Applied Intelligence (2024) 54:7461–7478
https://doi.org/10.1007/s10489-024-05545-4

SMoTeF: Smurf money laundering detection using temporal order and flow analysis

Shiva Shadrooh[1] · **Kjetil Nørvåg**[1]

Accepted: 19 May 2024 / Published online: 7 June 2024

Abstract
Smurfing in financial networks is a popular fraud technique in which fraudsters inject their illegal money into the legitimate financial system. This activity is performed within a short period of time, with recurring transactions and multiple intermediaries. A major problem of existing graph-based methods for detecting smurfing is that they fall short of retrieving accurate fraud patterns. Consequently, the result is numerous non-fraudulent patterns alongside a few fraud patterns, causing a high false-positive rate. To alleviate this problem, we propose SMoTeF, a framework that extends existing graph-based smurf detection methods by distinguishing fraudulent smurfing patterns from non-fraudulent ones, thus significantly reducing the false-positive ratio. The core of the approach is a novel algorithm based on computing maximum temporal flow within temporal order of events. In order to evaluate the approach, a framework for injecting various smurfing patterns is developed, and experimental results on three real-world datasets from different domains show that SMoTeF significantly improves on the effectiveness of the state-of-the-art baseline, with only marginal runtime overhead.

Keywords Anomaly detection · Smurfing · Temporal networks · Money laundering · Maximum temporal flow

1 Introduction

The task of anomaly detection in time-evolving graph networks has recently drawn increasing attention, due to the broad applicability. In time-evolving graph, anomalies can range from noticeable outliers that deviate significantly from normal behavior, to complex patterns requiring reexamination and new measures. Therefore, conventional rule-based anomaly detection techniques have become ineffective in detecting complex patterns.

Smurfing is among the most complex pattern anomalies, enabling fraudsters to smoothly camouflage malicious actions as normal behavior. This pattern frequently appears in financial [1–3], cryptocurrency [4, 5], and internet traffic network [6–9] domains. In financial networks, smurfing money laundering is the process of transferring massive amounts of illegal money from one fraudulent account to another by dispersing small cash quantities through intermediary accounts known as smurfs. An example of malicious smurfing money laundering activity from January to February of the same year is depicted in Fig. 1, representing three different sets of transfers described as follows: Figure 1 represents three different sets of transfers described as follows:

- The red solid arrows from bandit X to Y present transfers that are possibly money laundering, sending a relatively large amount of money (more than regular daily transfers) through a smurf account within a short period.
- The green dashed arrows from X to Y indicate clean transfers through the smurf accounts, which pass a relatively small amount of money from X to Y.
- The yellow shaded arrows represent shady transfers from X to Y, where a considerable amount of money in January was sent from a smurf account to Y, then in February, a large amount of cash was transferred from account X to the same smurf. However, considering also the temporal order of the transfers, X is not the source of the money that was transferred to Y.

Current state-of-the-art methods for discovering smurfing patterns, e.g., [1, 2, 5], suffer from high false-positives

✉ Shiva Shadrooh
shiva.shadrooh@ntnu.no

Kjetil Nørvåg
noervaag@ntnu.no

[1] Department of Computer Science, NTNU, Trondheim, Norway

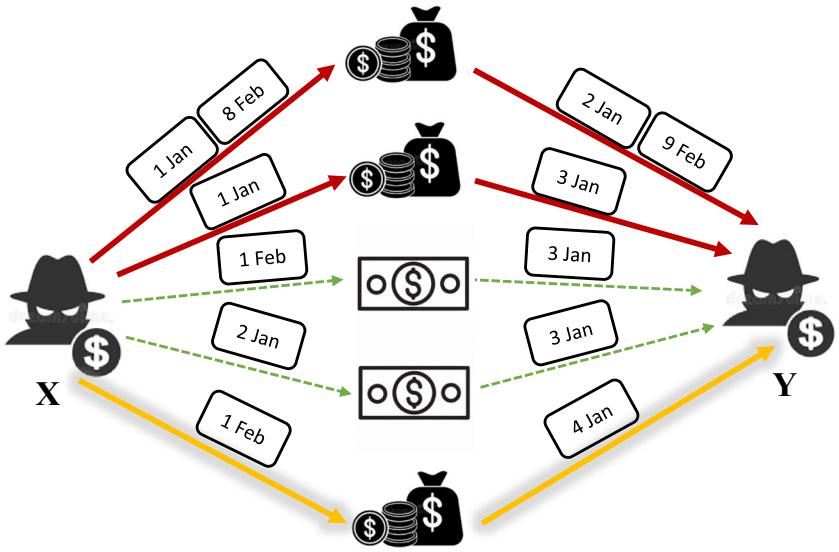


Fig. 1 Example of smurfing money laundering from January to February

rates due to only considering the graph structure, ignoring the importance of the money quantity and the temporal order of the transactions in the pattern. As a result, these methods retrieves a significant number of non-fraudulent patterns (between 75% and 99% of the total patterns found [2]), leading to a high cost for the subsequent manual investigation of the found patterns.

In this paper, we introduce SMoTeF, which aims to utilize maximum money flow computation within the temporal order of transfers to significantly reduce the non-fraudulent smurfing graph patterns extracted from big time-evolving financial networks. This work, motivated by the high false-positives rates in the retrieved smurfing patterns from the recent state-of-the-art methods [1, 2], alongside the study of the maximum temporal flow problem (MTF) by Akrida et al. [10], extends existing methods by efficient pruning of non-fraudulent patterns. To the best of our knowledge, none of the previous work considers the importance of the temporal order constraint of transfers and maximum temporal flow computation to improve the accuracy of the results. Experiments on real-world data show that the proposed methods in this paper can detect various types of smurfing patterns not only for money laundering but also in other real-world datasets that share similar characteristics, such as internet traffic networks.

In summary, we make the following contributions:

- **Temporal order constraint:** We propose novel techniques used in the SMoTeF framework to detect fraudulent money laundering smurfing patterns in a set of retrieved smurfing patterns, by utilizing the maximum temporal flow of money alongside the temporal order of edges to eliminate false positive cases. SMoTeF can also report the most suspicious periods for each investigated fraud pattern which provides auditors with more information.
- **Maximum temporal flow algorithm:** In order to compute the maximum temporal flow for each pattern, we divide the maximum temporal flow problem into two sub-problems to decrease the algorithm's time complexity by foremost pruning considering events' temporal sequence. To the best of our knowledge, our approach is the first solving the smurf-detection problems based on these techniques.
- **Injected smurfing patterns:** We present a framework for evaluating smurfing pattern detection, based on injecting a number of complex smurfing structures considering temporal order and money quantity.
- **Effectiveness:** We perform an extensive experimental evaluation, using three real-world datasets from different domains. The results shows that SMoTeF significantly improves on the effectiveness of the state-of-the-art baseline, with only marginal runtime overhead.

Organization. The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents preliminaries and the problem definition. Section 4 describes in detail the proposed approach. In Section 5, we present the results of the experimental evaluation. Finally, in Section 6 we conclude and discuss future research directions.

2 Related work

Graph-based anti-money laundering can be be classified into approaches based on algorithmic data mining, and techniques based on machine learning [11–20]. A major drawback of using machine learning models to detect money laundering

activities, is the these approaches are based on supervised learning. This implies hat they require a large number of both positive and negative labels, which in general can be costly, and even impossible, to acquire. Also, money laundering detection can be regarded as adversarial attacks in which deep neural networks may not be robust [21]. We will in this paper focus on approaches using algorithmic data mining, for an overview of machine learning techniques and detecting financial fraud we refer to [11, 13–15].

There are many techniques for detecting pattern anomalies in both streaming and static graph processing [13, 22, 23], however only a few are appropriate for detecting money laundering. In the following, we will review the most recent approaches to graph-based pattern anomaly detection in financial networks, which are most related to our work.

2.1 Static graph processing

AutoAudit [1] uses an unsupervised method for detecting smurfing patterns in a time-evolving graph of financial transactions, by reordering the graph adjacency matrix. However, the results might contain many false-positives, as the approach cannot efficiently prune the retrieved patterns. CUBEFLOW [24] and FlowScope [21] focus on detecting dense subgraphs, and the aim is to detect dense money transfers from a source to untraceable destinations. These transfers use many middle accounts to avoid getting noticed. Although both methods contribute to anti-money laundering tasks, money laundering schemes do not necessarily generate dense subgraphs.

Starnini et al. [2] introduce a smurf-based anti-money laundering technique based on the intuition that the velocity characteristics of smurfing will allow smurfs to be found by using a standard database join. However, the preprocessing steps they use may reduce the problem size for applying a database join; it may also cause the unwanted removal of multiple possible fraud patterns.

Given a cryptocurrency transaction network as application context, the AntiBenford subgraph framework [5] propose an unsupervised method for finding subgraphs such as smurf-like structures that perform many transactions among its node that deviate significantly from Benford's law. In the presence of a mature network, this framework can detect discriminative patterns, though cryptocurrency networks are often too small in emerging stages to develop discriminative subgraphs. Also, fraud subgraphs are not always highly discriminative from normal behavior.

2.2 Stream graph processing

Rather than exploring fixed patterns, MonLAD [25] examines how normal nodes and their neighborhoods behave based on their behavioral score deviation from the generalized Pareto distributions of certain features. MonLAD utilizes this score to spot money laundering agents.

Given a stream of edges, F-FADE [26] employs frequency-based matrix factorization and computes the anomaly score based on the likelihood of the observed frequency of each incoming interaction. The issue with an edge-level anomaly is that it cannot recognize node-level expansion when no direct edge event is associated with that node.

Spotlight [23] is a randomized sketching-based approach to detect anomalous subgraphs containing the sudden appearance or disappearance of large dense subgraphs that act far from normal. To overcome the problem of detecting dense subgraphs from scratch with graph updates, Spade [27] proposed an incremental real-time fraud detection framework based on three incremental peeling sequence reordering techniques to avoid the detection of fraudulent communities from scratch. Since in both Spade and Spotlight, the density of subgraphs is taken into account to determine a pattern of malicious edges; they fell behind in many money laundering cases in which dense subgraphs are not formed.

