GraphRAD: A Graph-based Risky Account Detection System

Jun Ma*
Amazon
junmaa@amazon.com

Danqing Zhang*†
University of California Berkeley
danqing0703@berkeley.edu

Yun Wang
Amazon
yunwng@amazon.com

Yan Zhang
Amazon
zhngyn@amazon.com

Alexey Pozdnoukhov
University of California Berkeley
alexeip@berkeley.edu

ABSTRACT

Given the linked account graph with tens of millions of vertices, and a list of confirmed risky accounts, how can we quickly find a short list of potentially risky accounts for further human expert investigation? Most mainstream graph-based fraud detection algorithms focusing on detecting dense blocks of fake follows, or fake reviews from the social media graph, however, do not align well with the answer to the question.

Here we hypothesize that fraud accounts share dense connections within a "fraud community", but have less so with accounts outside of the community. We propose GraphRAD, a risky account detection system based on *local graph clustering* algorithms. Our experiments show that from a real-world account graph of $\approx$ 60 million vertices and $\approx$ 500 million edges, GraphRAD was able to catch 67 previously unidentified fraud accounts by proposing 28 small-scale local communities, which is significantly more effective than the baseline model.

KEYWORDS

fraud detection, payment fraud, local graph clustering, community detection, semi-supervised learning, graph regularization

ACM Reference format:
Jun Ma, Danqing Zhang, Yun Wang, Yan Zhang, and Alexey Pozdnoukhov. 2018. GraphRAD: A Graph-based Risky Account Detection System. In *Proceedings of ACM SIGKDD Conference, London, United Kingdom, August 2018 (MLG'18),* 9 pages.
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Payment fraud is a worldwide criminal activity causing huge financial losses every year. According to the trade newsletter Nilson Report, payment fraud has caused $22.8 billion losses worldwide, while the U.S. alone takes 39.5% of the volume (https://www.nilsonreport.com/). Due to the intrinsic physical-card-not-present property, payment fraud is especially a threat to online retail customers, where by using stolen payment information, the fraudsters profit from getting products without paying. Although leading online retailers such as Amazon protects their customers via customer-friendly policies, the financial losses should be managed by effective prevention mechanisms.

In Amazon, while most fraud attacks were stopped by *Risk Detection System* at the transaction time, the escaped attacks are much harder to catch, because these escaped attacks show fewer indicators of riskiness: e.g., using better camouflage, linking to good accounts, or pretending to behave normally. The online retailer will eventually notice these attacks when receiving reports from customers or financial institutes.

Then how do we prevent future attacks from the same identified fraudsters? We hypothesize that fraudsters attack with group accounts, which are usually unintentionally linked with transaction attributes such as shipping address. Once a member of the group accounts has been reported as fraud, riskiness of rest of the group should be revealed. However, good accounts could possibly become false positives simply because of one shared attribute, say, a shared rental home address. Here we hypothesize that fraud accounts form dense subgraphs like communities in social networks, while good accounts are not present in such communities.

In one of the designed work-flows, the human experts need to periodically start from identified fraud accounts, find the fraud-dense subgraphs, make decisions and take actions on suspicious accounts. Due to the high volume demands of human labor, there's need for an automated system that efficiently extracts these subgraphs. The informal problem definition is: given a graph of linked accounts, and a list of identified fraud accounts, such a system should return a **short** list of **small-scale**, **non-overlapping** subgraphs such that **highly suspicious** accounts are concentrated within.

Here we propose GraphRAD (Graph-based Risky Account Detection system), a scalable local-community-detection-based system tailored to meet all our work-flow requirements. The system has 4 major modules: 1) the Graph Generator module that builds the account link graph from transaction records, 2) the Community Detection module that outputs suspicious communities using risky account seeds, 3) the Community Extractor module that outputs non-overlapping communities for human experts' check, and additionally, 4) a semi-supervised Scoring module that provides risk score ranking for human experts to prioritize the work. Overall, the GraphRAD system has shown great potential by successfully

*Both authors contributed equally to the work.
†Work performed while at Amazon as intern.

detecting 67 fraud accounts that escaped the transaction time evaluation from only 28 extracted fraud communities. For comparison, the baseline system only captures 19 from 200 communities.

2 RELATED WORK

In this section, we will first review related work on graph-based fraud detection, followed by *local graph clustering*, the key technology used in our community detection. Finally, we will review work on the graph-based semi-supervised classification.

2.1 Graph-based Fraud Detection

Many graph-based fraud detection methods focus on abnormality detection in social media, such as detecting fake followers, or fake reviews, based on the fact that abnormality usually emerges from unusual high-density connections. There are roughly two categories of these methods: global and local.

The global methods model on the entire graph for fraud detection. The singular value decomposition (SVD) based method SPOKEN [31] uses plots in the space spanned by paired eigenvectors to identify abnormalities. The HITS [21]-like methods CatchSync [19] uses random-walk to catch synchronized behavior. FRAUDAR [17] finds dense blocks through optimizing a density metric. While these methods achieve good performance on graph abnormality detection through densely connected blocks, they do not fit into our design requirements for: 1) not accepting input vertex labels; 2) accepting input labels but may return blocks too large for human experts; 3) not providing risk scores on individual vertex. ZooBP [7] estimates node riskiness through fast and scalable belief-propagation (BP), but does not directly output small subgraphs. Another BP-based method NetProbe [28] may meet all these requirements, but it has a strong assumption on three types of accounts (fraud/accomplice/honest) of online auction, and whether that type assumption also holds for payment fraud is still questionable.

