Spade: A Real-Time Fraud Detection Framework

Jiaxin Jiang
National University of Singapore
jxjiang@nus.edu.sg

Zhen Zhang
National University of Singapore
zhen@nus.edu.sg

Bingqiao Luo
National University of Singapore
luobq@nus.edu.sg

Bingsheng He
National University of Singapore
hebs@nus.edu.sg

Min Chen
GrabTaxi Holdings
min.chen@grab.com

WeiYang Wang
GrabTaxi Holdings
weiyang.wang@grab.com

Jia Chen
GrabTaxi Holdings
jia.chen@grab.com

ABSTRACT

In this demonstration, we introduce Spade, a sophisticated real-time fraud detection framework adept at navigating the complex transaction graph. Unlike conventional methods that are limited by performance and lack incremental update capabilities, Spade leverages advanced incremental updates in dense subgraph peeling algorithms to enhance efficiency, usability, and reduce latency, achieving a significantly better fraud prevention ratio. The demo showcases an interactive GUI prototype, allowing users to customize and explore dense subgraphs with various metrics and algorithms. This interactive demonstration also effectively highlights Spade's robust capacity to unearth fraudulent transactions within varied settings, including Grab's services and cryptocurrency transactions.

PVLDB Reference Format:
Jiaxin Jiang, Zhen Zhang, Bingqiao Luo, Bingsheng He, Min Chen, WeiYang Wang, and Jia Chen. Spade: A Real-Time Fraud Detection Framework. PVLDB, 17(12): 4253 - 4256, 2024.
doi:10.14778/3685800.3685848

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/samjjx/Spade.

1 INTRODUCTION

Fraud detection is pivotal in online marketplaces, where malicious activities can precipitate substantial financial losses. The rapid evolution of social and transaction graphs in these domains intensifies the challenge of timely fraud identification. Traditional peeling algorithms, lauded for their efficiency and theoretical guarantees, face limitations due to their static nature and inability to adapt to evolving graphs. Spade, leveraging advancements in real-time analytics, overcomes these obstacles by offering an incremental approach to graph analysis, significantly reducing latency and enhancing the detection of fraudulent activities within narrow timeframes. This adaptability is crucial for platforms like Grab, where transaction graphs evolve swiftly, necessitating immediate responses to potential fraud.

Our motivation stems from the critical need for real-time fraud detection in transaction graphs, highlighted by our collaboration with Grab. The fraud detection process at Grab, depicted in Figure 1, confronts challenges like customer-merchant collusion, where promotions are exploited through fake accounts and transactions, forming dense subgraphs. Leveraging our prior work with Spade [8], we've developed an incremental algorithm framework that enhances the capability for real-time detection of such fraud within dense subgraphs, accommodating various definitions of density. This approach significantly advances the efficiency and responsiveness of fraud detection mechanisms in dynamic transaction environments.

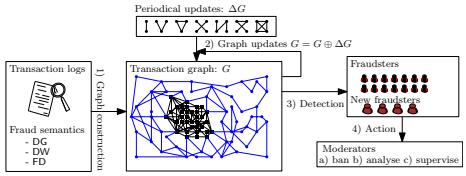


Figure 1: Common Data Pipeline for Fraud Detection

In this demo paper, we introduce an interactive system prototype featuring a GUI that builds upon our previous research [8]. This system empowers users to tailor and apply various density metrics and peeling algorithms for constructing and analyzing dense subgraphs. It's designed to enhance fraud detection capabilities across real-world datasets, including those from industry and cryptocurrency sectors. The demonstration will highlight specific features of this system, showcasing its practical application and effectiveness in identifying fraudulent activities through an intuitive and customizable interface:

(1) *Ease of Use.* Spade provides a user-friendly approach, allowing developers to incorporate their suspiciousness functions for detecting fraudulent communities. It offers a comprehensive suite of APIs for customizing fraud detection semantics. Additionally, its GUI supports coding, result visualization, and integrates a code generation tool for the APIs.

(2) *Automated Incrementalization.* Spade simplifies the incrementalization process across various peeling algorithms, concealing the complexities from the developers. This feature empowers developers to create custom fraud detection semantics through specific edge and vertex suspiciousness functions.

(3) *High Efficiency.* With data sets from Grab and cryptocurrencies, we will demonstrate that Spade greatly accelerates the fraud detection process, with speed-ups of up to six orders of magnitude.