3 Problem statement

In this section, we present the basics regarding temporal flow networks, based on Akrida et al. [10], and then we proceed to define our problem statement.

3.1 Preliminaries

Definition 1 *(Temporal multidigraph).* Given a temporal multidigraph $G = (V, E)$, let V be the set of nodes and E be the set of edges. Consider each edge is represented as $e = (u, v, t, w)$, where u is the source node, v is the target node, t is the timestamp, and w is the weight. This graph structure allows for multiple edges between the same nodes, each differentiated by their temporal characteristics.

Definition 2 *(Journey).* A journey j from a vertex u to a vertex v, denoted as u $\rightarrow$v journey, is a sequence of timestamped edges $(u, u_1, t_1), (u_1, u_2, t_2), ..., (u_{k-1}, v, t_k)$, such that $t_i < t_{i+1}$, for each 1 $\leq$ i $\leq$ k - 1.

Definition 3 *(Flow network).* A flow network is a directed graph (digraph) $G = (V, E)$, where V represents a set of vertices and E represents a set of directed edges connecting these vertices. The network is characterized by a source vertex, from which the flow originates, and a sink (destination) vertex, where the flow is intended to end. Unlike traditional flow networks, in this specific application area, there are no predefined capacities on the edges, allowing for a more flexible representation of flow.

Definition 4 *(Flow).* In the context of a flow network, flow refers to a real-valued function $f : E \rightarrow \mathbb{R}$, which is assigned to the edges of the network. The flow f represents the quantity or magnitude of an entity (like data, money, or traffic) being transferred through the network. The key constraints in this definition is:

Non-Negativity Constraint: For every edge (u, v) in E, the flow $f(u, v)$ must be non-negative, i.e., $f(u, v) \geq 0$.

Definition 5 *(Smurfing flow network).* A smurfing flow network (G, s, d, J) is a temporal multidigraph $G = (V, E)$ with the following properties:

1. There are three special vertices in this network: the source vertex (v_s), the middle (smurf) set V_m consist of (v_m) vertices and the sink vertex (v_d).
2. For a smurfing flow network (G, s, d, J) such as one shown in Fig. 2, we consider J as a set of journeys j from $v_s \rightarrow v_{mi} \rightarrow v_d$. The number of journeys are essentially equal to number of v_m vertices (based on the application in mind, we make the assumption that smurf accounts avoid contact each other to bypass suspects).
   For a journey $j \in J$, we have two sets, fan-in set l^+ and fan-out set l^-. Fan-in set l^+ consists of ordered edge occurrences from $v_s \rightarrow v_m$ and fan-out set l^- consists of ordered edge occurrences from $v_m \rightarrow v_d$. If l^+ represents incoming edges to an intermediary v_m and l^- as outgoing edges from the same v_m, then:

$$l^+(v_m) = \{e \in E \mid v_s, v_m \in V, e = (v_s, v_m, t, w)\}$$
$$l^-(v_m) = \{e \in E \mid v_d, v_m \in V, e = (v_m, v_d, t, w)\} \quad (1)$$

3. A capacity function which is a mapping of $c : E \rightarrow \mathbb{R}^+$ represented by $c(u, v)$ for $(u, v) \in E$. It denotes the maximum flow amount that can pass through an edge.

Definition 6 *((traditional) Cut).* A cut in traditional graph theory is an imaginary line that divides a network into two groups of nodes, where the nodes in each group are no longer directly connected to the nodes in the other group by the edges that cross this imaginary line. In a non-acyclic (traditional) flow network from vertex s to t, in which the timestamp is ignored, an s-t cut is a set of edges such that removing them causes no directed paths from s to t in G. An s-t cut is to split the network's vertices into two components S and T, with the source in one component and the sink in the other. Then cut-set X_C is the set of edges connecting the cut's source part to the sink part:

$$X_C := \{(u, v) \in E : u \in S, v \in T\} \quad (2)$$

Definition 7 *(Cut capacity).* Cut capacity in a graph is the total capacity of the edges that you would need to remove or

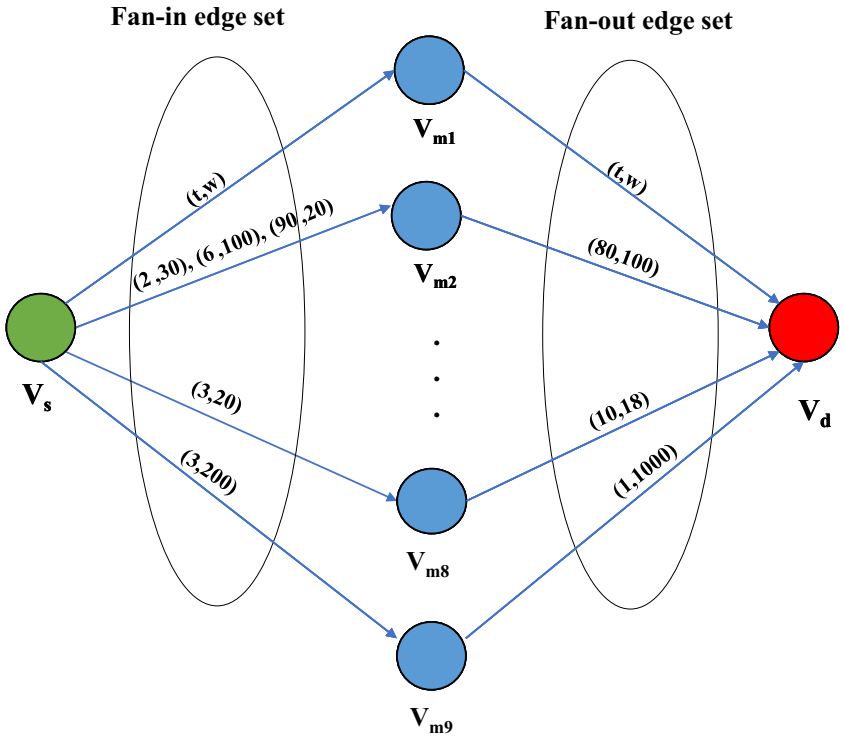


Fig. 2 Detailed representation of smurfing pattern

block to split the graph into two parts - separating the source S from the destination T. Cut capacity gives a measure of how much flow can be interrupted in the network by blocking these edges. The capacity of an s-t cut is the sum of the capacities of the edges in its cut-set. Considering source S and sink T, the capacity of the cut is computed as follows:

$$c(S,T) = \sum\nolimits_{(u,v)\in X_C} c(u,v) \tag{3}$$

Definition 8 *(Temporal cut).* Let (G, s, d, J) be a smurfing temporal flow network on a digraph G. For each $j \in J$ from $\mathrm{s} \rightarrow \mathrm{d}$, a temporal cut C_{sd} is a set of timestamped edges such that removal from C_{sd} results in no path from $\mathrm{s} \rightarrow \mathrm{d}$ (s and d are not connected directly).

Definition 9 *(Minimum temporal cut).* Minimum cut problem in classical graph theory aims to find the smallest set of edges that can be removed to disconnect the source S from the destination (sink) T. In the temporal version, the objective is to find a cut in the network considering the time dimension. This might involve identifying a set of edges whose removal would disconnect the source from the sink over the longest duration. In formal language, CT_{sd} for a journey from $\mathrm{s} \rightarrow$ d is a set of timestamped edges called minimal temporal cut if:

- CT_{sd} is a temporal cut.
- If we remove any subset of CT_{sd}, then leaves no flow from s to d during the $\mathrm{s} \rightarrow \mathrm{d}$ lifetime.

Minimum temporal cut capacity of an s-t cut is computed as follows:

$$Mtc(s,t) = \min_{X_C} \sum_{(u,v)\in X_C} c(u,v). \tag{4}$$

Definition 10 *(Maximum temporal flow).* For each journey j from $s \rightarrow d$ in graph G, maximum temporal flow $Mtf(s,d)$ is the maximum outgoing flow from the node v_s that transfer as incoming flow to the destination v_d. According to the maximum-flow minimum-cut theorem, the maximum temporal flow for a journey is equal to the capacity of minimum temporal cut in it, such that:

$$Mtf(s,d) = Mtc(s,d) \tag{5}$$

Note. In the smurfing flow network (G, s, d, J) (special case of flow network), the maximum temporal flow equals to the sum of maximum temporal flow of all journeys in J.

$$Mtf(G) = \sum_{s\rightarrow d\in J} Mtf(s,d) \tag{6}$$

Example. Figure 2 provides a detailed representation of a smurfing pattern, representing nine journeys (j) from $\mathrm{v}_s \rightarrow \mathrm{v}_{mi} \rightarrow \mathrm{v}_s$ $(1 \leq \mathrm{i} \leq 9)$. The smurfing flow network is a temporal multidigraph, allowing multiple edge occurrences for each edge (as the example shown in the edge $\mathrm{v}_s \rightarrow \mathrm{v}_{m2}$ of Fig. 2). Edge occurrences are then defined with attributes (t, w), where w represents the amount of money transferred between two nodes while t represents the timestamp associated with a transaction.