The local methods analyze local graph centered around suspicious vertices. The OddBall [2] analyzes the egonet through carefully designed local graph features focusing on near-cliques and stars. The CopyCatch [5] finds local dense bipartite subgraphs with temporally synchronized behavior, and thus requires time stamps.

The GraphRAD in this paper is a mix of global and local treatments. While candidate communities are initially found by the community detection module globally, they are further filtered and analyzed locally before presented to human experts.

We list in Table 1 the major differences between GraphRAD and existing methods.

2.2 Community Detection

Almost all graph-based fraud detection methods seek to find unusual densely connected subgraphs, but they vary in definition of "dense connection". In this section, we will review the definitions from community detection approach, along with popular algorithms in the field.

2.2.1 Community and Metrics. Community detection and graph clustering are similar concepts used in different research communities. In this paper, community and subgraph, as well as community detection and subgraph clustering, are interchangeably used. Specifically, community is defined as a group of vertices having denser

Table 1: Differences between GraphRAD and existing methods.

	SPOKEN	CopyCatch	CatchSync	FRAUDAR	NetProbe	ZooBP	OddBall	GraphRAD
Input labels		✓	✓		✓	✓	?	✓
Output small subgraphs		?			?		✓	✓
Non-overlapping subgraphs	✓	✓	✓	✓	?		?	✓
No assumption on vertex types	✓	✓	✓	✓		✓	✓	✓
No need of temporal input	✓		✓	✓	✓	✓	✓	✓
Scores for vertices		?	✓		✓	✓		✓

connections among each other compared to connections with other parts of the graph. Community detection is then the process of finding the communities. There are two major community detection approaches: modularity-based methods [26] and conductance-based methods [18], where accordingly *modularity* and *conductance* serve as the community definition metrics. With *conductance* being a local metric, and *modularity* being a global metric, small *conductance* or large *modularity* indicates a dense community well separated from the rest part of the graph.

In GraphRAD, we chose *conductance* as the community metric since we are interested in extracting small-scale local communities. The *conductance* $\phi(S)$ of a community S is defined in Equation 1:

$$\phi(S) = \frac{\sum_{i \in S} \sum_{j \in S^c} A_{ij}}{\min\left(\sum_{i \in S} d_i, \sum_{i \in S^c} d_i\right)}. \tag{1}$$

where A is the graph adjacent matrix, d_i is the degree of vertex i. Here community detection methods with *conductance* as the metric are known as local community detection.

2.2.2 Local Community Detection. Local community detection, or *local graph clustering*, is a semi-supervised method, which returns a cluster of vertices given a single or a list of seed vertex inputs. Specifically, *local community detection* is a way of extracting small-scale communities with small (i.e., good) *conductance* through regularizing the graph with sparsity operations [10].

One of the most important advantages of local community detection is that there exist simple algorithms for which the running time depends on the volume of the output community instead of the volume of the input graph [3]. Practically this means that if the solution of the regularized graph problem is a small community, the running time of the algorithm will depend on the volume (sum of outgoing edges per vertex) of the small output community.

Local community detection methods have been proven particularly powerful in identifying small-scale structure in large graphs [18, 22], and therefore they are chosen as the primary algorithms in GraphRAD. There are a variety of related local community detection algorithms, each one with slightly different pros and cons.

We refer readers to [10] for a survey of methods for the local community detection problem. These algorithms can be generally characterized into two major categories: spectral-based and flow-based [10]. Spectral-based methods have the advantage that they discover small communities each consisting of a single connected component [14, 22, 32, 38]. The flow-based algorithms, on the other hand, discover communities with small *conductance* [22, 27], but tolerate disconnected components within a community [22].

GraphRAD chose the spectral-based methods [3] since it makes more sense to present a single connected subgraph to human experts, and call that a fraud community.

2.3 Graph-based semi-supervised classification

If feature vectors are provided on vertices, and labels are partially known for some vertices, the semi-supervised classification methods can be applied to predict labels (score likelihood) for rest of the vertices, with assumptions conditioned on the graph structure. Here we review three approaches based on different assumptions.

2.3.1 Skip-gram. The *skip-gram* technique was introduced to learn the semantic embedding of English words, by maximizing the likelihood of observing related words appearing in the context window from a given word [24, 25]. The *skip-gram* architecture was later extended to graph embedding, where it maximizes the likelihood of observing vertices within the random-walk range starting from a given vertex. Different choices of random-walk lead to different methods such as *DeepWalk*[30] and *node2vec*[13]. The random-walk can also be reduced to pure k-hops neighbors as in Tang et al. [33]. By adding a supervised term into the objective function, the multi-task learning method learns both vertex embedding and semi-supervised classifier at the same time [34].

2.3.2 Graph Convolutional Neural Network. Recent studies in deep learning have extended the convolutional neural network from image domain to the non-Euclidaen domains such as graph [6, 16, 20]. With the redefined convolution operation on the spectral domain of graph, the graph convolutional network layer can be stacked with other nonlinear operations [16]. Defferrard et al. [6] used K-localized filters to replace the spectral convolution operations, making significant speed-up on the feed-forward operation. Kipf et al. [20] made even more simplification by using linear first-order approximations. Like the multi-dimensional "input channels" in traditional convolutional neural network, the input of graph convolutional neural network could also be multi-dimensional on vertices, making it easy to be extended as a semi-supervised problem with incorporated vertex feature vectors.

