud users with varying $\rho$ respectively, where the X-axes are the synchronization ratio $\rho$ (varying from 0 to 0.5), and the Y-axes are the F1 scores. We have the following observations. 1) Without camouflage and with a high synchronization ratio $\rho$, both FraudTrap and CatchSync can catch all injected frauds. 2) At lower $\rho$’s, even without camouflage, the performance of Fraudar decreases significantly, but FraudTrap maintains its performance. In fact, even for $\rho = 0.1$, the edge density of the fraud group is only 0.006, FraudTrap can still achieve an F1 score of 0.97. The effects confirm the robustness of our novel approach ($C + LPA-TK + F$). 3) Camouflage significantly decreases the performance of CatchSync, but both FraudTrap and Fraudar are resistant to camouflage. Not surprisingly, FraudTrap performs much better when camouflage and loose synchronization exist together. 4) As shown in Fig. 6, without the camouflage and loose synchronization, CatchSync and Fraudar perform perfectly, but their performance degrade quickly when $\rho$ decreases with camouflage. 5) CrossSpot performs poorly for any $\rho < 0.5$.

[Injection Scheme 2]. In this experiment, we injected 5 fraud groups $G(|N| = 200, |M| = 50, \rho, \theta = |M| \times \rho)$ into AmazonOffice, AmazonBaby, AmazonTools, AmazonFood, AmazonVideo, and AmazonBook, in which $\rho$ was randomly chosen from $[0.2, 0.6]$, respectively. Out of the 5 fraud groups, 1 of them is no camouflage, 4 out of them are augmented with four types of camouflage respectively. The performances are shown in Table 5 and Table 6 with respect to the detection of fraud objects and users. Overall, FraudTrap is the most robust and accurate across all variations and camouflages. The semi-supervised FraudTrap+ with a random selection of 5% labeled fraud users kept or further improve the performance, verifying the conclusion in Section 4.4.

Table 5: Performance (AUC) of detecting fraud objects on the [Amazon] datasets

|                  | AmazonOffice | AmazonBaby | AmazonTools |
|------------------|--------------|------------|-------------|
| Fraudar          | 0.8915       | 0.8574     | 0.8764      |
| CatchSync        | 0.8512       | 0.8290     | 0.8307      |
| CrossSpot        | 0.8213       | 0.8342     | 0.7923      |
| Fraudar+         | 0.9987       | 0.9495     | 0.9689      |
| FraudTrap+       | 0.9987       | 0.9545     | 0.9675      |

Table 6: Performance (AUC) of detect fraud users on the [Amazon] datasets

|                  | AmazonOffice | AmazonBaby | AmazonTools |
|------------------|--------------|------------|-------------|
| Fraudar          | 0.9015       | 0.8673     | 0.8734      |
| CatchSync        | 0.8732       | 0.8391     | 0.8304      |
| CrossSpot        | 0.8113       | 0.8422     | 0.7823      |
| Fraudar+         | 1            | 0.9795     | 0.9796      |
| FraudTrap+       | 1            | 0.9845     | 0.9855      |

Table 7: Performance (AUC) on the [YELP] datasets

|                  | YelpChi | YelpNYC | YelpZip |
|------------------|---------|---------|---------|
| Fraudar          | 0.9905  | 0.8531  | 0.7471  |
| CatchSync        | 0.9889  | 0.8458  | 0.7779  |
| CrossSpot        | 0.9744  | 0.7965  | 0.7521  |
| Fraudar+         | 0.9905  | 0.8613  | 0.7793  |
| FraudTrap+       | 0.9905  | 0.8653  | 0.7953  |

[DARPA] DARPA[23] was collected by the Cyber Systems and Technology Group in 1998. It is a collection of network connections, some of which are TCP attacks. Each connection contains source IP and destination IP. Thus the dataset can be modeled as a bipartite graph $G$ formed by (source IPs, destination IPs) and we evaluate the performance of detecting malicious source IPs and destination IPs respectively for each method. The dataset includes labels indicating whether each connection is malicious or not and we labeled an IP as ‘malicious’ if the number of fake reviews it receives exceeds 40 (because legit restaurants also may contain a few fake reviews). Table 8 shows the results. FraudTrap and FraudTrap+ have the best accuracy on all three datasets.