Consider the example of a minimum temporal cut in the journey through intermediary v_{m2} (Fig. 2). Based on the minimum temporal cut definition, there are two temporal minimal cuts, $CT_{\mathrm{v_s v_d}}$ and $CT'_{\mathrm{v_s v_d}}$, defined as follows:

$$\begin{aligned} CT_{\mathrm{v_s v_d}} &= \{((\mathrm{v_s}, \mathrm{v_{m2}}), 2, 30), ((\mathrm{v_s}, \mathrm{v_{m2}}), 6, 100)\} \\ CT'_{\mathrm{v_s v_d}} &= \{((\mathrm{v_{m2}}, \mathrm{v_d}), 80, 100)\}. \end{aligned} \tag{7}$$

In this example, the smallest set of edges that can be removed to disconnect the source from the sink with respect to the journey for $CT_{\mathrm{v_s v_d}}$ are edges $(V_s, V_{m2}, 2, 30)$ and $(V_{m2}, V_d, 6, 100)$. Both occurrences must be considered; removing only one would still allow for a connection from v_s to v_d to be established. This is the smallest possible set of edges to be removed, as the edge occurrence (90, 20) has no effect on the flow within the network for the given temporal scope, as it is not utilized subsequently. In this journey, The capacity of the minimum temporal cuts denoted by $c(CT_{\mathrm{v_s v_d}})$ and $c(CT'_{\mathrm{v_s v_d}})$ where $c(CT_{\mathrm{v_s v_d}})$ equals to the summation of money transferred in timestamps 2 and 6, while $c(CT'_{\mathrm{v_s v_d}})$ equals to the money transferred in timestamp 80.

The maximum temporal flow from $\mathrm{v}_s \rightarrow \mathrm{v}_d$ through v_{m2} is then determined by the minimum capacity of the two temporal cuts (100 and 130), resulting in a maximum flow of 100, reflecting the maximum money that can flow in this journey through intermediary v_{m2}, based on the temporal order of transactions.

Definition 11 *(Flow threshold).* Let T_f represent the minimum threshold for a money flow from node u to node v in a smurfing flow network. If the amount of money transferred from the source to the sink through the all journeys exceeds T_f, the transaction is considered for a fraud. Consequently, nodes u and v as well as intermediaries involved become candidates for further review and investigation.

Definition 12 *(Smurf threshold).* A minimum number of intermediaries (smurf) nodes T_s, participating in a fraudulent smurfing network. A smurfing flow network involving larger intermediaries than T_s can be placed under suspicious patterns.

3.2 Definition of fraudulent smurfing network

Given a set of smurfing flow networks (G, s, d, J), smurf threshold T_s and the flow threshold T_f, estimate if each of (G, s, d, J) can be considered as a fraudulent smurfing network and prune the non-fraudulent smurfing networks. Finally, if a fraudulent network is detected, propose its most suspicious activity period.

A smurfing money laundering fraud network satisfies the following characteristics:

- **Money flow density**: Maximum temporal flow in a smurfing flow network measures the total volume of transferred funds within a specified period, reflecting the density of money flow. A high maximum temporal flow in the specified period indicates a dense money flow, suggesting frequent and extensive transactions indicative of an active financial exchange. Conversely, a low maximum temporal flow in a specified period reflects a non-dense money flow, potentially a non-fraudulent pattern. Money launderers attempt to transfer massive funds through smurf accounts, underscoring the significance of money flow density computation in fraud detection. In this paper, we introduce the concept of the flow threshold (T_f), established by financial institutions as the maximum allowable money transferred between two entities, whether direct or indirect. When the maximum temporal flow of the network surpasses this threshold, it indicates potential money laundering activity.
- **Temporal order constraint**: Understanding the chronological sequence of transactions within a financial network is crucial in analyzing financial patterns. This sequence, called temporal order, plays a vital role in distinguishing between random and actual fraudulent patterns. It becomes particularly important in maximum temporal flow problem where time aspect is involved. In standard maximum flow problems, the maximum flow can be computed efficiently by considering the weight of each edge. However, in maximum temporal flow problem, it is essential to ensure that the destination node is temporally accessible from the source node in each journey (results in non-empty cut-set). If the destination node is not temporally accessible in a specific journey, the flow of funds to destination node also become impossible, resulting in a the non-fraudulent journey. By considering the temporal order constraint for each journey, we can confirm that v_s is indeed the source of money transferred to v_d.
  If the cut-set of a journey appears empty, meaning there is no feasible temporal path from source to destination, we can prune the journey and check if number of the valid journeys exist in the smurfing network is bigger than or equal to T_s (the quantity of smurf accounts involved in the pattern meets or exceeds the threshold, indicating fraudulent activity.). Examples of valid and invalid temporal order can be seen in Fig. 2, where the journey through V_{m8} indicate a valid temporal order, while journey with V_{m9} in the middle, indicates an invalid temporal order, since there is no feasible temporal path from source to destination.

4 SMoTeF

The SMoTeF framework is a multi-stage process that facilitates a thorough analysis of smurfing activities in time-evolving graphs. shown in Fig. 3. In its initial stage, known as *smurf detection*, the framework takes a time-evolving graph G as input and applies a smurfing detection method to identify smurfing graph patterns based on graph characteristics. The output of the first stage consists of a set of smurfing patterns within the graph, along with the nodes involved in each identified pattern, yet in this stage retrieved patterns contains vast amount of false positives. Subsequently, the second stage, *temporal order constraint check,* takes its input from the output of the previous phase. Here, a smurfing flow network (G, s, d, J) built for each of patterns outputted from previous stage and then it undergoes temporal order checking to eliminate temporal invalid journeys, resulting in the generation of valid time-ordered smurfing flow networks (in this phase a lot of false patterns will be pruned). Finally, the framework moves to the *maximum money flow analysis* stage. Given a valid time-ordered smurfing flow network (G, s, d, J) derived from the previous stage output, final stage approach utilizes a maximum temporal flow algorithm to eliminate the remaining of false positive patterns. The output of this stage includes the most suspicious smurfing flow networks, along with the corresponding periods of their activity, which are the output of the SMoTeF framework. We will in the following describe each of the three stages in more detail, starting with the motivation and intuition behind our proposed method.

4.1 Motivation

As already noted, existing state-of-the-art financial smurf detection methods have high false positives rates [1, 2]. In order to reduce the false-positive rate, we reconsider the anomaly measurements for the retrieved patterns based on the maximum temporal flow analysis, which involves the temporal order constraint check and maximum money flow analysis.

To understand why conventional maximum flow algorithms are impractical in a temporal network, we provide

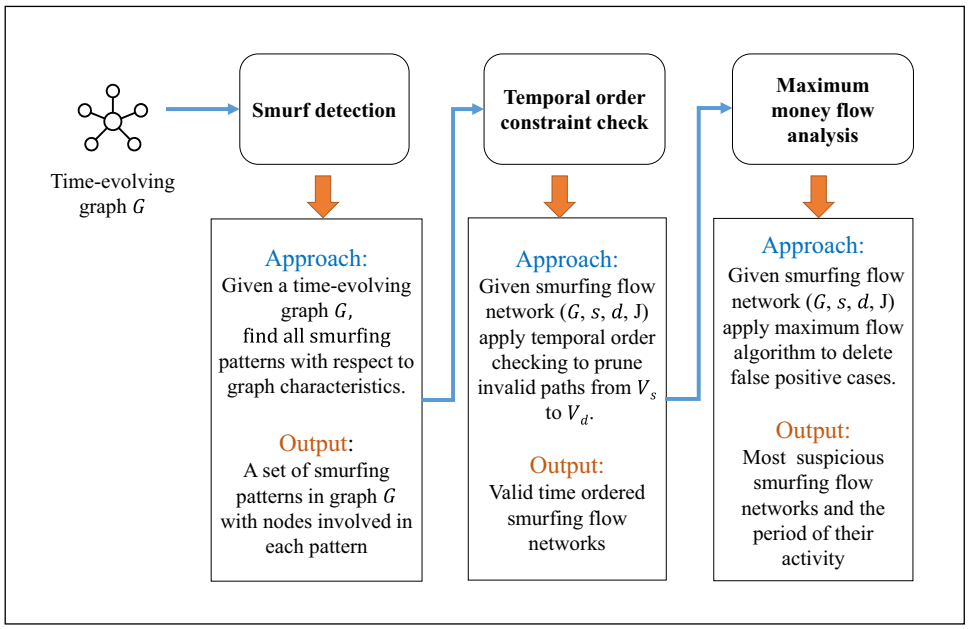


Fig. 3 Overview of the SMoTeF framework

two examples to compare Ford-Fulkerson algorithm [28] with SMoTeF in Fig. 4. Part (1) in both Fig. 4a and b are the computation of the maximum flow using Ford-Fulkerson algorithm (conventional method) without considering the time attribute.

Part (2) of Fig. 4a and b present how SMoTeF framework computes the maximum temporal flow, considering valid temporal order checking. First, we compute the minimum temporal cuts and based on the capacity of minimum temporal cut-set CT_{sd} and CT'_{sd}, and then calculate the maximum temporal flow passes from $v_s \rightarrow v_d$.

Figure 4a exhibits the effect of invalid temporal order pruning using SMoTeF-T algorithm. In this example, edge occurrence $(15, 200\$)$ should be pruned since no edge occurrence can transfer its flow afterward. Edge occurrence $(3, 20\$)$ can be considered the source of $(7, 30\$)$ but not for the $(10, 50\$)$ since edge $(7, 30\$)$ already transferred all the capacity came from $(3, 20\$)$. Finally, after pruning, the maximum temporal flow equals 20$.

4.2 Smurf detection

The initial smurf detection can be performed by adapting one of the existing algorithms, e.g., AutoAudit [1], employing it as described in the following.