With Spade, users can explore transaction graphs and frauds in an interactive and real-time manner. For example, as the dense subgraph

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685848

is highlighted, users can zoom in/out to examine the pattern. Another example is, users can specify the nodes of interest and explore its structure interactively. We will showcase the application of Spade in identifying fraud within both Grab and cryptocurrency scenarios.

2 THE Spade FRAMEWORK

2.1 Preliminary

Graph G**.** We consider a directed and weighted graph $G = (V, E)$, where V is a set of vertices and $E \subseteq (V \times V)$ is a set of edges. Each edge $(u_i, u_j) \in E$ has a non-negative weight, denoted by c_{ij}.

Induced Subgraph. Given a subset S of V, we denote the induced subgraph by $G[S] = (S, E[S])$, where $E[S] = \{(u, v) | (u, v) \in E \wedge u, v \in S\}$. The number of vertices in S is represented as $|S|$.

Density Metrics g**.** To measure the density of a subset S of vertices in the graph, we adopt the class of metrics g commonly used in previous studies [2, 5, 6], $g(S) = \frac{f(S)}{|S|}$, where f is the total weight of $G[S]$, *i.e.,* the sum of the weight of S and $E[S]$:

$$f(S) = \sum_{u_i \in S} a_i + \sum_{u_i, u_j \in S \wedge (u_i, u_j) \in E} c_{ij} \tag{1}$$

The weight of a vertex u_i represents the suspiciousness of the user u_i, denoted by a_i ($a_i \geq 0$). The weight of the edge (u_i, u_j) represents the suspiciousness of the transaction (u_i, u_j), denoted by $c_{ij} > 0$. Intuitively, $g(S)$ is the density of the induced subgraph $G[S]$. The larger $g(S)$ is, the denser $G[S]$ is.

Graph Updates ΔG**.** We denote the set of updates to the graph G as $\Delta G = (\Delta V, \Delta E)$. The updated graph after incorporating ΔG is represented as $G \oplus \Delta G$, which is equal to $(V \cup \Delta V, E \cup \Delta E)$. In our framework, we only consider edge insertions and not deletions, as transaction graphs continuously evolve.

Spade targets to identify the subset of V, denoted as S^*, whose density metric $g(S^*)$ is maximized on the updated graph $G \oplus \Delta G$.

2.2 Backend of Spade

Data Storage. The storage component within Spade leverages the RisingWave [9] distributed database for stream processing. This choice simplifies the development and reduces the costs of creating real-time applications. As new transactions are produced during execution, Spade not only inserts these transactions into RisingWave but also initiates an incremental maintenance procedure. Specifically, Spade is designed to update the graph G incrementally, accommodating new transactions as they are inserted. This process is represented by ΔG, signifying the changes applied to the graph structure.

Spade **Code Generation Tool Chain.** The peeling algorithms in our framework are incrementalized based on user-defined functions, such as the vertex suspiciousness function and the edge suspiciousness function. To generate the codes of the new peeling algorithms, we provide simple programming interfaces by allowing developers to define customized data structures using C/C++. Our framework automatically plugs in the user-defined functions to the peeling score function, which is invisible to the developers and saves their programming efforts. We also provide a module that allows developers to control the streaming process, including the batch size and the checking of benign transactions.

APIs and Data Structure. To enhance flexibility and cater to various application needs, we offer developers comprehensive APIs that allow for the customization and deployment of their peeling algorithms. Specifically, developers have the ability to tailor VSusp and ESusp, enabling the development of nuanced fraud detection semantics unique to their requirements. For the seamless integration of new data, we have introduced two APIs tailored for edge insertion: InsertEdge and InsertBatchEdges. Detailed documentation of the APIs is thoroughly presented in our research paper [7, 8].

2.3 Sample Semantics

We demonstrate that popular peeling algorithms, such as DG [2], DW [5] and FD [6], can be easily implemented and supported by Spade. As an example, we present the discussion of FD and refer to [8] for the other instances. FD was proposed by Hooi et al.[6] to counter the disguise of fraudsters by weighting edges and setting the prior suspiciousness of each vertex using side information.

To implement FD using Spade, developers simply need to plug in their defined suspiciousness functions for vertices and edges. This can be done by calling the VSusp and ESusp APIs, respectively. The suspiciousness function for vertices, vsusp, is a constant function, where for a given vertex u, vsusp$(u) = a_i$. The suspiciousness function for edges, esusp, is a logarithmic function, such that for a given edge (u_i, u_j), esusp$(u_i, u_j) = \frac{1}{\log(x+c)}$, where x is the degree of the object vertex between u_i and u_j, and c is a positive constant [6].

