

RESEARCH

Open Access



# GraphSIF: analyzing flow of payments in a Business-to-Business network to detect supplier impersonation

Rémi Canillas<sup>1,2\*</sup> , Omar Hasan<sup>1</sup>, Laurent Sarrat<sup>2</sup> and Lionel Brunie<sup>1</sup>

\*Correspondence:

[remi.canillas@insa-lyon.fr](mailto:remi.canillas@insa-lyon.fr)

<sup>1</sup>University of Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, Lyon, France

<sup>2</sup>SIS-id, 26 Boulevard Eugène Deruelle, 69003 Lyon, France

## Abstract

Supplier Impersonation Fraud (SIF) is a rising issue for Business-to-Business companies. The use of remote and quick digital transactions has made the task of identifying fraudsters more difficult. In this paper, we propose a data-driven fraud detection system whose goal is to provide an accurate estimation of financial transaction legitimacy by using the knowledge contained in the network of transactions created by the interaction of a company with its suppliers. We consider the real dataset collected by SIS-ID for this work.

We propose to use a graph-based approach to design an Anomaly Detection System (ADS) based on a Self-Organizing Map (SOM) allowing us to label a suspicious transaction as either legitimate or fraudulent based on its similarity with frequently occurring transactions for a given company. Experiments demonstrate that our approach shows high consistency with expert knowledge on a real-life dataset, while performing faster than the expert system.

**Keywords:** Fraud detection, Graph-based feature engineering, Financial networks, B2B network

## Introduction

Fraud is a recurring issue in many domains such as credit card transactions, insurance, telecommunication, and finance. Supplier Impersonation Fraud (SIF) is a specific case of identity theft, targeting a company rather than an individual. This type of financial fraud is widespread, resulting in the loss of hundreds of thousands Euros in 2018, and ranked 1st most frequent fraud affecting French companies in the latest survey about cyber-criminality conducted in 2019 by Euler Hermes and DFCG (2019). Supplier impersonation consists of a fraudster impersonating a member of a company providing goods and services to another, in order to trigger a payment on an account controlled by the fraudster AIG (2019).

We can illustrate SIF with a toy example. Let  $C$  be a company that produces computers. In order to acquire the necessary components for the fabrication,  $C$  buys electronic chips from  $S$ . Let  $F$  be a fraudster. A SIF takes place when  $F$  diverts a payment from  $C$

originally destined to  $S$ , usually by impersonating  $S$  in  $C$ 's eyes. Such an impersonation can take several forms: the tampering of an invoice from  $S$  (similar to phishing attacks), impersonating a high-ranking employee of  $S$  and requesting the next payments to be paid on an account controlled by the fraudster, or even requesting the payment of imaginary goods and services on behalf of  $S$ .

As more and more companies are using digital tools to process transactions due to numerous advantages provided by digitalization, the risk of supplier impersonation has never been higher. However, the tools and systems required to detect SIF have not evolved at the same pace, and still mostly rely on expert-based input and monitoring. This approach might not be the most suitable to handle the increasing volume of digital transactions. Thus, the need for automated, data-driven SIF detection system arises.

In this work, we present GraphSIF, a SIF detection system that uses a Business-To-Business (B2B) transactions dataset to construct a graph modeling the relationships between client and supplier companies in a B2B ecosystem, and describe how these relationships can be used to derive useful knowledge in order to assert the legitimacy of transactions.

We use the transaction network to create a time-evolving behavior sequence summing up the evolution of the graph through time. We then compare the new graph created by adding a suspicious transaction to the behavior sequence and investigate the potential discrepancy it introduces. If this discrepancy is low then the transaction is considered as legitimate, and if the discrepancy is high then the transaction is considered as likely fraudulent. In order to quantify this discrepancy, a Self-Organizing Map (SOM) is trained on the behavior sequence, and a clustering algorithm is used to quantify the similarity of the tested graph with the ones in the behavior sequence.

Finally, we analyze the results of GraphSIF using a set of transactions labeled by experts from the SiS-id company, in order to evaluate its performance, and investigate its potential shortcomings. We found that GraphSIF with the selected parameters shows high consistency with the expert system when focusing on low-legitimacy transactions.

### **Contributions**

The contributions of this paper are the following: a graph-based feature engineering process relying on a bipartite graph constructed from transactions between companies in a B2B context, a classification system that uses Self-Organizing Maps and K-means clustering to investigate the legitimacy of a new transaction, and a comprehensive evaluation of the proposed classification system using data from a real-life B2B ecosystem.