Table 8: Performance (AUC) on the [DARPA] datasets

|                  | DARPA   |
|------------------|---------|
| Fraudar          | 0.9071  |
| CatchSync        | 0.8732  |
| CrossSpot        | 0.8512  |
| Fraudar+         | 0.9071  |
| FraudTrap+       | 0.9071  |

[Yelp] [32], YelpChi, YelpNYC, and YelpZip are three datasets collected by [25] and [32], which contain a different number of reviews for restaurants on Yelp. Each review includes the user who made the review and the restaurants. Thus the three datasets can be represented by the bipartite graph $G$ formed by (users, restaurants). The three datasets all include labels indicating whether each review is fake or not. Detecting fraudulent users has been studied in [32] but used review text information. In this paper, we give the evaluation of catching fraudulent restaurants which bought fake reviews only using the two features. Intuitively, more reviews a restaurant contains, the more suspicious it is. Therefore, we label a restaurant as "fraudulent" if the number of fake reviews it receives exceeds 40 (because legit restaurants also may contain a few fake reviews). Table 7 shows the results. FraudTrap and FraudTrap+ have the best accuracy on all three datasets.
whether each entry (user ID) is fraudulent or not and the labels are obtained by tracking these user accounts for several months, and see if the user conduct fraud after registration and each fraud user has a group ID according to obvious attribute sharing among them. Of all user accounts, 10k are fraudulent. Note that the registration records do not contain the user-object interaction. We can easily adapt it to the FraudTrap framework assuming that each registered account ("object") has many followers identified by a feature value, IP subnet or phone-prefix ("user"). Intuitively, an IP subnet can be used in many registrations, and we model it the same as a user follows multiple objects in a social network. Moreover, we use FraudTrap* to denote the mode of FraudTrap using the side information timestamp, and an edge $\varepsilon = (\text{IPsubnet/phone}, \text{timestamp})$ in FraudTrap*.

### Table 8: Performance (AUC) on the [DARPA] dataset

| Detection of | source IP | destination IP |
|--------------|-----------|----------------|
| Fraudar      | 0.7420    | 0.7298         |
| CatchSync    | 0.8069    | 0.8283         |
| CrossSpot    | 0.7249    | 0.6784         |
| FraudTrap    | 0.9968    | 0.9920         |
| FraudTrap+   | 0.9968    | 0.9920         |

Table 9: Performance (AUC) on the [Registration] dataset

| Feature: IP | Feature: phone |
|-------------|----------------|
| Fraudar     | 0.7543         | 0.8742         |
| CatchSync   | 0.7242         | 0.8435         |
| CrossSpot   | 0.6976         | 0.8231         |
| FraudTrap   | 0.7658         | 0.8979         |
| FraudTrap+  | 0.7826         | 0.9113         |
| FraudTrap*  | 0.7724         | 0.9215         |

In our first experiment, we only used the IP subnet feature as the “user” side of the bipartite graph. The left half of Table 9 summarizes the results. We have the following observations: 1) FraudTrap, FraudTrap+, and FraudTrap* outperformed all the other existing methods by a small margin. Taking a closer look at the detection result, we found that FraudTrap captured a fraud group of 75 fraud users that all other methods missed. The group is quite loosely synchronized with the edge density of only 0.14 in the original bipartite graph. However, having an edge density in 1.0 in OSG makes it highly suspicious in FraudTrap. 2) FraudTrap+ performed better than the unsupervised version, even with only 5% of the fraud labels.

In the second experiment, we used phone features as the “user” side of the bipartite graph. The right column of Table 9 summarizes the results. The key observations are: 1) FraudTrap, FraudTrap+, and FraudTrap* still outperformed other methods. Other baselines have lower performance because they worked poorly on a group with 125 false positives (and 75 true positives). This is because they are based on edge density on the bipartite graph only, and the groups’ edge density is too big enough to distinguish this group from the normal.