Spade simplifies the implementation of customized peeling algorithms, significantly reducing the engineering effort required. As an example, the implementation of FD using Spade requires only around 20 lines of code (Listing 1), compared to the approximately 100 lines required in the original implementation of FD [6].

```
double vsusp(Vertex v, Graph g){
  return g.weight[v];
}
double esusp(Edge e, Graph g){
  return 1/log(g.deg[e.src]+5);
}
int main() {
  Spade spade;
  spade.VSusp(vsusp); //plug in vsusp (line 1-3)
  spade.ESusp(esusp); //plug in esusp (line 4-6)
  spade.TurnOnEdgeGrouping(); //enable edge grouping
  spade.LoadGraph("graph_sample_path");
  vector<Vertex> fraudsters = spade.Detect();
  //edge insertions prepared by developers
  vector<Edge> edge_insertions;
  for(Edge e: edge_insertions){
      fraudsters = spade.InsertEdge(e);
  }
  return 0;
}
```

Listing 1: Implementation of FD on Spade

3 DEMONSTRATION PLAN

The demonstration[1] consists of two parts. First, we showcase the ease-of-use and performance of Spade. Second, we provide real fraud cases from Grab and cryptocurrencies, highlighting its effectiveness.

Experimental Setup. Our Spade prototype [8] is implemented in C++ and demonstrated on a MacBook Air with an Apple M2 processor and 16GB of memory. The frontend is built using CoreUI [3] and the graph is visualized using G6 [1]. We compare the performance of Spade with three popular peeling algorithms (DG [2], DW [5],

[1] https://www.youtube.com/watch?v=VeiZ2vuwduY

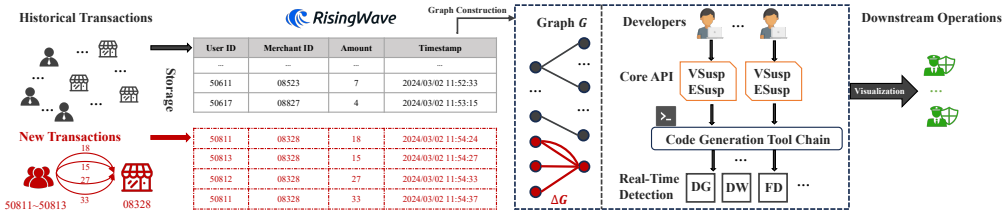
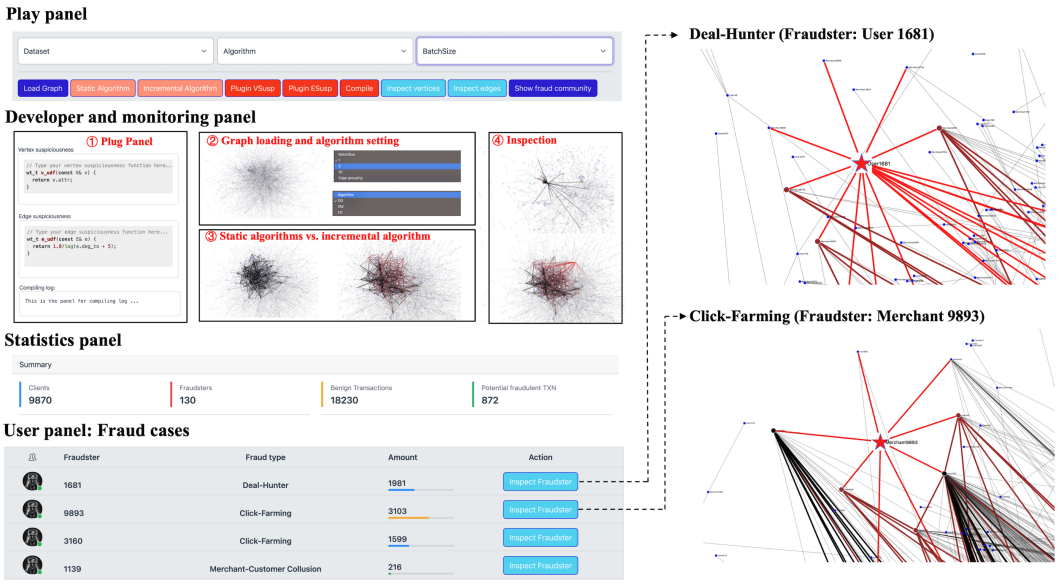