### **Previous work**

This paper extends the conference paper (Canillas et al. 2019) presented in Complex Networks 2019, by proposing a more thorough description of the algorithms used to perform the fraud detection process, and by proposing an analysis of the system's performance on a large number of companies.

### **Related work**

Due to the sensitivity of the data linked to supplier fraud detection for victim companies, we have not been able to find any publicly available research work directly related

to SIF. However, we can find several systems designed for fraud detection that also use a network-based approach:

In Sadowksi and Rathle (2014) several network-based fraud detection use-cases are introduced, showing examples of successful use of graph theory to detect bank fraud, insurance fraud and e-commerce fraud. However, the authors focus on specific industrial examples without proposing a formalized evaluation of their solution.

Akoglu, Tong and Koutra Akoglu et al. (2015) propose a survey on anomaly detection using graphs, notably in the domain of telecommunication fraud detection. An approach closely related to our own is found in Akoglu et al. (2010) where an egonet (1-step neighborhood graph) is used to derive features describing a node. However, this approach is applied to a static graph that does not evolve through time, contrary to our approach. The analysis of dynamic graphs through the use of windows, as it is the case in our work, is akin to the ideas developed in Priebe et al. (2005) and Mongiovì et al. (2013) where the graphs are analyzed using a moving window and detecting anomalous connectivity variation Priebe et al. (2005) or edges  $p$ -value variation (Mongiovì et al. 2013). To the best of our knowledge there is no previous attempt at combining a window-based analysis as seen in Priebe et al. (2005) with the feature approach used in Akoglu et al. (2015).

Van Vlasselaer et al. (2016) proposes an approach somewhat similar to ours, using graphs to represent interactions between companies in order to detect social security frauds. However, their work focuses on the application of social network algorithms on large graphs of interconnected entities, whereas our work considers smaller neighborhood graphs, focused on the behavior of a single company. In addition, the system proposed in Van Vlasselaer et al. (2016) bases itself on a propagation algorithm and Random Logistic Forests to perform their analysis, and does not make use of Self-Organizing Maps.

Furthermore, most of the aforementioned research uses supervised algorithms as they rely on the existence of known legitimate and fraudulent neighboring nodes to conduct their analysis. Our work proposes an unsupervised approach that does not take into account the legitimacy of neighboring nodes to propose a label, but instead uses the topology of the ego network.

Finally, as our approach makes use of significant patterns found in the 2-step egonet (the set of vertices found at most 2 edges away from the considered vertex) of a company, a parallel can be drawn with the discovery of network motifs, for example, as presented in Milo et al. (2002). However, the discovery of network motifs relies on the observation of patterns occurring with a statistically significant number of occurrences compared with random graphs showing similar topology. Our approach uses a different technique to create patterns from the egonet, by investigating the remaining connected sub-graphs when the ego node is removed. The techniques used in network motif analysis could however be used to discover the occurrences of payment motifs across for a number of targeted companies and thus assess if recurring payments behavior are shared in the payment ecosystem.

To the best of our knowledge, our work is the first to use a graph-based approach paired with Self-Organizing Maps in order to address the issue of SIF detection.

**Table 1** Structure of the History data record. The History dataset is composed of 950,929 records collected from 6,063 companies over two years

Feature	Type	Description
Client	Nominal (ID)	Identification number of the client issuing the transaction.
Supplier	Nominal (ID)	Identification number of the supplier receiving the transaction.
Account	Nominal (ID)	Identification number of the bank account to which the money is transferred.
Date	Timestamp (Month)	Timestamp indicating the date when the transaction took place.

### SiS-id fraud detection platform

While SIF is a widespread fraud, it is mostly dealt with internally by companies victim of the fraud. There are three main reasons that motivate the lack of collaboration in SIF detection: firstly, being public about being victim of a fraud can cause a breach of trust and bad publicity for the company. Secondly, due to the competitive nature of the Business-to-Business ecosystem, information about the relationship between a client and its suppliers is a sensitive knowledge that could lead to economic attacks if divulged. Finally, having a successful in-house fraud detection system provides an edge for the company that owns it, and thus such a system would not be willingly shared with competitors. However, the cost of creating and maintaining complex fraud detection systems is sometimes prohibitive for a large number of companies.

In this context, the company SiS-id<sup>1</sup> proposes to act as a trusted third party focused on Supplier Impersonation Fraud detection and mitigation. The company, started in 2016, develops several SIF fraud mitigation tools that other companies can use “as a service”. SiS-id emphasizes strongly on the privacy and security of the data shared by their clients that they use to develop detection techniques. Currently, SiS-id proposes two SIF mitigation systems: firstly a fraud detection system based on the relationship network created in the data shared by each of its clients, and secondly a secure repository for trusted bank accounts corresponding to verified suppliers. In this paper, we propose a data-driven system that proposes a way to improve SiS-id’s current expert-based system to perform SIF detection.