Given a time-evolving multidigraph G, first, we apply the smurf detection algorithm to find all smurfing patterns with respect to the description of smurfing pattern fraud in the application. The algorithm will retrieve a set of smurfing patterns, each containing the nodes involved in the pattern called *smurf index list*, $SMI = [v_s, V_m, v_d]$. The *SMI* consists of the source, intermediaries, and destination nodes which

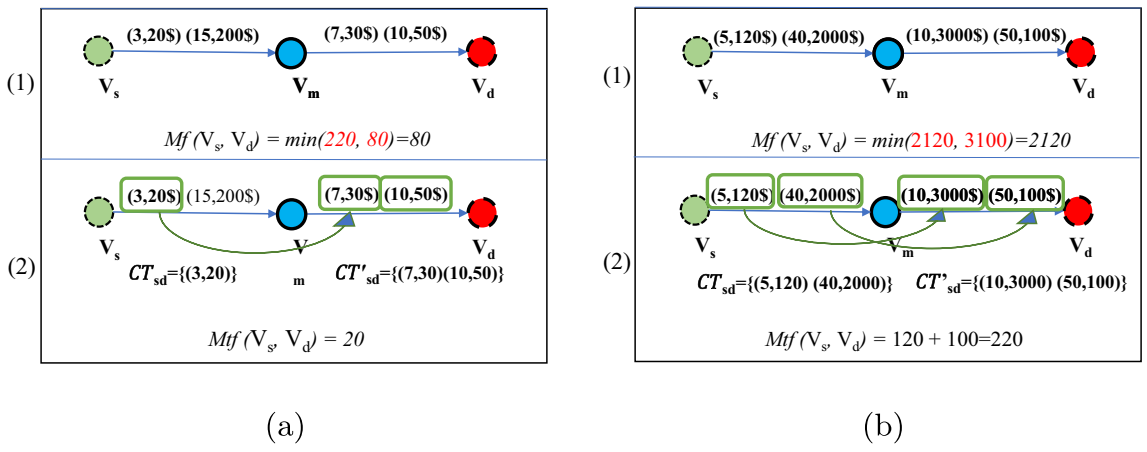


Fig. 4 Comparison of conventional maximum flow with the maximum temporal flow problem

Algorithm 1 GetJourney.

Data: $SMI = [v_s, V_m, v_d]$, Adjacency List $Adj(G)$
Result: Journey set J for smurfing flow network (G, s, d, J)
$l^+ \leftarrow \emptyset, l^- \leftarrow \emptyset, J \leftarrow \emptyset$;
for v_m **in** V_m **do**
  $j \leftarrow \emptyset$,
  $l^+ \leftarrow$ All edge occurrences of (v_s, v_m) in $Adj(G)$;
  $l^- \leftarrow$ All edge occurrences of (v_m, v_d) in $Adj(G)$;
  $j = (l^+, l^-)$;
  J.insert(j);
return J;

later will be used alongside the adjacency list of graph G ($Adj(G)$) to obtain the smurfing flow network (G, s, d, J) for each detected pattern. The $Adj(G)$ and the set of all *SMI*s correspondent to each pattern are the output from this section which are the input of SMoTeF framework.

4.3 Temporal order checking

This section will describe the temporal constraint checking algorithm SMoTeF-T, which aims to check the temporal order of edges in each journey $j \in J$ of retrieved smurfing patterns from the smurf detection phase. The initial step is to construct the smurfing flow network from the smurf index list *SMI*. It will be a costly search if we want to retrieve edge attributes from the edge list of a time-evolving graph G, so we use the adjacency list, $Adj(G)$, created while reading data. For each pattern, we recover all the journeys in J and their ordered edge occurrences which formed a smurfing flow network using the GetJourney function shown in Algorithm 1.

To compute the maximum temporal flow on the detected patterns from the initial smurf detection section, we need to be sure that the order of the l^+ and l^- for each journey is valid, which means that flow can pass through an available edge e, then through subsequent edges only at some time (t $\geq$ t+1). To retrieve the valid temporal order, we construct the minimum temporal cut-set for all journeys in J. Based on the maximum-flow minimum-cut theorem, we need to find the capacity of the minimum temporal cut-sets to precisely compute the maximum temporal flow.

Algorithm 2 presents the steps of the temporal order checking section. Given the smurfing flow network (G, s, d, J), we want to find the valid temporal order of each journey in J by finding the minimum temporal cuts.

Lines 2-17 compare the timestamp of each edge occurrence in the l^+ with edge timestamps in l^- edge sets, one by one. In this algorithm, for each journey j, the incoming edges of $v_s \rightarrow v_m$ in l^+ are shown by eIn, while the outgoing edge from the smurf $v_m \rightarrow v_d$ in l^- called as $eOut$.

The *Visited* variable with value of 0 indicates that we have seen the edge occurrence in l^- set for the first time which helps to avoid visiting the same edge occurrence twice. If

Algorithm 2 SMoTeF-T.

Data: Journey set J, Smurf threshold T_s
Result: Checked temporal ordered journey set OJ for smurfing flow network (G, s, d, J)
1 $OJ(G) \leftarrow \emptyset$, *Visited* $\leftarrow 0$;
2 **for** j **in** J **do**
3 $f(eOut) \leftarrow 0$;
4 **for** eIn **in** l^+ **do**
5 **for** $eOut$ **in** l^- **do**
6 $f(eOut) += w_{eOut}$;
7 **if** $t_{eIn} > t_{eOut}$ **then**
8 Continue;
9 **if** $t_{eIn} \leq t_{eOut} \wedge$ *Visited* $== 0$ **then**
10 $CT_{sd}.insert(eIn)$;
11 $CT'_{sd}.insert(eOut)$;
12 *Visited*++;
13 **if** $t_{eIn} \leq t_{eOut} \wedge$ *Visited* $\neq 0 \wedge w_{eIn} \geq f(eOut)$ **then**
14 $CT'_{sd}.insert(eOut)$;
15 Continue;
16 **if** $t_{eIn} \leq t_{eOut} \wedge$ *Visited* $\neq 0 \wedge w_{eIn} < f(eOut)$ **then**
17 break;
18 **if** $CT_{sd} \neq \emptyset \wedge CT'_{sd} \neq \emptyset$ **then**
19 $OJ.insert(j)$;
20 **else**
21 ignore j;
22 **if** $|OJ| \geq T_s$ **then**
23 return OJ;
24 **else**
25 return $\emptyset$;

t_{eIn} is smaller than t_{eOut}, and t_{eOut} first visited, we can add eIn to CT_{sd} and $eOut$ to CT'_{sd} and jump to compare t_{eIn} with the next timestamp in l^-. If t_{eIn} is bigger than t_{eOut}, there is no chance that eIn is the source for $eOut$, so ignore the $eOut$.

In line 6, $f(eOut)$ represents the accumulated temporal flow of visited occurrences of edge e in l^-. Line 13 states that if w_{eIn} (the edge weight of current eIn) be greater and equal to the temporal flow capacity $f(eOut)$, then ensures that the money flow consumed later in the l^- edge came earlier from the l^+ edge (in the actual smurfing fraud we expect the least interaction between intermediaries involved, in order to escape from suspicion of forming a dense subgraph).

For a journey from $v_s \rightarrow v_d$, CT_{sd} and CT'_{sd} are derived from the ordered timestamped edge list of l^+, l^- edges, which form the minimum temporal cut for l^+ and l^- respectively. We formalize them as follows:

$$CT_{sd} = \begin{cases} eIn, & \text{if } Visited = 0 \wedge\ t_{eIn} \leq t_{eOut}. \\ null, & \text{if } t_{eIn} >\ t_{eOut}. \end{cases} \tag{8}$$

$$CT'_{sd} = \begin{cases} eOut, & \text{if } Visited = 0 \wedge\ t_{eIn} \leq t_{eOut}. \\ eOut, & \text{if } Visited = 1 \wedge\ t_{eIn} \leq t_{eOut} \\ & \wedge\ f(eOut) \geq w_{eOut}. \\ \emptyset, & \text{otherwise.} \end{cases} \tag{9}$$

Lines 18-21 check if CT_{sd} and CT'_{sd} are empty. If one or both are empty, then journey j and intermediary v_m in j should be disregarded. Finally in lines 22-25, we should count the number of remained journeys in J or intermediaries to see if the number of intermediaries is still equal and more than smurf threshold T_s; if not, algorithm should return an empty set which shown the pattern should be discarded.

Now that we have pruned the retrieved patterns from invalid occurrence of edges, we need to devise a method for computation of maximum flow for each journey by traversing the journey throughout time attribute. Figure 4b indicates the necessity of next section considering the time aspect. Part (2) in this figure shows that even if the edge occurrences for each edge follow a valid order, accumulating weights of valid edge occurrences to compute maximum flow can result in erroneous maximum temporal flow. In this example, (5, 120$) is the source of (10, 3000$) and (40, 2000$) can be the source for (50, 100$); thus, this instance doesn't require pruning. Using the conventional maximum flow algorithm, the maximum flow value in Part (1) is computed as 2120. While in Part (2), at timestamp 10, v_m transferred 120$ from the transaction (5, 120$) and at timestamp 50, it only transferred 100$ of incoming (40, 2000$). Here the maximum flow transferred from v_s to v_d equals 220. There is a significant difference between the maximum flow computed in Part (1) and (2).

4.4 Money flow analysis

The primary motivation of criminals for smurfing is to transfer a significant amount of illegal cash. With this in mind, a smurfing flow network can be considered highly suspicious if it transfers a massive amount of money, greater than a minimum threshold T_f. Furthermore, if the money transfer happens periodically from the same source to the same destination nodes but with slightly different smurfs, the probability of money-laundering fraud are even higher.

To avoid the errors illustrated in Fig. 4, SMoTeF-M computes maximum temporal flow by traversing edge occurrences of each journey j on its timestamp order so that we calculate maximum flow gradually. Furthermore, for each selected pattern we are interested in figuring out the most suspicious periods in which the fraud possibly has happened, to help experts understand the temporal aspect of patterns better. To accomplish this, we will observe the transition of money throughout time (as a flow time series data) for each smurfing flow network to find periods in which a significant amount of money has been transferred.