Figure 2: Backend of Spade and the real-time detection workflow



(a) Main panels

(b) Inspect two fraud cases

Figure 3: Spade's Graphical User Interface

Table 1: Statistics of Real-World Datasets

Datasets	$\|V\|$	$\|E\|$	avg. degree	Increments	Type
GFG	3.38M	29M	16.94	2.8M	Transaction
Grab1	3.991M	10M	5.011	1M	Transaction
Grab2	4.805M	15M	6.243	1.5M	Transaction
Grab3	5.433M	20M	7.366	2M	Transaction
Grab4	6.023M	25M	8.302	2.5M	Transaction
Amazon	28K	28K	2	2.8K	Review
Epinion	264K	841K	6.37	84.1K	Who-trust-whom
Slashdot0811	77K	905K	23.41	90.5K	Social network
Slashdot0902	82K	948K	23.09	94.8K	Social network
Youtube	1.13M	2.99M	5.27	299K	Social network
DBLP	317K	1.05M	6.62	105K	Collaboration network
NFT	18K	34K	3.71	3.4K	Transaction

and FD [6]), which identify the fraudulent community from scratch with each edge insertion. The improved versions of these algorithms, IncDG, IncDW, and IncFD, are implemented within Spade.

Datasets. We evaluate our framework on a diverse set of eleven datasets (see Table 1). Five of the datasets (GFG, Grab1-Grab4) are provided by our industry partner Grab and represent real-world transaction records as edges. The edges are ordered based on their increasing timestamps to reflect the temporal nature of the transactions. We also use three popular open datasets, including Amazon, Epinion and some datasets from SNAP [10]. Finally, we include a new dataset, NFT, which we crawled from the online marketplace [11]. The datasets vary in size, sparsity, and degree distribution, providing a comprehensive evaluation of our framework's performance.

A Walk through Spade**.** We demonstrate the ease-of-use and performance of Spade through the following five steps. Each step could be used by users to perform more in-depth studies on the data in a interactive manner to our system.

(1) Plug-in (developers). As shown in Figure 3 (Step ①), developers can specify the vertex suspiciousness and edge suspiciousness functions in the plug-in panel. Spade will integrate these functions into the engine and compile a new peeling algorithm for fraud detection.

(2) Play (moderators). In the play panel (at the top of Figure 3), moderators can select a dataset, a built-in peeling algorithm or a custom algorithm, and a batch size to adjust the efficiency and prevention rate. We also provide a special batch size option called "edge grouping" that enables real-time checking of whether a new transaction is benign or potentially fraudulent in $O(1)$. Spade provides three built-in fraud detection algorithms: DG, DW, and FD. Moderators can compare the performance of these algorithms with different suspiciousness functions. Spade enables moderators to manually submit graph updates and interact dynamically with the system.

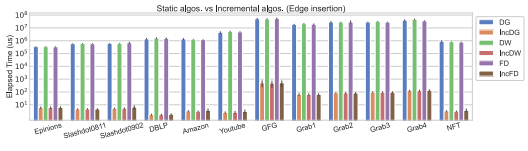


Figure 4: Efficiency Comparison between Peeling Algorithms and their Corresponding Incremental Versions on Spade

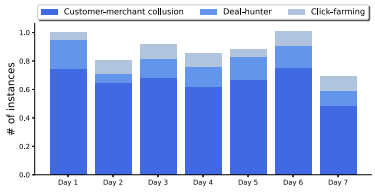


Figure 5: Identifying Fraud with Spade**.**

(3) Statistics. The detection results are displayed in the statistics panel, including the number of clients, the number of fraudsters, benign transactions, and potential fraudulent transactions. The details of the frauds are listed in the user panel, including the user ID, fraud type, and amount of fraudulent activities. The statistics are updated in real-time as new fraudsters are detected, allowing moderators to monitor the transaction network in real-time.