The remainder of this section describes the dataset available to SiS-id to perform SIF detection, along with the system they implemented on the platform.

### History dataset

The SiS-id SIF detection system uses a set of historical transactions performed by SiS-id’s clients. In this section, we describe this dataset in detail.

The set of B2B transactions used by the SiS-id detection system is an aggregation of the payments performed by SiS-id’s client companies between July 2016 and July 2019. These transactions consists of a feature vector of 4 features : client identification number, supplier identification number, target account identification number, and date of the payment. For the sake of storage, all of the transactions involving the same client, supplier and destination account during a single month are aggregated, resulting in the creation of a fifth feature representing the number of similar transactions issued during the month. The time granularity of the transaction is thus a month. Table 1 shows an overview of the features.

<sup>1</sup><https://www.sis-id.com/>

The amount of payment is a feature found in most financial fraud detection systems (as seen in Bolton et al. (2002)). However, this data is very critical for companies (as described in “[SiS-id fraud detection platform](#)” section), this feature was not shared with SiS-id by the companies that agreed to collaborate on the creation of the History dataset. Indeed, disclosing the amount of the transactions issued to their suppliers might divulge economic insights regarding their financial well-being, as well as provide a useful baseline for potential frauds. Thus, GraphSIF relies only on payments without their amounts.

In order to preserve the confidentiality of the data, a secure hash function is applied to the three distinct identifiers (client, supplier, account) so that no link can be established between the data in the history and real-life companies. While this mitigates the risks of damage in case of data leak, it also means that the same company will have a different identifier whether it has issued a transaction, or received a transaction.

At the time of writing, 950,929 transaction records are available in the History dataset. These transactions are issued by 6,063 unique companies. This number is more than the number of SiS-id’s client companies. This is explained by the fact that SiS-id’s client companies can represent a group of several branches such as a multinational group. In this case, each firm possesses its own identification number, but only a global entity will be SiS-id’s client. 215,056 unique supplier companies are also found in the dataset, along with 262,157 unique bank accounts to which a payment was transferred. The fact that more bank accounts than supplier companies exist indicates that some suppliers use more than one bank account to be paid.

This dataset represents all the transactions performed by all of SiS-id clients for two years. However, there is no available information about the legitimacy of these data, and thus no knowledge of which transactions are fraudulent and which are legitimate.

Due to the economic sensitivity of the data for the companies (as discussed in “[SiS-id fraud detection platform](#)” section), no sample can be made publicly available at the moment. However, a request for data samples can be submitted directly to SiS-id through their website<sup>2</sup>, or by contacting Laurent Sarrat, co-author of this paper.

### **Audit dataset**

A second set of transactions is available thanks to SiS-id. It consists of the list of transactions that were analyzed using the expert system in the past 2 years (July 2017 - July 2019). The dataset, called the “Audit” dataset, is composed of 218,325 suspicious transactions submitted by 317 unique client companies. The transactions underwent the fraud detection process devised by SiS-id, and were labeled with a legitimacy label: “high” indicate that the transaction have a high chance of being legitimate, “medium” meaning that the rule engine lacks the necessary information to assert if the transaction is legitimate or not, and “low” means that the transaction’s legitimacy is low, which can be the case if the transaction is either invalid or fraudulent.

This dataset possesses the following properties:

- 1 It contains real life queries for transaction validation made by companies.
- 2 Each of the transaction of this dataset contains a target variable corresponding to the classification done by the rule engine.
- 3 The transactions were cross-validated by the clients for consistency.

<sup>2</sup><https://www.sis-id.com/#contact>

The legitimacy label found in the dataset might tempt us to use this dataset to perform supervised learning. However, this approach has a major drawback: by using data analysis on a dataset that is the result of the rule-engine system (described in the next section), we will only manage to “rediscover” the rules. However, this dataset might be used as a validation set for other fraud detection systems, in order to compare their performance with SiS-id’s expert system and investigate the potential convergence of their results.

### SiS-id expert system

The fraud detection system SiS-id currently runs on its platform<sup>3</sup> is an expert fraud detection system, where a potentially fraudulent transaction is examined in order to assert its legitimacy, using knowledge available on the platform. This kind of systems inherits directly from the tradition of fraud detection teams Kim and Sohn (2012), and aims to formalize their knowledge in order to efficiently process a large number of transactions. The fraud detection system designed by SiS-id consists of two separate steps: a feature engineering step where the features from the tested transaction are used to gather additional information, and then the gathered data is matched against a set of expert-defined rules in order to assert the transaction’s legitimacy. For confidentiality reasons, it is not possible to discuss the inner workings of the system in detail. However, it has been designed by a team of SIF detection experts and thus provides a valid approximation of the expert knowledge.