2.3.3 Graph Regularization. The graph regularization methods assume neighbors in graph having more shared commonality, and penalize the differences between neighbouring vertices [1, 23]. One way of such penalization is to insert a graph *Laplacian* regularization term into the supervised loss, thus making it a semi-supervised method [36, 37]. For parameter estimation, Zhu et al. [37] and Zhou et al. [36] proposed diffusion-based learning algorithms that solve alternative linear systems directly through matrix operations. Gleich et al.[12] formed the diffusion-based learning problem as an optimization problem, and added ℓ_1 regularization term to obtain a more robust solution, while a local push algorithm as in [4] was introduced to calculate the local solution.

In GraphRAD, we inserted the graph regularization term after a supervised logistic loss on vertex feature vectors. Similar to Zhang et al. [35], the logistic loss assumes shared feature weights over the entire graph, along with the graph-regularized vertex-dependent biases.

3 PROPOSED SYSTEM ARCHITECTURE

In this section, we describe the system architecture of GraphRAD (Figure 1).

Transaction Record: The Transaction Record is a data storage service that keeps records of information such as transaction time, account id, and attributes like shipping address, etc. It also keeps records of the transaction evaluation results from the *Risk Detection System.* Note that these are real-time evaluations at the transaction time, and certainly may contain false negatives. The service is available to provide query service for transactions spanning a given period of time.

Graph Generator: The Graph Generator module takes transactions spanning a given period of time from the Transaction Record, and construct the account link graph (referred to as the "big-graph") through reduce operation by account ids. The graph is constructed such that any two vertices (accounts) were connected if they share at least one link attribute in the same transaction. A heuristic degree filter is applied where vertices with very large degrees are removed. Another set of heuristic rules are applied to the transaction evaluation records to generate three types of labels for every account in the graph: *Trusted*, *Fraud*, and *Risky*. While it is possible to construct a single graph from the entire bulk of transactions, it is practical to include only the most recent transactions, both for the computational efficiency, and for the observation that recent activities weigh most in future prediction.

Seeding: The Seeding module uses a set of heuristic rules to pick up seed vertices (accounts) for the Community Detection module. The seed vertices include all *Fraud*-labeled and a selected portion of the *Risky*-labeled vertices. This module gates number of seed accounts fed into the downstream module by putting a heuristic riskiness threshold, using risk indicator features not described in this paper.

Community Detection: The Community Detection module takes in the seed vertices provided by Seeding module, and performs the *local graph clustering* algorithm on the "big-graph". The outcome are a list of subgraphs (communities) each centered around a seed vertex. As we assume that fraudsters tend to use accounts linked somehow, these communities should contain undiscovered accounts owned by fraudsters.

Screen + Merge: To control the scale of final communities presented to human experts, a regression model trained on community statistics is used to rank the community riskiness, keeping only the risky ones. The screened communities are then merged into a supergraph referred to as the "big-comm".

Community Extractor: This module clusters fraudster accounts and locally extracts community for each cluster. It extracts approximately non-overlapping small fraud communities from the "big-comm". Together with the account risk scores from the Scoring module described later, these small communities are presented to human experts. This

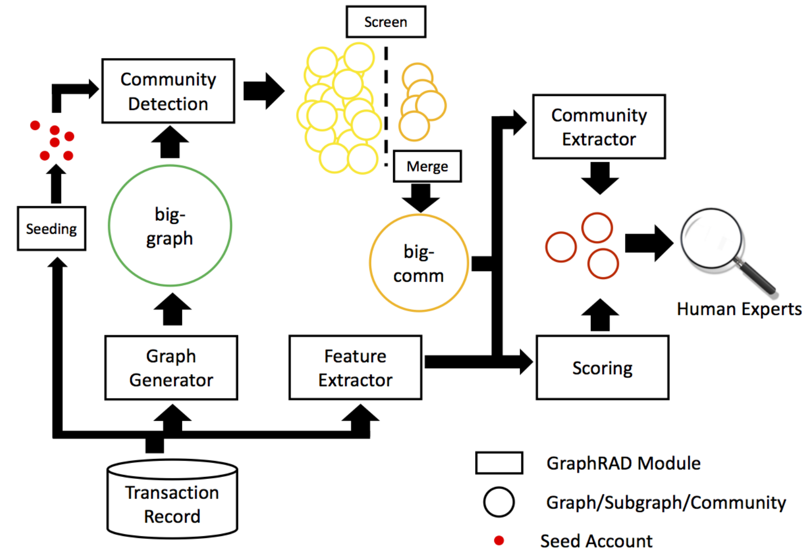


Figure 1: GraphRAD: System Architecture

module is necessary because: 1) it is easier for human experts to scrutinize small graphs, 2) a list of small graphs are easy to be allocated between human experts, and 3) the "non-overlapping" treatment avoids repeated checks.

Feature Extractor: This module generates the feature vector for each vertex (account) of the "big-comm" using the attributes stored in the Transaction Record. Note that these attributes are different from those used to build the "big-graph", but instead a set of transaction risk indicators not described in this paper.

Scoring: The Scoring module takes features from the Feature Extractor, and performs a semi-supervised classification regularized by the "big-comm" graph. The outcome are risk scores assigned to each vertex (account) of the "big-comm".