The SMoTeF-M method is described in Algorithm 3. Lines 2-11, computing the maximum temporal flow $Mtf(G)$ for each of the temporal networks G. Considering (G, s, d, J) as a smurf temporal flow network, first, we construct the time-ordered list of edges for each journey and call it $tol(j)$. $tol(j)$

Algorithm 3 SMoTeF-M.

```
   Data: OJ, Flow threshold T_f
   Result: OJ, Anomalous Periods AP
 1 FanInSum ← 0  fts ← ∅;
 2 for j in OJ do
 3 |   capacity(l+) ← 0;
 4 |   for e in tol(j) do
 5 |   |   if e in l+ then
 6 |   |   |   capacity(l+) += w_e;
 7 |   |   |   Continue;
 8 |   |   if e in l− then
 9 |   |   |   Mtf(s, d) = min(capacity(l+), w_e);
10 |   |   |   Mtf(G) += Mtf(s, d);
11 |   |   |   capacity(l+) -= Mtf(s, d);
12 if Mtf(G) < T_f then
13 |   OJ ← ∅ , AP ← ∅ ;
14 else
15 |   form fts from OJ;
16 |   MA ← moving − average(fts);
17 |   AP ← relative − extrema(MA);
18 return OJ, AP;
```

will be used later to traverse each journey and compute the maximum temporal flow and the residual value in each edge in fan-in l^+. If the visited edge in $tol(j)$ belongs to fan-in l^+, then we can not add up its flow to $Mtf(s, d)$ because the flow still needs to be consumed in the smurf node ($Mtf(s, d)$ represents maximum temporal flow for each journey). Next, if the visited edge in $tol(j)$ belongs to fan-out l^-, we can calculate the maximum temporal flow $Mtf(s, d)$. As the invalid temporal orders are now pruned, after each computation of $Mtf(s, d)$, we can accumulate the value $Mtf(s, d)$ of journey j into $Mtf(G)$. Yet, we need to subtract the amount of consumed flow from the capacity of only fan-in $capacity(l^+)$, because the residual in fan-out is the flow that the source does not contribute to it. Finally, if the maximum temporal flow $Mtf(G)$ is below the specified threshold, the pattern will be removed from the suspicious patterns (pruning step) and return empty sets. Otherwise, if the maximum temporal flow exceeds the flow threshold, algorithm will return the patterns' OJ alongside the investigated suspicious activity periods.

The last stage of our proposed method in lines 16-18 is for investigating suspicious periods. The amount of money flowing through the time in each fraud pattern forms a flow time series (fts) that will be used to find critical periods.

Our goal is to find the periods in which a significant quantity of money has been transferred through $v_s \rightarrow v_d$. To have a clear vision on data trends, we need to smooth the time series to avoid retrieving many undesirable peak periods. Therefore, we apply the moving average on fts before exploring for the suspicious peaks. The moving average has the main benefit of discarding trivial noise peaks in the data. Finally, we utilized the relative extrema to find interesting periods in our data. The relative extrema describe the points where the time series has maxima or minima (here

we are interested in maxima). Points of relative extrema can be acquired using the first derivative test by scanning the change of sign in the function's first derivative in the nearest neighborhood of that point.

To validate if the retrieved periods match an actual peak in money transfer in the pattern time series, we can visualize and then decompose the data into trend and cyclical components, as illustrated in Fig. 5. The red circles on the extrema points of the first chart are the suspicious months found by SMoTeF-M algorithm, fitting the actual high peaks that exist in the data.

We chose to apply the SMoTeF-T algorithm prior to SMoTeF-M because without this pruning phase, SMoTeF-M is computationally expensive when we have a massive set of journeys in the smurfing pattern and if there exist many edge occurrences in each journey. For these extreme cases, which can be expected in large dense networks, in each journey, we should traverse the edge occurrences on the temporal order to compute the maximum temporal flow (MTF). Instead, we chose to prune the invalid journeys (to prune clean and shady transfers) first and then compute MTF, which resulted in a trivial time overhead over the state-of-the-art baseline method which are shown in the experiments done in Section 5.

5 Experimental evaluation

In this section, we evaluate the performance of our proposed method using three real-world datasets, studying effectiveness as well as runtime efficiency.

Baselines: The proposed approach in this paper extends existing methods, e.g., [1, 2], by pruning non-fraudulent smurfing patterns. While these studies have contributed to the successful identification of smurfing patterns, their methodologies exhibit a high degree of specialization. These approaches are only practical in one specific domain, as they use unique features of graphs and associated node or edge meta-data to reduce dataset dimensions and suggest simplifying assumptions. This specificity makes the algorithms less adaptable and limits their usefulness across domains. Furthermore, relying on meta-data for preprocessing introduces additional constraints, as this information may not be readily accessible in certain fields due to confidentiality and security considerations.

The need to detect smurfing patterns in financial domains has led to various innovative approaches. AutoAudit [1], FlowScope [21] and CUBEFLOW [24] are most recent and state-of-the-art methods that share similarity in fighting money laundering in financial network. However, FlowScope and CUBEFLOW have limitations regarding practical application in the detection of the smurfing pattern type defined in this paper. FlowScope focuses on transfer chains which becomes impractical for smurfing pattern detection when only one intermediary involves in each journey from source to destination. CUBEFLOW also aims to find money laundering patterns consisting of multiple sources and destinations within transfer chains. Experiments using the datasets we used in this paper shown that this approach is also not practical in our scenario where transactions follow a linear path from a single source through one intermediary to a single destination.

Given these considerations and our review of the most recent and related cutting-edge research we chose AA-SMURF algorithm [1] as our baseline. Its unsupervised nature and successful application across various domains

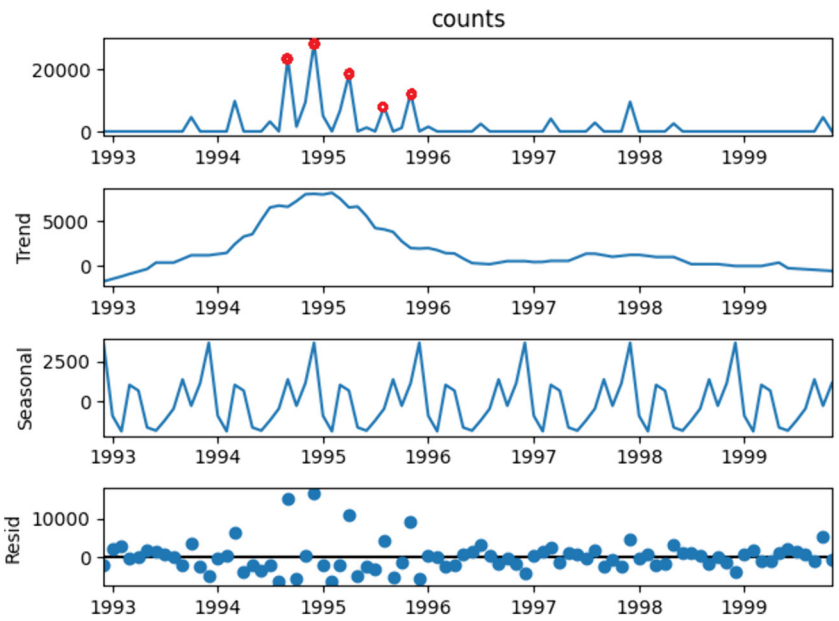


Fig. 5 Seasonality decomposition on a smurfing time series

Table 1 Statistics of the real-world datasets

Datasets	#Nodes	#Edges	Time Span
CFD	11.38K	273.51K	1993/01/01 – 1998/12/31
Enron	16.8K	1.04M	1998/01/06 – 2002/01/02
DARPA	371K	230K	87.7K minutes

have demonstrated its versatility, making it an appropriate foundation for our study. Unlike FlowScope and CUBEFLOW, AA-SMURF's adaptability makes it a more suitable candidate for our analysis, allowing us to address the specific challenges presented by smurfing with a more generalized and robust approach.

Evaluation metrics: We evaluate the performance on the datasets measuring recall, precision and F1-measure, to see how SMoTeF-T and SMoTeF-M perform over the baseline. However, due to the lack of ground truth for the Enron dataset, we had to turn to news feeds to verify our results. As the Enron company was involved in one of the largest accounting scandals and bankruptcy in recent history, important events related to the company were covered extensively in the news, and we could use the gathered information about Enron's activities and events leading up to its collapse. Efficiency is measured as runtime of of the baseline AA-SMURF, vs. AA-SMURF plus SMoTeF.

Experimental setup: The algorithms are implemented in Python, and the experiments are run on a server with dual 12-core 2.30GHz Intel Xeon Gold 5118 CPUs, with 384 GB RAM, using Ubuntu 22.04.

5.1 Datasets

In order to show the generality of our approach, we evaluated it on three real-world datasets, illustrating that it can be used not only to distinguish actual money laundering cases from false positive cases, but is also applicable in other domains in which anomalies form the same structure. The datasets' statistics are summarized in Table 1. We can categorize our datasets into two classes, datasets with ground truth (CFD and DARPA) and dataset without ground truth (Enron) (in which we used the news feeds to extract the important periods).

Czech financial dataset (CFD)[1] represents anonymous transfers of a Czech bank from January 1993 to December 1998, released for the Discovery Challenge in PKDD'99. The original CFD dataset provides information on the bank's clients, accounts, and transactions, loans, credit cards, etc. We use the information in the approx. 280K transactions provided in the dataset to model a financial graph.

DARPA intrusion detection dataset (DARPA)[2] has 4.5M IP-IP communications between 9.4K source IP and 23.3K destination IP over 87.7K minutes. Each communication is a directed edge (srcIP, dstIP, timestamp).

Enron email dataset (Enron)[3] is a collection of over 500k emails generated by 158 employees of the Enron Corporation in the years leading up to the company's collapse in December 2001. We use emails in the collection sent from 1998 to 2002. In this dataset each node represents an email address and edges represents the email interactions between nodes [29].

5.2 CFD

For CFD, we create smurfs that are injected into the graph.

5.2.1 Injecting smurfs

There are several approaches that evaluate their algorithms using injected money laundering patterns [1, 21, 24]. However, none of the these approaches consider the importance of temporal characteristics of edges and how this will impact the results of experiments. For instance, CUBEFLOW [24] and FlowScope [21] consider only the money flow amount in the edge characteristics, and AA-SMURF [1] inject smurfing patterns without any edge characteristics. We demonstrate later how ignoring edge characteristics can affect the false positive rates in the final results.