### GraphSIF overview

GraphSIF is a SIF detection system based on anomaly detection that uses the relationships created between the companies interacting in a B2B environment in order to determine the legitimacy of a transaction, given the company that issued the payment.

This system is composed of four phases:

- 1 A pre-processing phase, where historical transactions are sorted by the companies that issued them, and grouped in time windows in order to describe a time-evolving sequence composed of several fixed-length windows of transactions, describing the behavior of a specific client. This phase is described in “[Transactions pre-processing](#)” section.
- 2 A feature engineering phase where each of the windows of transactions is transformed into a graph. This graph sums up the interactions that occurs between the client and its suppliers during the specified windows. “[Graph-based feature engineering](#)” section describes this phase.
- 3 An anomaly detection phase where a specific transaction (the “suspicious” transaction) is added to the most recent graph, creating a “test graph”. This graph’s similarity with the ones occurring in the historical sequence is computed. “[Anomaly detection](#)” section provides more details about this phase.
- 4 A label attribution phase where the similarity of the test graph given a set of different sizes of the windows of transactions are aggregated. A legitimacy label derived from the aggregation score is computed. Details about this phase are found in “[Label attribution](#)” section.

---

<sup>3</sup><https://my.sis-id.com>

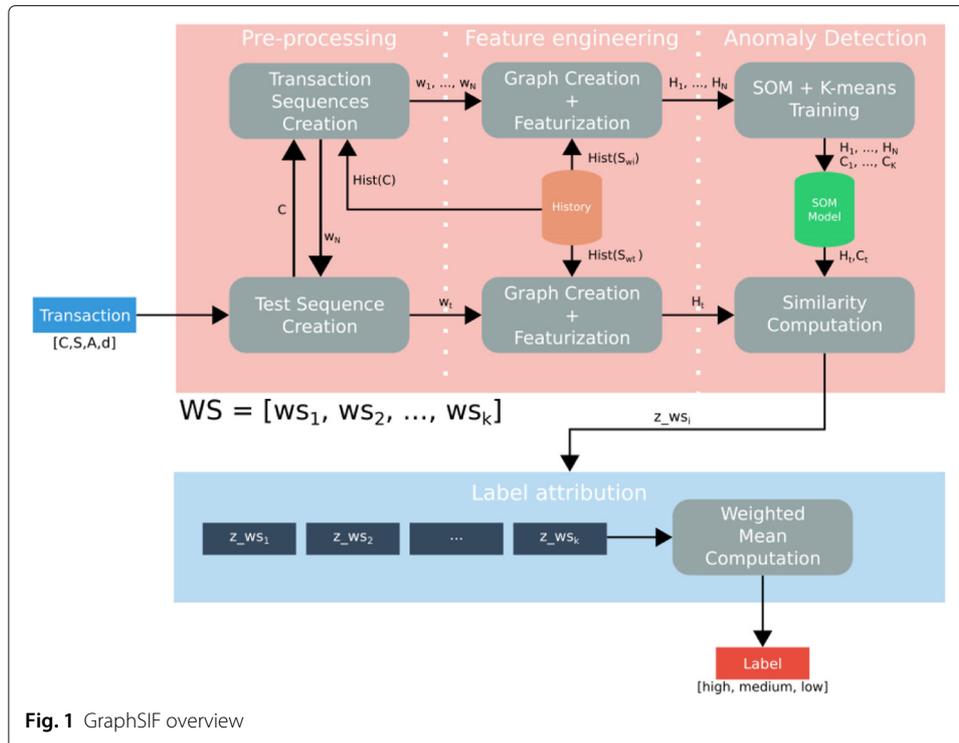


Figure 1 shows an overview of the process. A transaction  $t$  involving the client  $C$ , the supplier  $S$  and the account  $A$  at a date  $d$  is given as input to GraphSIF. For a set of window sizes  $ws_1, ws_2, \dots, ws_k$ , the following process is repeated:

First, the identifier of  $C$  is used to gather all transactions involving  $C$  ( $\text{Hist}(C)$ ) from the History dataset (that contains the list of all the transactions occurring between all the companies of the B2B ecosystem). Then, this list of transactions is ordered by date of occurrence and split in  $N$  fixed-size windows of size  $ws_i$  where  $i \in 1, \dots, k$ . The sequence of transactions (dubbed “behavior sequence”) is then sent to the next phase. At the same time, the oldest transaction of the most recent window of the sequence ( $w_N$ ) is removed and  $t$  is added as its most recent transaction, creating the “test window”  $w_t$ . This step allows us to isolate the transaction relevant to the specific client and suppliers potentially victims of the supplier impersonation fraud.