As an additional benefit, FraudTrap can provide insights on the grouping of fraud users/objects by their similarity. Fig. 7 plots a projection of the data onto a 2D space using t-SNE[40] on [Registration]. We labeled the fraud groups and normal groups according to the F-score ranking. We plotted the users in the same group with the same color. We expect that points in the similar groups are clustered together. We observe that the results from FraudTrap are much better than Fraudar and CrossSpot, since the users with the same color cluster better, which is very similar to the clustering result of labels.

### 5.3 Scalability

**Sparsity of OSG edges.** All the three datasets above have low edge densities. In fact, we also studied several public datasets by the construction of the OSG and the computation of the edge density. For example, three datasets in [18] and one dataset in [21] have edge densities of 0.0027, 0.0027, 0.0028, 0.0013 respectively. With this density, the time and space complexities of FraudTrap are both near-linear to the number of edges in the graph.

Based on the dataset AmazonFood, we vary the number of edges using downsampling, and verify the running time of FraudTrap is indeed near-linear, as shown in Fig. 8.

### 6 CONCLUSION

Fraudsters can adapt their behavior to avoid detection. Specifically, they can reduce their synchronized behaviors and conduct camouflage to destroy the performance of state-of-the-art methods in the literature. To solve the challenges, we propose FraudTrap to capture the more fundamental similarity among fraud objects, and work on the edge density on the Object Similarity Graph (OSG) instead. We design FraudTrap with many practical considerations...
for the general fraud detection scenario in many applications, such as supporting a mixture of unsupervised and semi-supervised learning modes, as well as multiple features. We believe the metrics of FraudTrap are much harder for fraudsters to manipulate.

REFERENCES

[1] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. 2013. Opinion Fraud Detection in Online Reviews by Network Effects. In ICWSM. The AAAI Press.

[2] Leonid Baranenhoum and Michael Elkin. 2009. Distributed (δ+1)-coloring in linear (in δ) time. In ACM Symposium on Theory of Computing. 111–120.

[3] Alex Beutel, WanHong Xu, Venkatesan Guruswami, Christopher Polos, and Chris Faloutsos. 2013. CopyCatch: Stopping Group Attacks by Spotting Lockstep Behavior in Social Networks. In WWW. ACM, New York, NY, USA, 119–130.

[4] Yazen Boshmaf, Dionysis Logothetis, Georgos Siganos, Jorge Lera, Jose Lorenzo, Matei Ripeanu, and Konstantin Beznosov. 2015. Integeo: Leveraging Victim Prediction for Robust Fake Account Detection in OSNs. In Network and Distributed System Security Symposium. 142–168.

[5] Qiang Cao, Christopher Palow, Christopher Polos, and Christopher Polos. 2014. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In ACM CCS. 477–488.

[6] G. Cordasco and L. Gargano. 2010. Community detection via semi-synchronous label propagation algorithms. In 2010 IEEE International Workshop on: IASNA. 1–8.

[7] George Daneri and Prateek Mittal. 2009. Sybilinfer: Detecting Sybil Nodes using Social Networks. In NDSS the Internet Society (2009).

[8] Manuel Egele, Gianluca Stronghini, Christopher Kruegel, and Giovanni Vigna. 2017. Towards Detecting Compromised Accounts on Social Networks. IEEE TIDSC 14, 4 (2017), 447–460.

[9] Hector Garcia-Molina and Jan Pedersen. 2004. Combating web spam with trustrank. In VLDB. 576–587.

[10] Saptarshi Ghosh, Yiyeng Peng, Alex Beutel, Christos Faloutsos, and Chris Faloutsos. 2015. Understanding and combating link farming in the twitter social network. In Proceedings of the 2015 SIAM International Conference on Data Mining. SIAM, 495–503.

[11] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. CatchSync: catching synchronized behavior in large directed graphs. In ACM SIGKDD. 941–950.

[12] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2016. Inferring lockstep behavior from connectivity pattern in large graphs. Knowledge & Information Systems 48, 2 (2016), 399–428.

[13] McAuley Julian. [n. d.]. Amazon product data. http://jmcauley.ucsd.edu/data/amazon/.