The overall system architecture is shown in Figure 1. Circles are colored in green, yellow, orange, and red, reflecting the increasing fraud risk enrichment of graphs/communities after modules.

4 PROPOSED METHODS

In this section, we describe the detailed algorithms used in Community Detection, Community Extractor, and Scoring modules.

4.1 Community Detection: *local graph clustering*

In the GraphRAD system, the community detection problem is formed as finding a subgraph through graph random walks starting from the seed, with the subgraph meeting certain criteria, e.g., minimized *conductance*. Given a graph $G = \{V, E\}$, the graph random walk can be performed through personalized page rank (*PPR*) iterations. In Equation 2, we denote the *PPR* vector $p \in \mathbb{R}^{|V|}$ initialized by $s \in \mathbb{R}^{|V|}$, whose elements are 1 if index in S, and 0 otherwise. S is the set of starting vertices (seeds); $W \in \mathbb{R}^{|V|\times|V|}$ is the lazy random walk matrix; $D \in \mathbb{R}^{|V|\times|V|}$ is the diagonal degree matrix; $A \in \mathbb{R}^{|V|\times|V|}$ is the adjacent matrix; and α controls locality of the random walk, with increasing α leading to more localized random walk. Once convergence is reached, elements of *PPR* vector would correspond to the probability of each vertex appearing in random walk paths starting from seeds in S.

$$\begin{aligned} p &= \alpha s + (1-\alpha)pW \\ W &= \frac{1}{2}(I + D^{-1}A) \\ s_i &= 1, i \in S \\ s_i &= 0, i \notin S \end{aligned} \tag{2}$$

In practice, instead of running *PPR* iterations for convergence, fast approximate methods are widely used. One popular method forms the problem of obtaining *PPR* vector as solving a linear system, and uses a specialized iterative scheme along with early stopping criterion. *ACL* algorithm (R. Andersen, F. Chung and K. Lang)[4], one of the first such methods, solves the linear system by optimizing an equivalent problem through coordinate descent.

In addition to α, the parameter ρ in *ACL* corresponding to an ℓ_1 regularization term controls sparsity of the solution [9–11]. Here we chose $\alpha = 0.15$, and $\rho = 1e-6$.

ACL algorithm is a strongly local algorithm with running time depending only on the size of the output community, therefore it is a scalable method for large graphs in GraphRAD. Here in the Community Detection module, we chose *ACL* with metrics using *conductance* to obtain the *PPR* vector. We treat each of the N seeds provided by the Seeding module independently, and run *ACL* for each of them with s initialized as one-hot vector every time, ending up with N *PPR* vectors. Finally, *Sweep-cut* algorithm [4, 10] is applied on each of the *PPR* vectors to obtain the partition that yields lowest *conductance*, i.e., local community.

4.2 Community Extractor: *hierarchical clustering + local graph clustering*

The Community Extractor module needs to extract small-scale, "risk-rich", and non-overlapping communities from the "big-comm" for human check. For this problem, we could directly run the *local graph clustering* algorithm as in the Community Detection Module, provided with *Fraud*-labeled accounts as seeds. However, this would not guarantee the communities being non-overlapping, and would lead to repeated tasks for human experts. For example, if two *Fraud*-labeled accounts exist in the same community, and both of them were in the seed list, this community will be found twice after *local graph clustering*. Therefore, an efficient mechanism is necessary to remove such redundancy.

Here we propose using *hierarchical clustering* before running the *local graph clustering*, as in Algorithm 1. Starting with a list of *Fraud*-labeled vertices from the "big-comm", the *hierarchical clustering* is performed on the *PPR* vector of these vertices, returning clusters of *Fraud*-labeled accounts. Since *PPR* vector represents the destination distributions of random walks starting from a vertex, the distance between two *PPR* vectors indicates how close they are in the graph. Therefore the *PPR*-distance-based cluster would contain *Fraud*-labeled accounts that are likely to be in one community. We set 2000 to be the maximum allowed cluster numbers. In the next step, we remove clusters of size less than L ($L = 5$), because *Fraud*-labeled accounts in these small clusters are less connected with others, thus less likely to form risk-rich communities. This step is yet another control of how many communities are finally presented to human experts.

Next we keep a set of vertices called *touched fraud*. For each cluster, we take their vertices as seeds, and remove vertices already seen in the *touched fraud* set. This removal operation helps reducing community overlapping if not completely eliminating it. The seeds were used to initialize the *PPR* vector for the following *ACL*, with indices of set to 1 if in seeds, and 0 otherwise. Running such an *ACL* on "big-comm" returns one local fraud community. In the meanwhile, We keep adding new *Fraud*-labeled vertices into the *touched fraud* set. Finally, we return all extracted local fraud communities.

Note that it is unfeasible to apply the Community Extractor module directly to the "big-graph", mainly for the following two reasons: 1) *PPR* vector has size equivalent to the number of vertices in the graph, which makes computing vector distances during clustering cursed by the million size dimension of "big-graph", whereas the "big-comm" has a much lower dimension (fewer vertices) after the community screening; and 2) the dangling *Fraud*-labeled vertices (*Fraud*-labeled accounts that are not connected with any others) in the big-graph would be each extracted as a standalone local fraud community, which however, degrades the proposal quality for human experts. The dangling vertices, however, do not exist in the "big-comm".