In the next step, the transactions found in each window  $w_1, \dots, w_N$  and  $w_t$  are used to create a graph that represents the relationships between  $C$  and all of its suppliers. Each window is complemented by a set of transactions from the History dataset  $\text{Hist}(S_{w_i})$ , where  $S_{w_i}$  is the set of supplier involved with  $C$  during  $w_i$  found in the History Dataset. Similarly,  $\text{Hist}(S_{w_t})$  is used to create the test graph corresponding to  $w_t$ . The graphs are then transformed into a histogram that uses relationships between the accounts paid by  $C$  and used by its supplier as characterizing features of the graphs. Each of the graphs is converted into its corresponding histogram  $H_i$ , thus creating the sequence  $H_1, \dots, H_N$  and  $H_t$  the histogram corresponding to the test graph. This step allows us to transform a sequence of transaction into a feature vector representing the relationship between a client and its suppliers, taking into account the historical behavior of the client.

These histograms are then used to assert the similarity of  $H_t$  with the histograms of the behavior sequence  $H_1, \dots, H_N$ . First, the histograms are clustered using the clustering algorithm K-Means Jain (2010), and then are used to train a Self-Organizing Map (SOM) (Brockett et al. (2006)). The  $k$  centroids  $C_1, \dots, C_K$  are also located on the SOM. Then,  $H_t$  is assigned a cluster using the previously trained K-Means algorithm, and its similarity with the other members of the cluster is computed using the z-score metric. This anomaly detection step allows the system to distinguish usual relationship and unusual ones, that are more likely to be fraud attempts.

Once all the z-scores corresponding to the set of window sizes  $ws_1, \dots, ws_k$  is computed, a threshold function is applied and a weighted mean is used to aggregate the results into a label indicating the legitimacy of the transaction. This step is needed to consider the different granularity of each windows, and to produce a label that synthesizes all the knowledge provided by the previous analysis.

In the remainder of the paper, we first describe the different algorithms used at each phase of the system, and discuss the underlying motives behind their design. We then provide an experimental evaluation of the system using the labeled transactions found in the Audit dataset that contains the results of the expert system designed by SiS-id.

### Transactions pre-processing

This section details the first phase of the system, where the transactions from the History dataset are pre-processed in order to create local behavior profiles. The goal of this pre-processing is to partition the transactions emitted by a client  $C$  in order to detect repeatability in its payments.

#### Company local profile

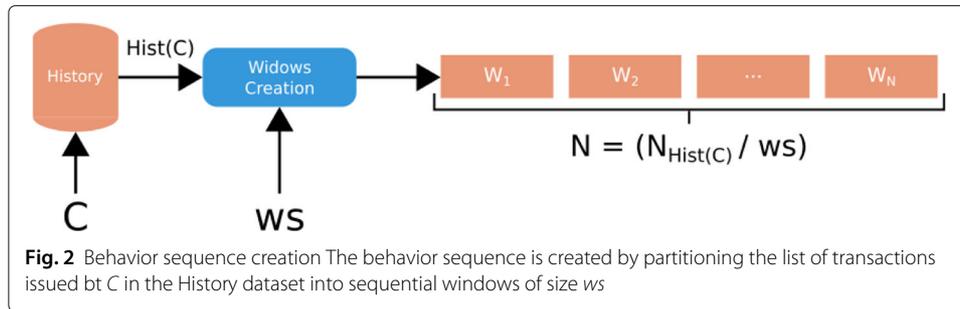
In this phase of the fraud detection system, all the transactions involving the client  $C$  involved in the tested transaction  $t = [C, A, S, d]$  are gathered from the History dataset. The History dataset contains all the  $N_H$  transactions made by all clients in the studied B2B environment. By isolating only the transactions issued by  $C$ , we create a local profile of  $C$ . As shown by Bolton et al. (2001), the use of local profile allows to detect anomalous behavior that would have been deemed legitimate when using a global profile. This local profile is dubbed  $\text{Hist}(C)$ .

#### Behavior sequence

As companies evolve and thus interact differently with suppliers or clients, the transactions they issue or receive change as time passes. In order to quantify this evolution, we first order all the transactions in  $\text{Hist}(C)$  temporally. This gives us an overview of  $C$ 's interaction with its supplier through time. Then, in order to characterize this behavior, we partition  $\text{Hist}(C)$  into sets of transactions of size  $ws$ , that we call "windows". Using fixed-length windows in order to describe the behavior of a system is a well-known technique, and has been successfully used in fraud detection systems such as Qian and Xin (2007) and Mongiovì et al. (2013).