In order to inject synthetic smurfing patterns to CFD, fraudulent accounts were selected randomly from nodes as the distinct groups of v_s, V_m and v_d. Sender and receiver nodes are randomly chosen from client accounts and intermediaries from customer accounts, respectively. The number of intermediaries vary from 5 to 50 nodes, to show the performance with different number of intermediaries in the smurfing patterns.

We considered two main features to make the injected classes, amount of flow and temporal order. The low-flow attribute has been chosen randomly between the range of $[x_1, x_2]$ and high flow between the range of $[x_3, x_4]$ such that $x_2 \ll x_3$. We can choose lowest and highest values based on money transfer limits of the banks. We considered two modes to build valid and invalid temporal order to test our system in different conditions in order to be sure if it can discriminate the false cases even in complex conditions.

The injected smurfs are generated as follows.

- Valid temporal order has been constructed with two criteria:

[1] https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions

[2] https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset

[3] https://www.cs.cmu.edu/~enron/

  - For half of the journeys, we considered occurrences of l^+ and l^- edges, following the valid temporal order. Hence, all the l^+ edge occurrences happened before the occurrences of l^- edges, so we can have proper minimum temporal cuts.
  - For the remaining half of journeys, we ensured the inclusion of edge occurrences that deviate from the valid temporal order, while maintaining the correct temporal sequence for the rest of the l^+ and l^- occurrences. This approach aims to challenge the algorithm by presenting difficult cases, allowing for thorough testing of its ability to detect and appropriately prune invalid temporal orders, thereby enhancing its effectiveness.

- Invalid temporal order has been constructed with two criteria:
  - For half of the cases, for each journey, we considered multiple occurrences of l^+ and l^- edges, following the valid temporal order. Hence, all the l^+ edge occurrences happened before the occurrences of l^- edges.
  - For the remaining half, we considered some edge occurrences don't follow the valid temporal order and the rest of the l^+ and l^- occurrences follow the correct temporal order.

We employed the following categories of injected smurfs:

1. High flow and valid temporal order
2. High flow and invalid temporal order
3. Low flow and valid temporal order
4. Low flow and invalid temporal order
5. Periodic pattern with high flow and valid temporal order (highest probability of a true case of smurfing pattern)
6. Mixed fraudulent pattern with high flow: 80% valid temporal order, 20% invalid temporal order
7. Mixed non-fraudulent pattern with high flow: 20% valid temporal order, 80% invalid temporal order

5.2.2 CFD with injected smurfs

To assess the performance of our framework (both algorithms) on CFD, we conducted two experiments, aiming to evaluate the scalability and runtime efficiency of the SMoTeF framework in money laundering scenarios. In both experiments, all seven types of patterns were injected with varying frequencies. In the first experiment, as depicted in Fig. 6a, we systematically increased the number of injected patterns from 10 to 100 (comprising all seven patterns). The number of smurfs in each pattern was kept constant at 10. In the second experiment, illustrated in Fig. 6b, we incremented the number of smurfs in each pattern from 10 to 100 while maintaining a fixed number of patterns at 50. As evident in both experiments (see Fig. 6a and b), the runtime overhead of the SMoTeF framework over AA-SMURF is in the order of microseconds, i.e., insignificant compared to the initial search for smurfing patterns performed by AA-SMURF. Notably, the effect of increasing the number of smurfs, while maintaining a constant pattern size, was more evident in its impact on AA-SMURF's performance compared to our framework. Remarkably, our framework demonstrated consistent runtime performance in both scenarios. This observation suggests that the SMoTeF framework is robust to changes in the number of injected patterns and smurfs, maintaining stable performance even under varying parameters.

For the next experiment, we injected 20 smurfing patterns (each with 10 intermediaries) into CFD. Among 20 injections, we chose three valid patterns from categories 1,5 and 6, and we injected four patterns from each of the invalid patterns 2,3,4 and 7 to synthesize rare cases of anomalies. Results from this experiment is illustrated in Table 2 in which $\#v_m$ presents the number of smurf accounts. Note that in the following, whenever we are mentioning our methods SMoTeF-T and SMoTeF-M, both algorithms are running over AA-SMURF, so in all figures and tables we use

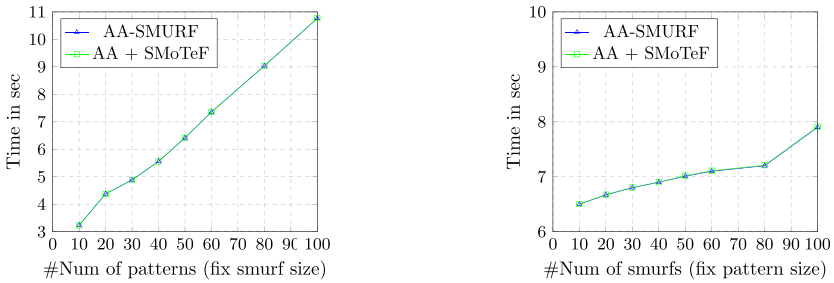


Fig. 6 Performance evaluation of SMoTeF framework on CFD

(a) Runtime performance of SMoTeF framework over AA-SMURF by increasing number of injected patterns while smurf size equals to 10

(b) Runtime performance of SMoTeF framework over AA-SMURF by increasing number of injected smurfs while pattern size equals to 50

Table 2 Comparison of proposed methods over baseline on the CFD dataset

Method	#v_m	#Inject	Precision	Recall	F1-Measure	# patterns
AA-SMURF	10	20	0.15	1	0.26	20
AA+SMoTeF-T	10	20	0.42	1	0.59	7
AA+SMoTeF-M	10	20	1	1	1	3

AA+SMoTeF-T and AA+SMoTeF-M and AA is used as the short form of AA-SMURF algorithm.

Performance results in Table 2 shows using both SMoTeF-T and SMoTeF-M over baseline method could successfully distinguish between valid and invalid patterns while keeping the high recall. However, in most complicated patterns, SMoTeF-M outperforms SMoTeF-T.

5.3 DARPA

The DARPA intrusion detection dataset consists of communications between source IPs and destination IPs addresses, with a timestamp. This dataset contains different attacks between IPs, containing Smurf distributed denial-of-service (DDoS), which form a smurfing pattern in the network communication graph. In a Smurf DDoS, a hacker (v_s) sends messages to intermediate host machines (v_m) with the spoofed source address of the victim machine to form a smurf DDoS attack. Then in response, the intermediate host machines flood the victim machine (v_d) with "ICMP Echo Reply" messages [30]. It was observed from the dataset that this attack happens usually in a very short time and it usually happens periodically with mostly the same smurf IPs engaged.

In the original DARPA ground truth, we have different types of attacks tagged to anomalous edges, which also contain a Smurf DDoS. The original version of the dataset contained around 1.2 M edges with only time as the edge attribute. We observed that many redundant edges with the same timestamp exist in the dataset as the result of fast communication in the internet network. Addressing this, instead of deleting redundant edges, we summarized all the same edge occurrences (u, v, t) that occurred in the same timestamp into one edge (u, v, t, c) containing the count of occurrences (c), which attribute c can be considered as the edge weight (w). Transforming DARPA into a transaction-like dataset changed the number of edges from 1M to 230k.

Results from experiments with SMoTeF-T and SMoTeF-M on the DARPA dataset are shown in Table 3. #v_m column corresponds to the number of smurf nodes and Flow shows the threshold amount of flow. Results from the Table 3 showed that both proposed methods improve the results of the AA-SMURF in precision, recall and F1-measure. Furthermore, to show the patterns found by AA-SMURF contain a high false positive rate, we also compared the number of retrieved smurfing patterns from all three methods. It can be observed that our methods significantly eliminate false positive patterns from AA-SMURF and provide better accuracy.

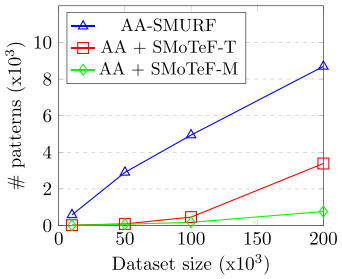
Figure 7b shows the runtime of our methods and AA-SMURF with increasing size of dataset, and again it can be seen that the overhead is trivial and that both methods scale linearly with AA-SMURF. Finally, Figure 7a shows the effects of our proposed method with varying number of found patterns. Similarly, as shown in Table 3, SMoTeF-M outperforms SMoTeF-T over AA-SMURF.

Figure 8a exhibits the distribution of intermediary size towards the retrieved patterns. From this figure, we can see that majority of patterns have small intermediary sizes (less than or equal to 5 intermediaries).

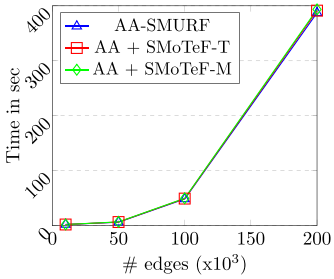
Quite a few patterns have more than 30 intermediaries involved (8 patterns), which is expected because, in the traffic network, we don't anticipate DDoS smurf attacks to happen

Table 3 Comparison of our algorithm against the baseline (AA-SMURF) on the DARPA dataset

Method	#v_m	Flow	Precision	Recall	F1-Measure	# patterns
AA-SMURF	5	20	0.636	1	0.77	8689
AA+SMoTeF-T	5	20	0.76	1	0.87	615
AA+SMoTeF-M	5	20	0.78	1	0.88	414
AA-SMURF	10	40	0.74	1	0.85	3388
AA+SMoTeF-T	10	40	0.83	1	0.91	279
AA+SMoTeF-M	10	40	0.90	1	0.92	228
AA-SMURF	15	100	0.858	1	0.92	768
AA+SMoTeF-T	15	100	0.93	1	0.97	62
AA+SMoTeF-M	15	100	0.97	1	0.97	53



(a) Comparing our methods with the baseline: Number of true patterns identified as dataset size increases (pruning)



(b) Comparing the runtime performance of our methods with AA-SMURF on patterns involving 10+ intermediaries and a maximum flow of 150+

Fig. 7 Performance evaluation of SMoTeF-T and SMoTeF-M on DARPA

abundantly as anomalous events rarely occur. Besides, it reveals another characteristic of these attacks that, though rare, will be large-scale with many intermediaries involved.