[14] Stijan Kumar, Bryan Hooi, Disha Makjhi, Mohit Kumar, Christos Faloutsos, and V. S. Subrahmanian. 2017. FairJudge: Trustworthy User Prediction in Rating Platforms. CoRR (2017).

[15] Kyumin Lee, James Caveree, and Steve Webb. 2010. Uncovering social spammers: social honeypots + machine learning. ACM SIGIR (2010). 435–442.

[16] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Signed networks in social media. In SIGCHI Conference on Human Factors in Computing Systems. 1361–1370.

[17] Yuan Li, Yiheng Sun, and Noshir Contractor. 2017. Mining Assisted Semi-supervised Learning for Fraudulent Cash-out Detection. In Proceedings of the 2017 IEEE/ACM ASONAM. 546–553.

[18] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zisman. 2000. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00, Vol. 2: 12–26 vol.2. https://doi.org/10.1109/DISCEX.2000.821506.

[19] Abdelaliz Mohaisen, Nicholas Hopper, and Yongdae Kim. 2011. Keep your friends close: Incorporating trust into social network-based Sybil defenses. In IEEE INFOCOM. 1943–1951.

[20] Atjijn Mukherjee, Vivek Venkataraman, Bing Liu, and Natalie Glance. 2013. What yelp fake review filter might be doing? In Proceedings of the 7th International Conference on Weblogs and Social Media, ICWSM 2013. AAAI press, 409–418.

[21] U.S. Department of Justice Federal Bureau of Investigation. 2015. 2015 Internet Crime Report. https://pdf.ic3.gov/2015_IC3Report.pdf.

[22] B. Aditya Prakash, Ashwin Sriraman, Mukund Seshadri, Sridhar Macharaju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In PKDD.

[23] V. S. Subrahmanian. 2017. FairJudge: Trustworthy User Prediction in Rating Platforms. In WWW. 201–202.

[24] McAuley Julian. [n. d.]. The DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. New Physiologist 11, 2 (n. d.)), 37–50.

[25] B. Aditya Prakash, Ashwin Sriraman, Mukund Seshadri, Sridhar Macharaju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In PKDD.

[26] Shashank Pandit, Daen Honig Chao, Samuel Wang, and Christos Faloutsos. 2007. Netprobe:a fast and scalable system for fraud detection in online auction networks. In WWW. 201–202.

[27] Shubotu Rayana and Leman Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and metadata. In ACM SIGKDD.

[28] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. 2014. Spottin- Spous: Suspicous Link Behavior with fBox: An Adversarial Perspective. In IEEE International Conference on Data Mining. 959–964.

[29] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnenmann, Disha Makjhi, Mohit Kumar, and Christos Faloutsos. 2017. EdgeCentric: Anomaly Detection in Edge-Attributed Networks. In IEEE ICDM. 327–334.

[30] Kijung Shin, Tina Elassi-Rad, and Christos Faloutsos. 2018. Patterns and anom- als in k-cores of real-world graphs with applications. Knowledge & Information Systems 54, 3 (2018), 677–710.

[31] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. D-Cube: Dense- Block Detection in Terabyte-Scale Tensors. In ACM WSDM. New York, NY, USA, 681–689.

[32] Ming Yang Su. 2011. Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers. Expert Systems with Applications 38, 4 (2011), 3492–3498.

[33] Hua Tang and Zhulin Cao. 2009. Machine Learning-based Intrusion Detection Algorithms. Journal of Computational Information Systems (2009). 1825–1831.

[34] V. S. Subrahmanian. 2017. FairJudge: Trustworthy User Prediction in Rating Platforms. In ACM SIGKDD. 895–904.

[35] Shuif Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. Hin- Droid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network. In ACM SIGKDD. 1507–1515.

[36] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2016. Spotting Suspicious Behaviors in Multimodal Data: A General Metric and Algorithms. IEEE TKDE 28, 8 (2016), 2187–2200.

[37] Meng Jiang, Peng Cui, Christos Faloutsos, and Shiqiang Yang. 2014. MatchSync: catching synchronized behavior in large directed graphs. In ACM SIGKDD. 941–950.