#### Window creation

In order to create the windows, two kinds of partitioning are possible: by transaction date (from July 1st to August 1st for example), or by transaction rank (10th transaction to 5th



transaction, 5th transaction to 1st transaction...). A major issue with partitioning by date is that there is no guarantee that the transactions will be homogeneously divided into the different windows. In the most extreme case, all transactions might occur in a single window, and all the other windows are rendered useless for the system. Thus, creating windows by transaction order allows us to ensure that an equal number of transactions will be found in each windows. Figure 2 shows an overview of the windows creation process.

The number of windows  $N$  created from  $Hist(C)$  is inversely proportional with the size  $ws$  of the windows:  $N = \frac{N_H}{ws}$ . In the case when  $N_H$  is not a multiple of  $ws$ , the  $N_H \% ws$  oldest transactions in  $Hist(C)$  are discarded, where  $\%$  is the modulo operation. Indeed, the oldest transactions are the least likely to inform us of a fraud in the present.

The size  $ws$  of the windows has a major impact on the system. A small window size creates more data points for the system to analyze, at the cost of a decreased variability in the possible payment behavior, and thus a less detailed view of  $C$ 's behavior. Inversely, a larger window allows for more detailed view of the system, but less input will be provided to the system. Additionally, adding more transactions might add too much noise to the window and thus obfuscate meaningful patterns. Therefore, a careful trade-off has to be found for  $ws$ . We propose a way to solve this issue in "Label attribution" section.

**Test window**

In order to assert the legitimacy of a singular transaction, we use the previously partitioned sequence as a basis. The key hypothesis is that a legitimate transaction will not significantly disturb the payment behavior of  $C$ , while a fraudulent transaction will be different from the previously recorded behavior. The transaction to be tested is added to the most recent windows of the sequence, whose oldest transaction has been removed, thus simulating the occurrence of the new transaction as the next step in the sequence. Indeed, if only a single transaction was tested against the partitioned sequence, an anomaly would always be found due to the discrepancy in the number of transactions.

In the next section we detail how the relational data found in partitioned sequence (called "behavior sequence") and the test window is extracted and used to assert the legitimacy of the tested transaction.

**Graph-based feature engineering**

In this phase, each of the windows created in the pre-processing phase is transformed into a graph. This graph, called "transaction graph", allows the representation of the relationships between companies as a mathematical structure. The graph is composed of companies and accounts represented as vertices (also called "nodes"), while the flow of

money between companies creates the edges of the graph. However, in order to use the graphs derived from the behavior sequence as a basis for the anomaly detection system, they need to be converted into a structured data form (commonly referred to as “feature vector”) in order to be used. This type of transformation is known as “graph embedding” (Goyal and Ferrara (2018)). We propose a tailored graph embedding approach in order to express a transaction graph as a feature vector called “graph histogram”. This approach exploits several properties of the transaction graph in order to construct the embedding. The graph histograms corresponding to the behavior sequence are then used to train our anomaly detection system, while the graph histogram corresponding to the test windows is investigated.

The role of graph theory in GraphSIF is to transform the four attributes found in the History dataset into a set of graph-theoretic features allowing to perform fraud detection by taking into account the underlying relationships between companies and bank accounts found in the dataset. In the original form of the History dataset, the data points cannot be used directly to construct meaningful models, as their feature are categorical variables. The use of graph theory allows GraphSIF to express the links between each client and their suppliers through the bank account that they share, and to transform the list of categorical variables into a set of embedded graphs. Without this step the model could not be computed and thus fraud detection could only be performed on the reduced subset of original features from the History dataset.

### Transaction graph creation

In this subsection, we describe the process of creating a transaction graph from a set  $T$  of transactions (as a reminder, a transaction  $t$  has the following structure:  $t = [C_t, A_t, S_t, d_t]$  where  $C_t$  is the client issuing the transaction,  $A_t$  is the account receiving money from  $C_t$ ,  $S_t$  is the supplier receiving the transaction, and  $d_t$  the date when the transaction takes place). A graph  $G = \langle V, E \rangle$  is composed of two sets: a set of vertices  $V$  that represents the entity of the targeted system, and a set  $E$  of edges that represents the relationship of the entities, and  $e \in E = \langle n_1, n_2 \rangle$  with  $n_1, n_2 \in V^2$ .