Figure 8b depicts the distribution of flow within retrieved patterns. As a characteristic of a network, we expect that in the majority of patterns, there are a few connection requests to and from the few intermediaries, which contribute to a small flow within most of the smurfing pattern networks. Based on this, those patterns transfer a massive amount of flow inside them, revealing the enormous amount of requests from involved nodes to the sink node, resulting in higher probability of being an anomalous pattern. These two figures shows the power-law distribution of anomalous pattern characteristics in the DARPA dataset.

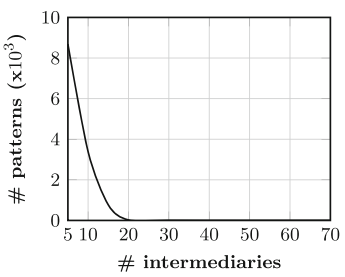
5.4 Enron

Among the three datasets, Enron has no ground truth. Enron has only the time attribute on the edges, so we used SMoTeF-T algorithm, which completely ignores the journeys that don't have a valid temporal order (those journeys that don't contain a minimum temporal cut). It can be interpreted such that for an intermediary node v_m, no emails can be forwarded from source node v_s to destination v_d through v_m. Here we assumed that the edge weight w for all the edge occurrences is the same and equal to one.

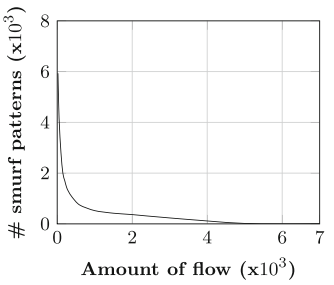
Smurfing patterns commonly occur in a company's email network due to an email forwarding mechanism throughout time, random and without any intentions. Therefore, we are interested in those smurfing patterns that at least have journeys through intermediaries that follow the valid temporal order; otherwise, the chances of them being an anomalous occurrence is very low.

The Enron dataset contains two types of smurfing anomalies. Type one contains a considerable number of intermediaries, which means anomalousness regarding the spread of news among the employees. Type one anomalies mainly happened when the company announced bankruptcy around December 2001 and January 2002. This anomaly can be detected by SMoTeF framework by extracting the anomalous periods in which a peak in the number of valid smurfing patterns detected. We verified the retrieved anomalous periods from our devised framework with the important days and events of Enron company from news feeds.

Type two is an indicator of insider trading, which involves few intermediaries and is the most complicated since there are a vast majority of patterns with small intermediary sizes as the power-law distribution shown in Fig. 10, though only a few are anomalous. Anomaly type 2 presents a challenge



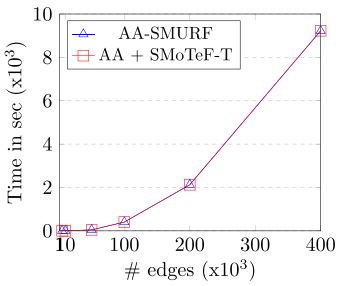
(a) Distribution of # intermediaries in retrieved smurfing patterns



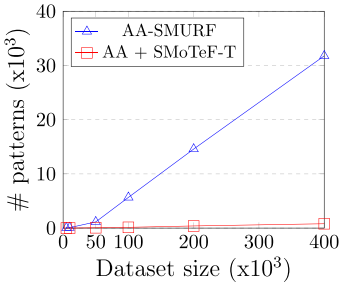
(b) Distribution of flow on smurfing patterns with 5+ intermediaries

Fig. 8 Distribution of smurfing pattern characteristics on DARPA

Fig. 9 Performance evaluation of SMoTeF-T algorithm on Enron



(a) Comparing the runtime performance of SMoTeF-T with AA-SMURF on patterns involving 10+ intermediaries



(b) Comparing our method with the baseline: Number of true patterns identified as dataset size increases (pruning)

for mapping news events to the patterns found due to the significant number of retrieved smurfing patterns in comparison to the number of patterns in DARPA. Moreover, identifying the specific event in the news that relates to a pattern indicating insider trading is exceedingly difficult. Thus, we have focused our algorithm on eliminating temporal invalid journeys, which correspond to occurrences that do not contribute to a valid smurfing pattern. This anomaly can be discovered precisely by exploring if the nodes participate in a pattern, belong to different departments that have few communications so that it shows the exchange of information.

Figure 9a shows the runtime of the SMoTeF-T algorithm compared to AA-SMURF, again illustrating that the overhead is minimal. However, as can be seen in Fig. 9b, it significantly prunes the temporal invalid patterns that can not contribute to fraud. SMoTeF-T retrieved all the noteworthy patterns for more investigation and the period associated with each.

The distribution of intermediary size towards the retrieved patterns after applying SMoTeF-T is shown in Fig. 10. The majority of patterns have small intermediary sizes (less than or equal to 5 intermediaries), while a few patterns contain a high number of smurfs, ranging from 60 to 70. Patterns with high number of intermediaries can provide valuable insights into the smurfing activities within a company's email network, especially anomaly type one.

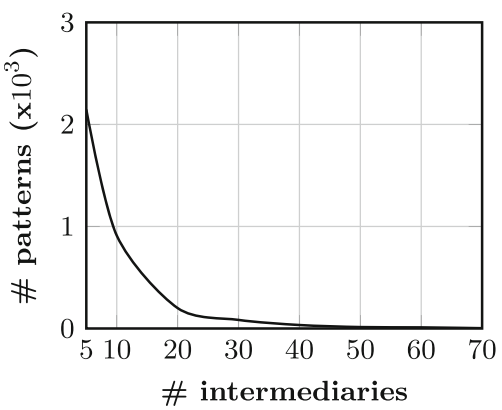


Fig. 10 Distribution of # intermediaries in retrieved patterns on Enron

Compared with the two other datasets used in this paper, Enron had the highest concentration of smurfing patterns, especially between October 2001 and February 2002, when most of the bankruptcy-related events occurred. This finding is supported by Table 1, which shows that Enron has the highest density of edges over the nodes among the datasets used in the study. Thus, the analysis of Enron's smurfing patterns can reveal critical information about the illegal activities that led to the company's downfall, and which can be employed to predict similar trends in other networks.

5.5 Impact analysis of key parameters

In this section, we demonstrated the effectiveness and scalability of the SMoTeF framework in real-world scenarios. The subsequent discussion will focus on clarifying the impact of the two critical characteristic thresholds (T_s and T_f) on the obtained results. In practical scenarios, criminals can manipulate the topology concerning the number of smurf accounts (T_s), thereby trying to deceive the anti-money laundering system. In order to evaluate the algorithm's robustness in this diverse context, it is essential to provide a clear explanation of the optimal value of chosen thresholds.

The influence of the density of money (T_f): In money laundering scenarios, the overall amount of money transferred through intermediaries is crucial in distinguishing genuine instances of money laundering from false positives. For each financial institution, the threshold for acceptable levels of money laundering may differ, and for security reasons, institutions keep this threshold undisclosed. To address this challenge, we introduced patterns characterized by distinctive overall amounts of money transfers with and without consideration of temporal order. This process evaluates our algorithm's performance across diverse money flows, which results in the accuracy achieved in Table 2.

In the DARPA dataset, the weight (denoted as w) assigned to each edge within a given timestamp is determined by the count of edge occurrences at that specific timestamp. Notably, in a genuine Smurf DDoS attack in DARPA, the number of smurfs involved can reach up to 7000. Likewise, the optimal flow volume from source to destination varies approximately between 5000 and 7000, as illustrated in the Fig. 8b. This figure highlights a limited number of patterns capable of facilitating this specific flow volume through their intermediaries.

It's worth noting that the Enron graph did not contain the concept of edge weights w, thereby no experiment can be done with respect to T_f.

In conclusion, the volume of money flow within a pattern, whether substantial or minimal, doesn't affect the performance of the algorithms. When a pattern passes through SMoTeF-M, the algorithm's performance depends only on the number of journeys within the pattern, which determines the number of iterations performed (Algorithm 3 lines 2-4).

The influence of the number of smurf accounts (T_s): Another essential consideration involves fraudsters' potential use of multiple intermediaries in money laundering activities. However, such a strategy may lead to the formation of detectable dense subgraphs. To lower this risk, fraudsters might employ considerably fewer intermediaries resembling random smurfing behavior, making it more challenging to identify. This explanation guided our approach in the CFD dataset experiment, where we incrementally raised the number of smurfs at a fixed ratio of 5. Throughout this process, we maintained consistent levels of money laundering and other conditions to assess the performance of our algorithms.

In the context of Smurf DDoS attacks, as examined in the DARPA dataset, the involvement of a considerable number of intermediaries proves crucial in minimizing false positive patterns. This is particularly relevant because Smurf attacks can employ many intermediary nodes on the network, unknowingly participating in the attack. The amplification effect of the attack increases with the larger number of smurf nodes. Notably, the efficient threshold for real smurfing patterns ranges from 150 to 7000 intermediaries in the DARPA dataset.

5.6 Limitations and challenges

In presenting the SMoTeF framework for detecting smurfing patterns in financial networks, we acknowledge certain limitations and challenges. One notable challenge lies in the potential extension of the proposed methods to address layered smurf patterns or chain structures, where instead of only one intermediary in the middle, there are chains of intermediaries.

Moreover, determining the two main parameters, the smurf and flow threshold, crucially relies on extracting values based on the specific properties of the graph network within its application domain. This process requires the knowledge and expertise of professionals specializing in the domain and understanding its complexities. Although our current framework significantly reduces the false-positive rates of existing methods, extracting these parameters may introduce variability and subjectivity, potentially affecting the algorithm's performance.