4.3 Scoring: *graph-based semi-supervised classification*

In addition to the local fraud communities presented to human experts, the Scoring module provides risk scores for accounts in these communities, based on which human experts can conveniently prioritize their tasks. We assume accounts linked closer

Algorithm 1 Hierarchical clustering + Local graph clustering

```
Input:
graph: input graph,
maxcluster: maximum number of clusters,
L: minimum size of cluster
Output: local fraud communities
  local fraud communities ⟵ []
  vectors ⟵ getPPRVectorOfFraudLabeledVertices(graph)
  Clusters ⟵ Hierarchical Clustering(vectors, maxcluster, L)
  touched fraud ⟵ {}
  for do each cluster
      seed ⟵ cluster \ touched fraud
      local fraud comm ⟵ LocalGraphClustering(graph, seed)
      touched fraud += fraud in local comm
      local fraud communities += local fraud comm
  end for
```

to the fraud-labeled are more likely to share the riskiness. On the other hand, risk indicator features from Feature Extractor module also independently tell riskiness of an account. Together, given that each community must contain certain at least one *Fraud*-labeled account (from the seeds) but unlikely all, we form the risk scoring problem as a *graph-based semi-supervised classification* problem, and considered 3 approaches: *node2vec*, *graph convolutional neural network (GCNN)*, and the *graph-regularized local logistics regression (GLR)*.

4.3.1 node2vec. Following the *skip-gram* architecture in the work of *word2vec* [25], the *node2vec* [13] method learns representation of vertices in a graph, such that the conditional probability of observing the neighbors $N_s(u)$ of a vertex(node) u is maximized. With the assumptions of conditional independence and symmetry in feature space as in *node2vec* work, the unsupervised loss function L_u takes the form of:

$$\begin{aligned} L_u &= \sum_{u \in V} \log P(N_s(u)|f(u)) = \sum_{u \in V} \sum_{n_i \in N_s(u)} P(n_i|f(u)) \\ &= \sum_{u \in V} [-\log \sum_{v \in V} \exp(f(u)^T f(v)) + \sum_{n_i \in N_s(u)} f(n_i)^T f(u)] \end{aligned} \tag{3}$$

with $f(u)$ mapping vertex u to its representation.

To make it a semi-supervised loss, we added the supervised part as in Equation 4. L_s follows the transductive formulation as in [34], where label y_u of a vertex u depends on the concatenated vector of feature vector x_u and the representation $f(u)$. The w are model parameters.

$$\begin{aligned} L &= L_u + L_s = L_u + \sum_{u \in V} \log P(y_u | x_u, f(u)) \\ &= Lu + \sum_{u \in V} \log \frac{\exp[(x_u)^T, f(u)^T] w_{y_u}}{\sum_{\bar{y}} \exp[(x_u)^T, f(u)^T] w_{\bar{y}}} \end{aligned} \tag{4}$$

The per vertex partition function $\sum_{v \in V} \exp f(u)^T f(v)$ in Equation 3 is expensive to compute, and it was approximated using negative sampling like the *node2vec* work. The $N_s(u)$ in Equation 3 is not limited to the direct neighbors of u, and is typically sampled either by including neighbors within k-hop sphere [33] or various flavors of random-walk [13, 30]. Due to the real-time running efficiency, random-walk is usually performed separately before optimizing the loss function. In GraphRAD system, we directly took the *PPR* vector of u from previous steps, and sample $N_s(u)$ according to the random-walk distributions in the *PPR vector* [8]. The remaining parameters in the loss function are optimized using standard stochastic gradient descent method.

4.3.2 graph convolutional neural network (GCNN). Similar to the multiplication of image pixel intensities and kernels, the graph convolution operation is defined as multiplication of a signal $x \in \mathbb{R}^{|V|}$ (one scalar for every vertex) and a filter g_θ. In spectral graph convolution, g_θ is constructed from the eigenvectors of the graph Laplacian L, which however, is time-consuming for both the eigenvalue decomposition, and the multiplication of dense matrices at evaluation time ($O(|V|^2)$). In *GCNN*[6], the Chebyshev polynomials Kth-order approximation was introduced to achieve $O(|E|)$ evaluation time, as shown in Equation 5.

$$g_\theta \star x = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})x \tag{5}$$

where $\tilde{L} = \frac{2}{\lambda_{max}}L - I_{|V|}$, λ_{max} is the largest eigenvalue of L, $T_k(\tilde{L})$ is the k-th Chebyshev polynomial of $\tilde{L}$, and θ_k is the k-th polynomial coefficient. Note that K corresponds to number of steps away from the center vertex.

Now let $X \in \mathbb{R}^{|V| \times D}$, with columns being x of a individual feature channel, and D being total feature dimension, and let $\theta^{(d)}$ be filter parameters of channel d, we have the prediction of

$$Z = \sum_{d=0}^{D} g_{\theta^{(d)}} \star X_{:,d} \tag{6}$$

, and for vertices u with labels y_u, we minimize the cross-entropy loss of:

$$L = -\sum_{u \in V} y_u log(Z_u) + (1 - y_u)(1 - log(1 - Z_u)) \tag{7}$$

4.3.3 graph-regularized local logistic regression (GLR). In this approach, we model the label of a vertex by a logistic regression of vertex label y_i on vertex feature x_i, $i \in V$ (Equation 8), where weight W is shared across the entire graph, and the bias b is vertex-dependent.

$$Q(\theta; x, y) = \sum_{u \in V} \log\left(1 + e^{-y_u(Wx_u + b_u)}\right) + \lambda \sum_{(u,u') \in E} (b_u - b_{u'})^2 \tag{8}$$

For parameter estimation, we adapted the Alternating Direction Method of Multipliers (ADMM) algorithm used by the network lasso work [15]. Similar to the work of Zhang et al.[35], we introduced a local copy of W on each vertex, a copy of b_i on edges connected with vertex i, and rewrite the loss function with augmented *Lagrangian* such that parameter copies are penalized by their distances from W and b_i. The loss with augmented *Lagrangian* is then optimized through a coordinate descent style procedure [35]. The optimization steps can be easily parallelized by distributing vertices and edges separately.