(4) Inspection (moderators). First, Spade loads the graphs from RisingWave and visualizes the transaction graph in the inspection panel (Figure 3, Step ②). Moderators can explore the graph structure by zooming and dragging. After setting the parameters, moderators can run either incremental or static algorithms to detect fraudsters and fraudulent activities. When Spade detects new fraudsters, it sends them to the front-end for visualization. During runtime, the fraudulent transactions involving these fraudsters will be highlighted in the transaction network. Spade visualizes the fraud communities on the graphs in real-time (Figure 3, Step ③). Spade provides two modes for moderators to inspect the fraudsters and fraudulent transactions (Figure 3, Step ④). Moderators can inspect the fraud manually by inputting the fraudster IDs or transaction IDs, and they can also investigate the cases by exploring the suspects' neighbors. Spade also provides a one-click inspection function for detected fraudsters to investigate their connections in the network. In addition to vertex-level and edge-level inspection, moderators can also click "Show Fraud Community" button to inspect the fraud community at the subgraph level. This can provide more insight into the fraud patterns and help moderators gain a better understanding of the fraud activity.

(5) Performance Comparison. The comparison of performance between incremental peeling algorithms and traditional peeling algorithms is conducted to evaluate the efficiency of Spade (Figure 4). Our experiments demonstrate that IncDG (respectively, IncDW and IncFD) is up to 4.17×10^3 (respectively, 1.63×10^3 and 1.96×10^6) times faster than DG (respectively, DW and FD) with an edge insertion. The results also show that IncDG (respectively, IncDG and IncFD) can prevent 88.34% (respectively, 86.53% and 92.47%) of fraudulent activities. The scalability of Spade is also verified in [8].

Case Studies (Figure 5). To showcase the effectiveness of Spade, we present case studies in the datasets of Grab and NFT [4]. These case studies highlight three common fraud patterns: *Deal-Hunting*, *Click-Farming*, and *Customer-Merchant Collusion*. The first pattern, *Deal-Hunting*, involves a group of users taking advantage of promotions or merchant bugs. The second pattern, *Click-Farming*, involves merchants recruiting fraudsters to create false prosperity by performing fake transactions. The third pattern, *Customer-Merchant Collusion*, involves customers and merchants performing fake transactions to take advantage of promotions and earn bonuses. All three fraud patterns form dense subgraphs in a short period of time. Spade can detect the fraudsters in real-time as it incrementally updates the fraud community without having to recompute from scratch. We show the "Inspect Fraudster" inspection by using two fraud cases.

Deal-Hunter. When the moderators click on "Inspect Fraudster" for client 1,681, Spade zooms in to show the connections around the fraudster in the network. We can see from the panel that the fraudster frequently makes purchases from merchants with IDs 6,476, 5,867, and 849, as well as from some normal or fraudulent merchants. The moderators can observe that these transaction activities form a deal-hunter fraud pattern, providing them with useful insights for detecting and preventing similar fraudulent activities in the future.

Click-Farming. In the same way, by clicking on "Inspect Fraudster" for merchant 9,893, the moderators can observe that apart from normal users, several fraudsters, including 8,863, 3,996, and 7,293, make frequent purchases from this merchant. These fraudulent transactions form a dense subgraph and indicate a click-framing fraud pattern.

ACKNOWLEDGMENTS

Bingsheng He is the corresponding author. This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002) and the Ministry of Education AcRF Tier 2 grant, Singapore (No. MOE-000242-00/MOE-000242-01).

REFERENCES

[1] AntV. 2021. G6 - A Graph Visualization Framework in JavaScript.
[2] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.
[3] CoreUI. 2023. Open Source Bootstrap Admin Template. https://coreui.io/.
[4] Dune. 2023. DUNE. *https://dune.com/cryptok/NFT-Wash-Trading*.
[5] Naga VC Gudapati, Enrico Malaguti, and Michele Monaci. 2021. In search of dense subgraphs: How good is greedy peeling? *Networks* 77, 4 (2021), 572–586.
[6] 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*. 895–904.
[7] Jiaxin Jiang, Yuhang Chen, Bingsheng He, Min Chen, and Jia Chen. 2024. Spade+: A Generic Real-Time Fraud Detection Framework on Dynamic Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2024).
[8] Jiaxin Jiang, Yuan Li, Bingsheng He, Bryan Hooi, Jia Chen, and Johan Kok Zhi Kang. 2022. Spade: A Real-Time Fraud Detection Framework on Evolving Graphs. *Proceedings of the VLDB Endowment* 16, 3 (2022), 461–469.
[9] RisingWave Labs. 2021. RisingWave. https://github.com/risingwavelabs/risingwave. Accessed: March 22, 2023.
[10] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[11] Zhen Zhang, Bingqiao Luo, Shengliang Lu, and Bingsheng He. 2024. Live graph lab: towards open, dynamic and real transaction graphs with NFT. *Advances in Neural Information Processing Systems* 36 (2024).