5.7 Broader application domains

The SMoTeF framework has proven its effectiveness for detecting smurfing in financial, internet traffic, and email networks. The temporal dynamic presented in this paper can also be adapted to other areas that analyze sequential and dynamic data. In this context, we will explore its application to natural language processing (NLP), human activity recognition (HAR), and social network analysis.

In the field of NLP, SMoTeF's temporal dynamics can assist with document summarization and processing by removing irrelevant content. Motivated by techniques in [31], SMoTeF can be adapted to model documents as temporal graphs. This approach represents text as a graph of sentences and key phrases connected by edges. SMoTeF's temporal dynamics algorithm prunes the graph edges that do not contribute to the overall narrative flow, similar to the compression algorithm in [31]. Thus, integrating SMoTeF could lead to more effective summarization tools in NLP, capable of handling large volumes of information with temporal precision.

For HAR, temporal dynamics can be utilized to optimize the recognition of human activities through sensor data by modeling a graph of temporal interaction besides the spatial properties of activities [32]. Adapting SMoTeF to this area, series of movements can be depicted as nodes in a graph, and the transitions between them are represented as edges that follow a logical temporal order. To simplify the graph structure, SMoTeF can use temporal dynamics to eliminate illogical transitions or possible errors. For instance, a sudden shift from a high energy activity to a static activity state would be removed. This optimized graph structure can improve the accuracy of activity recognition.

In social network analysis, understanding the temporal dynamics of actions plays a pivotal role in detecting fake news effectively. By analyzing the evolving patterns of user interactions and semantic similarities, we can identify subtle shifts that indicate the spread of misinformation. Recent research [33] has shown that dynamic graphs that capture the temporal evolution of user networks are essential for this purpose. Using SMoteF's temporal dynamics, we can eliminate irrelevant edges, such as out of order interactions or automated bot activities often associated with fake news propagation. By focusing on the temporal aspect of user behavior, we can improve the precision of our detection

algorithms, making it easier to distinguish between genuine content and misinformation.

Each application highlights the versatility of SMoTeF's temporal dynamic technique, indicating its potential to extend beyond financial fraud detection to contribute significantly to various fields that can be modeled using a temporal graph.

6 Conclusion

In this paper, we have presented the SMoTeF framework, which contains two algorithms that extends existing graph-based smurf detection methods, distinguishing fraudulent smurfing patterns from non-fraudulent ones. In order to achieve this, the new algorithms compute maximum temporal flow by evaluating the temporal order of edge occurrences in a smurfing flow network.

Experiments on three real-world datasets from different domains reveal that both methods significantly reduce the false positives in the smurfing pattern result-set from the state-of-the-art baseline. Moreover, our methods could spot suspicious activity periods for each fraud-suspected pattern, which can provide financial administrators and auditors with further information on each pattern.

In our future work, we intend to extend the proposed methods to handle chain of smurf patterns (layering fraud), to reduce the false positive rate on that structure. To further increase the performance of our approach within financial networks, we plan to adapt the framework to graph stream environments.

Acknowledgements The research in this paper was funded by the collaborative project of DNB ASA and the Norwegian University of Science and Technology (NTNU). The authors would like to thank the anonymous reviewers for their fruitful comments that helped in clarifying the technical details of our approach.

Funding Open access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital).

References

1. Lee M-C, Zhao Y, Wang A, Liang PJ, Akoglu L, Tseng VS, Faloutsos C (2020) Autoaudit: Mining accounting and time-evolving graphs. In: 2020 IEEE International Conference on Big Data (Big Data), pp. 950–956
2. Starnini M, Tsourakakis CE, Zamanipour M, Panisson A, Allasia W, Fornasiero M, Puma LL, Ricci V, Ronchiadin S, Ugrinoska A (2021) Smurf-based anti-money laundering in time-evolving transaction networks. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 171–186
3. Corselli L (2020) Italy: Money transfer, money laundering and intermediary liability. Journal of Financial Crime
4. Monamo PM, Marivate V, Twala B (2016) A multifaceted approach to Bitcoin fraud detection: Global and local outliers. In: Proceedings of the 15th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 188–194
5. Chen T, Tsourakakis C (2022) AntiBenford subgraphs: Unsupervised anomaly detection in financial networks. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '22, pp. 2762–2770
6. Choudhury S, Holder L, Chin G, Ray A, Beus S, Feo J (2013) StreamWorks: a system for dynamic graph search. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1101–1104
7. Kumar R, Gupta BB (2016) Stepping stone detection techniques: Classification and state-of-the-art. In: Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing, pp. 523–533
8. Kumar S (2007) Smurf-based distributed denial of service (DDoS) attack amplification in internet. In: Second International Conference on Internet Monitoring and Protection (ICIMP 2007)
9. Zhang Y, Paxson V (2000) Stepping stone detection. In: Proceedings of the 2000 USENIX Security Symposium
10. Akrida EC, Czyzowicz J, Gasieniec L, Kuszner Ł, Spirakis PG (2019) Temporal flows in temporal networks. Journal of Computer and System Sciences, 46–60
11. Kurshan E, Shen H (2020) Graph computing for financial crime and fraud detection: Trends, challenges and outlook. International Journal of Semantic Computing. 14(04):565–589
12. Jiang Y, Liu G (2022) Two-stage anomaly detection algorithm via dynamic community evolution in temporal graph. Appl Intell 52(11):12222–12240
13. Pourhabibi T, Ong K-L, Kam BH, Boo YL (2020) Fraud detection: A systematic literature review of graph-based anomaly detection approaches. Decision Support Systems. **133**
14. Chen Z, Van Khoa LD, Teoh EN, Nazir A, Karuppiah EK, Lam KS (2018) Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review. Knowl Inf Syst 57(2):245–285
15. Hilal W, Gadsden SA, Yawney J (2021) A review of anomaly detection techniques and applications in financial fraud. Expert Systems with Applications 116429
16. Wang H, Yang R, Shi J (2023) Anomaly detection in financial transactions via graph-based feature aggregations. Big Data Analytics and Knowledge Discovery. Springer, Cham, pp 64–79
17. Huang D, Mu D, Yang L, Cai X (2018) Codetect: Financial fraud detection with anomaly feature detection. IEEE Access. 6:19161–19174
18. Chetia A, Buragohain N, Mazumder SH, Singh MT (2023) Abnormality detection in financial transactions using graph representation learning. In: International Conference on Data Analytics and Insights, pp. 135–145. Springer

19. Jin M, Liu Y, Zheng Y, Chi L, Li Y-F, Pan S (2021) Anemone: Graph anomaly detection with multi-scale contrastive learning. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 3122–3126
20. Patel V, Pan L, Rajasegarar S (2020) Graph deep learning based anomaly detection in ethereum blockchain network. In: International Conference on Network and System Security, pp. 132–148. Springer
21. Li X, Liu S, Li Z, Han X, Shi C, Hooi B, Huang H, Cheng X (2020) FlowScope: spotting money laundering based on graphs. Proceedings of the AAAI Conference on Artificial Intelligence 34:4731–4738
22. Ranshous S, Shen S, Koutra D, Harenberg S, Faloutsos C, Samatova NF (2015) Anomaly detection in dynamic networks: a survey. Wiley Interdisciplinary Reviews: Computational Statistics 7(3):223–247
23. Eswaran D, Faloutsos C, Guha S, Mishra N (2018) Spotlight: Detecting anomalies in streaming graphs. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1378–1386
24. Sun X, Zhang J, Zhao Q, Liu S, Chen J, Zhuang R, Shen H, Cheng X (2021) CubeFlow: money laundering detection with coupled tensors. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 78–90
25. Sun X, Feng W, Liu S, Xie Y, Bhatia S, Hooi B, Wang W, Cheng X (2022) MonLAD: money laundering agents detection in transaction streams. In: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining. WSDM '22, pp. 976–986
26. Chang Y-Y, Li P, Sosic R, Afifi MH, Schweighauser M, Leskovec J (2021) F-FADE: frequency factorization for anomaly detection in edge streams. In: Proceedings of the 14th ACM International Conference on Web Search and Data Mining. WSDM '21, pp. 589–597
27. Jiang J, Li Y, He B, Hooi B, Chen J, Kang JKZ (2022) Spade: A real-time fraud detection framework on evolving graphs. Proceedings of the VLDB Endowment. 16(3):461–469
28. Ford LR, Fulkerson DR (1956) Maximal flow through a network. Canad. J. Math. 8:399–404
29. Klimt B, Yang Y (2004) The Enron corpus: A new dataset for email classification research. In: European Conference on Machine Learning, pp. 217–226
30. Chin G, Choudhury S, Feo J, Holder L (2014) Predicting and detecting emerging cyberattack patterns using StreamWorks. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference. CISR '14, pp. 93–96
31. Li M, Ma T, Yu M, Wu L, Gao T, Ji H, McKeown K (2021) Timeline summarization based on event graph compression via time-aware optimal transport. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6443–6456
32. Chen Y, Zhu H, Chen Z (2024) Multi-dgi: Multi-head pooling deep graph infomax for human activity recognition. Mobile Networks and Applications, 1–12
33. Plepi J, Sakketou F, Geiss H-J, Flek L (2022) Temporal graph analysis of misinformation spreaders in social media. In: Proceedings of TextGraphs-16: Graph-based Methods for Natural Language Processing, pp. 89–104

Shiva Shadrooh is a PhD candidate in Computer Science at the Norwegian University of Science and Technology (NTNU). She received her MSc in Computer Software Engineering from Shahid Beheshti University. Her research interests include graph mining, deep learning and databases, and, with a focus on dynamic graph networks.

Kjetil Nørvåg received his MSc and PhD degrees from the Norwegian University of Science and Technology (NTNU). He is currently Professor in the Department of Computer Science at NTNU. His research interests include distributed and parallel database systems, query processing, information retrieval, and Big Data in general. Further details concerning his work can be found in https://www.ntnu.no/ansatte/noervaag.