5 EXPERIMENTS

5.1 Data Pipeline

The Graph Generator module loaded historical transaction records spanning a long enough period of time. Accounts in these transactions were first aggregated onto link indicative attributes (e.g., shipping address), then edges were created between every pair of accounts, if they co-occur in at least one aggregates. In this way, the "big-graph" was generated such that accounts are represented by vertices, and shared connectivity are represented by unweighted edges. The Graph Generator module was implemented using Apache Spark running on Amazon Web Services EMR (Elastic Map Reduce) instances, generating a "big-graph" of over 60 million vertices, and 500 million edges.

Transactions were evaluated by *Risk Detection System* (not described in this paper) when orders were placed by customers. From the historical evaluation results, along with reported fraud attacks, the Graph Generator module uses a set of heuristic rules to assign three types of account labels: *Trusted*, *Fraud*, and *Risky*. Specifically, *Trusted* labels are assigned to long-established accounts without any reported fraud; *Fraud* labels are assigned to reported accounts confirmed with fraud actions; *Risky* labels are assigned to accounts showing risky indicators but not confirmed as fraud.

The transaction records were chronologically split into training records and testing records. For parameter estimation and tuning, we only use labels of accounts appearing in the training records, while performance was evaluated on accounts appearing in the testing records. In testing we paid special attention to the "missed bad": the *Fraud*-labeled accounts that were not evaluated as risky by the *Risk Detection System.*

In our experiment, the Community Detection module was provided with a list of 90401 *Risky*-labeled accounts as seeds, returning 71074 local communities. Note that not all seeds successfully yielded community because of the *conductance* requirement.

These communities are selected subgraphs that GraphRAD system will focus on. However, 70K is a much too large number for human check consumption. The Screen module uses a regression model trained on community statistics (using risk indicators calculated from Transaction Record, but nothing related to graph topology) to filter the communities, and the Merge module merges the filtered communities into "big-comm". The "merge" step is necessary because the semi-supervised Scoring module will build features regularized by the "big-comm" instead of by local communities, such that a wider range of relations are taken into count. In our experiment, the "big-comm" contains 24393 vertices and 351741 edges.

5.2 Community Extractor

After running Algorithm 1 on the "big-comm", we detected 28 fraud communities with sizes ranging from 15 to 500. In total, these 28 communities contain 4433 accounts, among which are 67 "missed bad" out of 1899 *Fraud*-labeled accounts (Figure 2, all communities combined). For easy understanding and operation distribution, the communities are presented to human experts separately (Figure 3).

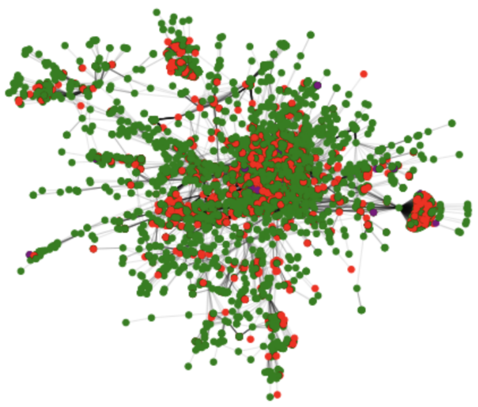


Figure 2: Graph visualization of all proposed communities combined. Vertices are colored as (1) green: *Trusted*, (2) red: *Fraud*, (3) purple: "missed bad".

We observed that some "missed bad" accounts are not linked with a *Fraud*-labeled account within one hop range, but linked with a *Trusted*-labeled account first, and then a *Fraud*-labeled account at the second hop (see arrows in Figure 3 a, b). Since the "missed bad" were once trusted, and escaped the *risk detection system*, it suggests that some of the *Trusted*-labeled accounts (circled in Figure 3 a, b, and c) in these communities are highly risky, maybe playing similar roles as the accomplice users mentioned in the NetProbe work[29]. Therefore it is reasonable to add these "trusted" accounts into a special watch list.

5.3 Scoring

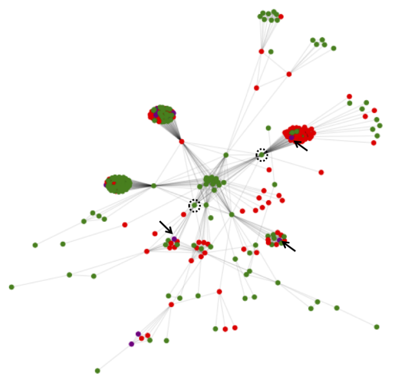
The local community presented to human experts could still contain hundreds of accounts, and there are needs of a risk scoring method to help prioritize the human working schedule. For this purpose, the Scoring module uses semi-supervised learning algorithm to provide scores for every vertex in all the proposed local communities. In this work, we tried 3 approaches: *node2vec*, *graph convolutional neural network (GCNN)*, and *graph-regularized logistic regression (GLR)*, and used the best-tuned parameters for each of the three.