Algorithm 1 shows the process that creates a transaction graph  $G_{C,T}$  from a set of transaction  $T$ . An example of output of Algorithm 1 is given in Fig. 3. The algorithm takes as input a client  $C$  and parses  $T$  in order to map all of the accounts and suppliers involved in a transaction with  $C$  as vertices. For each transaction, two edges are created: one that links  $C$  and the account involved in the transaction, and one that links the account with the supplier receiving payment for the transaction. If a vertex representing an account or a supplier is already in the vertices set, it is not duplicated. Similarly, since edges already in the edge set are not duplicated, an occurrence metric is updated in both cases in order to prevent the loss of information.

If the algorithm stops at this point, only the payment information related to  $C$  is used. This means that if an account is used to pay a supplier  $S$  by a client different than  $C$ , it will not appear on the graph. In order to add the information provided by other clients, the accounts they use to pay  $S$  are appended to the graph, along with edges that link them to  $S$ . This addition allows us to make use of the collaborative knowledge of the other clients.

If the amount of payment was available, a possible use for the feature would be to assign weight to the different edges instead of using a simple binary weight. While having no impact on the graph structure, this might indicate the accounts privileged by a client company.

---

**Algorithm 1:** Transaction Graph Creation

---

**Data:**

- $C$ : identifier of Client company
- $T$ : Set of transactions

**Result:**

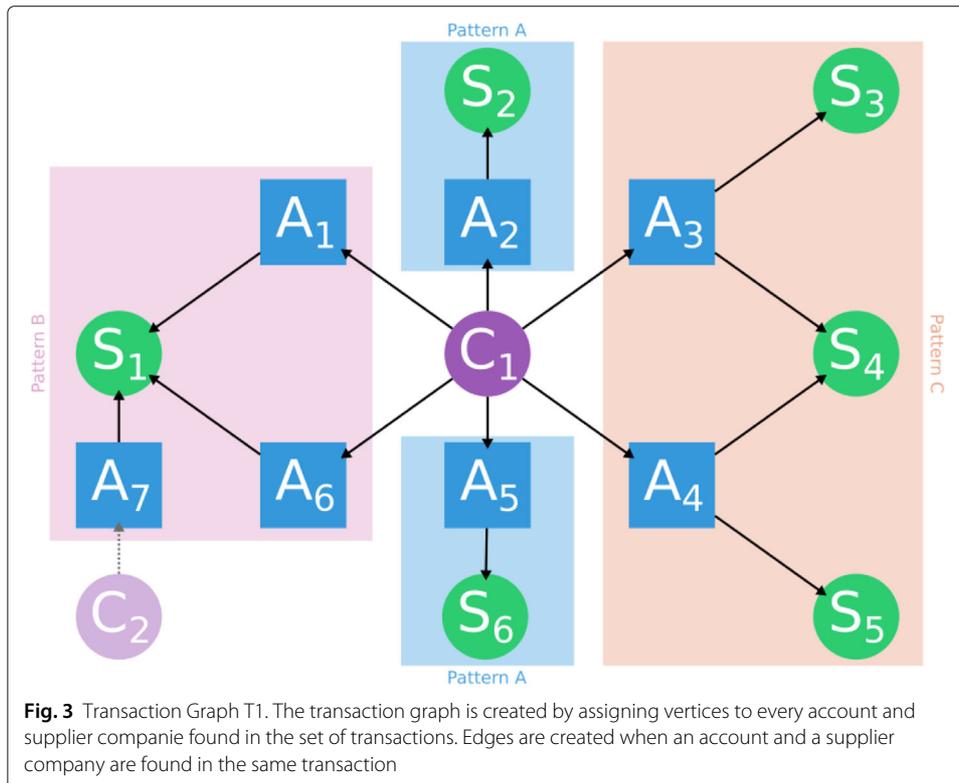
- $V$ : Set of vertices of  $G_{C,T}$
- $E$ : Set of edges of  $G_{C,T}$

```

1  $V_c = C, V_s = [], V_a = [], E = [];$ 
2 foreach  $t = [C_t, A_t, S_t, d_t]$  in  $T$  do
3   if  $C_t = C$  then
4      $V_a.insert(A_t)$ 
5      $V_s.insert(S_t);$ 
6      $E.insert((C, A_t));$ 
7      $E.insert((A_t, S_t));$ 
8      $T.remove(t);$ 
9 foreach  $r = [C_r, A_r, S_r, d_r]$  in  $T$  do
10  if  $S_r$  in  $V_s$  then
11     $V_a.insert(A_r);$ 
12     $E.insert((A_r, S_r));$ 
13  $N = V_c \cup V_s \cup V_a$ 