The semi-supervised algorithms were implemented using TensorFlow (https://www.tensorflow.org/), and were trained using risk indicator features from the Feature Extractor, account links from the "big-comm", along with *Fraud* labels from the training records. The performance was evaluated using labels from the testing records.

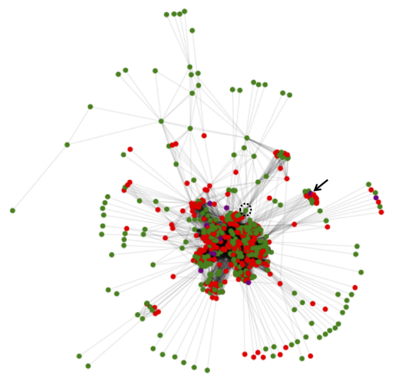
As shown in Figure 4, *GLR* (*AUC* = 0.736) achieves significantly better performance compared to *GCNN* (*AUC* = 0.699) and *node2vec* (*AUC* = 0.680). Note that *GLR* also has a higher true positive rate at low false positive region (< 0.2), which is closer to the business practice. Additionally, as a simpler model, *GLR* is less prone to over-fitting, and was also found in our experiments using less training time than the other two.

5.4 GraphRAD End-To-End Performance

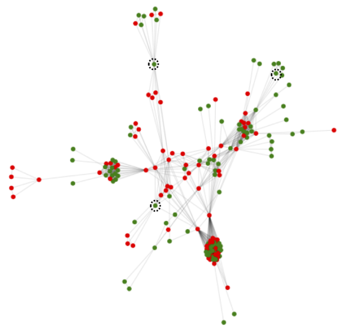
Following the GraphRAD architecture (Figure 1), communities started from the "big graph", an unprocessed pool within which



(a) extracted local community of 294 accounts, 119 *Fraud*, 13 "missed bad"



(b) extracted local community of 704 accounts, 306 *fraud*, 13 "missed bad"



(c) extracted local community of 177 customers, 81 *fraud*, 0 "missed bad"

Figure 3: Examples of communities extracted by Community Extractor. Vertices are colored as (1) green: *Trusted*, (2) red: *Fraud*, (3) purple: "missed bad". Arrows point to "missed bad" not directly connected to *Fraud* accounts. Circles highlight the suspicious *Trusted*.

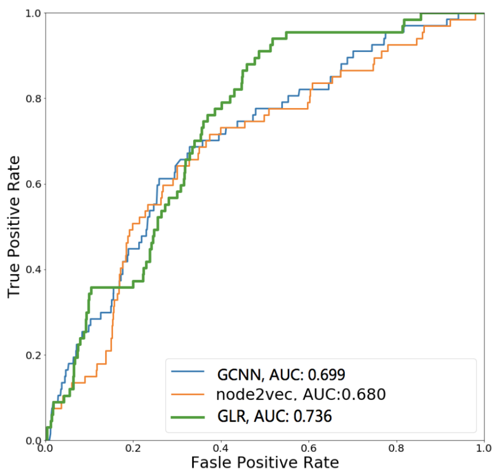


Figure 4: Receiver Operating Characteristic Plot for Testing Data. Models used: (1) *GCNN*, (2) *node2vec*, (3) *GLR*

Table 2: End-To-End Performance Comparison

method	"missed bad"	communities	ratio
baseline	19	200	0.095
GraphRAD	67	28	2.393

0.017% accounts for the "missed bad". After Community Detection and Screen modules, within the "big comm" there are 0.054% "missed bad". After Community Extractor module, within final 28 communities there are 1.51% "missed bad". With semi-supervised *GLR* model, the number further increased to 2.77%, which is overall 163 times of end-to-end "missed bad" enrichment. The enrichment will significantly reduce the amount of work by human experts. While the percentage of "missed bad" is used as an estimate of the effectiveness, it only serves as a lower bound of the fraud detection performance, since some "trusted" accounts will be questioned once presented to human experts.

We also compared GraphRAD with a simple one-hop neighborhood approach (baseline), where the communities are extracted simply as vertices one-hop away from the seed accounts (Table 2). The baseline was able to capture 19 "missed bad" from 200 proposed communities. As a comparison, the more effective GraphRAD was able to capture 67 "missed bad" from 28 proposed communities, which would significantly reduce the amount of required human labor.

5.5 Scalability

All modules in GraphRAD are implemented with scalability in consideration. Running time of the *local graph clustering* algorithm used in Community Detection and Extractor module is proportional to the output community volume rather than the input "big-graph". The running time of ADMM in Scoring module is $O(E+V)$, and the distributed algorithm is easy to parallel. The Graph Generator and Merge modules are implemented in Apache Spark for easy scaling up. The running time of hierarchical clustering is $O(n^2V)$, with n being the number of *Fraud-labeled* accounts within *big-comm*, and magnitudes smaller than V.

6 CONCLUSIONS

In this paper, we propose GraphRAD, a graph-based risk account detection system. The system takes in a list of risky seedling accounts, and output a short list small-scale, non-overlapping communities containing highly suspicious accounts. Our experiments on real business data have shown that from an account linkage graph of 60 million vertices and 500 million edges, the system detected 67 fraud account that escaped the real-time risk evaluation by proposing 28 communities. Such a short list of small-scale fraud communities could significantly reduce the workload of human experts. We need to emphasize that GraphRAD works as a final stage gatekeeper complimentary to the main *Risk Detection System* which catches the vast majority of frauds. Any detected missing fraud accounts are additional gains.

We also observed that a *graph-regularized logistic regression* method outperformed *node2vec* and *convolutional* neural network in the semi-supervised fraud prediction task.

For our workflow requirements, most state-of-the-art fraud detection methods cannot be applied directly without non-trivial modifications. Therefore, GraphRAD is the first published method tackling such a practical problem. To our knowledge, it is also the first published method applying *local graph clustering* into the payment fraud detection problems.

With these observations, in the future, we will work on integrating insights from this work into our production system.

ACKNOWLEDGMENTS

We thank Christos Faloutsos (Carnegie Mellon University), for comments that greatly improved the manuscript. We thank Michael W. Mahoney and Kimon Fountoulakis (University of California, Berkeley), for the help and suggestions. We also thank Xin Luna Dong (Amazon) and other Amazon colleagues, for support of the work.

REFERENCES

[1] J. Abernethy, O. Chapelle, and C. Castillo. 2010. Graph regularization methods for web spam detection. *Mach. Learn.* 81, 2 (2010), 207–225.
[2] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. oddball: Spotting Anomalies in Weighted Graphs. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II.* 410–421. https://doi.org/10.1007/978-3-642-13672-6_40
[3] R. Andersen, F. Chung, and K. Lang. 2006. Local graph partitioning using pagerank vectors. *FOCS '06 Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science* (2006), 475–486.
[4] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06).* IEEE, 475–486.
[5] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013.* 119–130. http://dl.acm.org/citation.cfm?id=2488400
[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems.* 3844–3852.
[7] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. 2017. ZooBP: Belief Propagation for Heterogeneous Networks. *PVLDB* 10, 5 (2017), 625–636. http://www.vldb.org/pvldb/vol10/p625-eswaran.pdf

[8] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. 2017. LASAGNE: Locality And Structure Aware Graph Node Embedding. *arXiv preprint arXiv:1710.06520* (2017).

[9] Kimon Fountoulakis, Xiang Cheng, Julian Shun, Farbod Roosta-Khorasani, and Michael W Mahoney. 2016. Exploiting Optimization for Local Graph Clustering. *arXiv preprint arXiv:1602.01886* (2016).

[10] Kimon Fountoulakis, David F. Gleich, and Michael W. Mahoney. 2017. An Optimization Approach to Locally-Biased Graph Algorithms. *Proc. IEEE* 105, 2 (2017), 256–272. https://doi.org/10.1109/JPROC.2016.2637349

[11] D. F. Gleich and M. M. Mahoney. 2014. Anti-differentiation approximation algorithms: a case study with min-cuts, spectral, and flow. *Proceedings of the International Conference on Machine Learning (ICML)* (2014), 1018–1025.

[12] David F Gleich and Michael W Mahoney. 2015. Using local spectral methods to robustify graph-based learning algorithms. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 359–368.

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[14] S. Guattery and G. L. Miller. 1998. On the quality of spectral separators. *SIAM J. Matrix Anal. Appl.* 19 (1998), 701–719.

[15] David Hallac, Jure Leskovec, and Stephen Boyd. 2015. Network lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 387–396.

[16] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[17] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 895–904. https://doi.org/10.1145/2939672.2939747

[18] Lucas GS Jeub, Prakash Balachandran, Mason A Porter, Peter J Mucha, and Michael W Mahoney. 2015. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Physical Review E* 91, 1 (2015), 012821.

[19] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. CatchSync: catching synchronized behavior in large directed graphs. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. 941–950. https://doi.org/10.1145/2623330.2623632

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[21] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632. https://doi.org/10.1145/324133.324140

[22] J. Leskovec, K. L. Lang, A. Dasgupta, and M. W. Mahoney. 2009. Community structure in large networks: natural cluster sizes and absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[23] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. 2011. Recommender systems with social regularization. *WSDM '11 Proceedings of the fourth ACM international conference on Web search and data mining* (2011), 287–296.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[26] Mark EJ Newman. 2006. Modularity and community structure in networks. *Proceedings of the national academy of sciences* 103, 23 (2006), 8577–8582.

[27] L. Orecchia and Z. A. Zhu. 2014. Flow-based algorithms for local graph clustering. *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), 1267–1286.

[28] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. 201–210. https://doi.org/10.1145/1242572.1242600

[29] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 201–210.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[31] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*. 435–448. https://doi.org/10.1007/978-3-642-13672-6_42

[32] D. A. Spielman and S. H. Teng. 1996. Spectral partitioning works: planar graphs and finite element meshes. *In Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science* (1996), 96–107.

[33] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[34] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[35] Danqing Zhang, Kimon Fountoulakis, Junyu Cao, Mogeng Yin, Michael Mahoney, and Alexei Pozdnoukhov. 2017. Social Discrete Choice Models. *arXiv preprint arXiv:1703.07520* (2017).

[36] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in neural information processing systems*. 321–328.

[37] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 912–919.

[38] Z. A. Zhu, S. Lattanzi, and V. Mirrokni. 2013. A local algorithm for finding well-connected clusters. *Proceedings of the 30th International Conference on Machine Learning* (2013), 396–404.