```

---



**Fig. 3** Transaction Graph T1. The transaction graph is created by assigning vertices to every account and supplier company found in the set of transactions. Edges are created when an account and a supplier company are found in the same transaction

**Properties of a transaction graph**

A transaction graph  $G_{C,T}$  shows interesting properties. It is a directed graph, as the edges represent the movement of funds from a client to a supplier through an account. A transaction graph is also a *bipartite graph*. A bipartite graph is defined in Baesens et al. (2015) as a graph whose vertices can be divided into two disjoint and independent sets  $u$  and  $v$  and such that every edge connects a vertex in  $u$  to one in  $v$ . The transaction graph satisfies this property as the created edges are only from company to account and from account to company (no account-to-account or company-to-company edges exist in the graph). Table 2 shows the representation of the transaction graph T1 (shown in Fig. 3) as a connectivity matrix: each of the row corresponds to a company vertex, while a column represents an account vertex. The value in the row indicate the number of times a transaction has been issued involving the specified company and account.

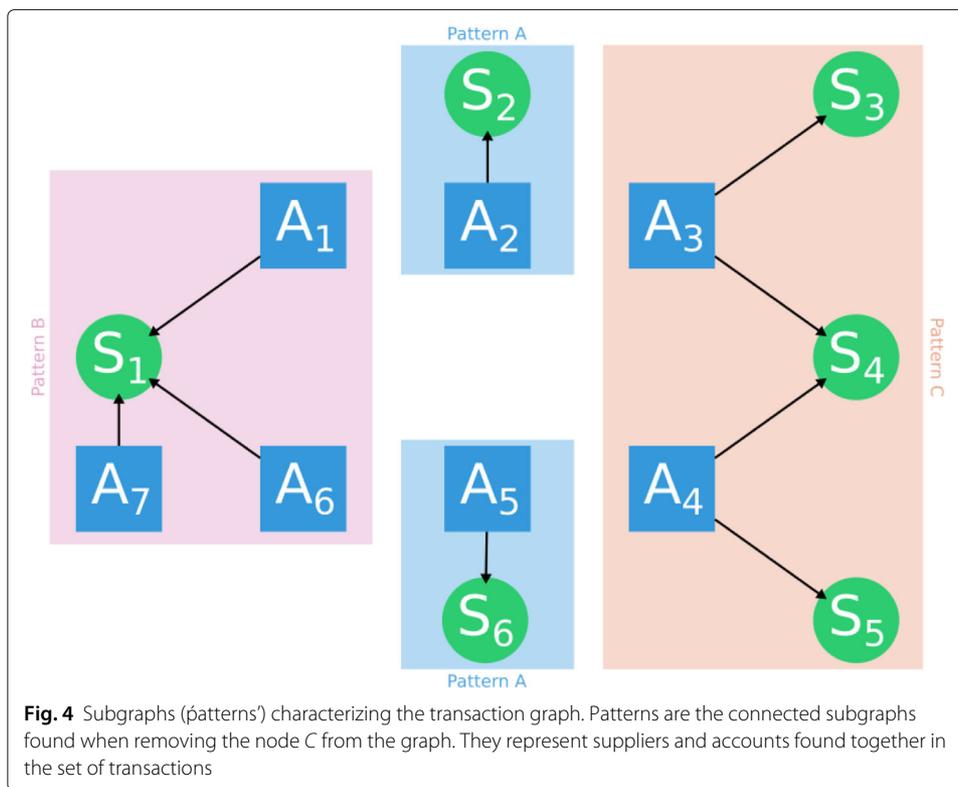
From the connectivity matrix, it is apparent that the sole vertex representing the client company (C1 in the example) plays a central role in the transaction graph. Centrality is an important metric in graph as it informs how a vertex can influence its neighbors. More specifically, the *graph theoretic center* is defined in Baesens et al. (2015) as the vertex with the smallest maximum distance to all other vertices in the network. This vertex is always the company vertex  $C$  in the case of a transaction graph  $G_{C,T}$ . We use this property to create payment patterns in order to characterize transaction graphs.

**Payment patterns**

In this subsection, we describe how we use the specific properties of a transaction graph in order to create a set of features capturing the transaction relationship between a client and the accounts used to pay its suppliers. Our approach is akin to the one developed by Akoglu et al. (2010) where a similar featuring process is used to characterize ego-network of specific nodes in the graph. In order to create the features, we first remove the client company’s vertex from the graph (C1 in Fig. 3), thus creating a set of  $D$  of disconnected sub-graphs composed of account vertices linked to supplier vertices. Among these  $D$  sub-graphs (that we dubbed “payment patterns”), if we only take into account the type of the node (“Supplier” or “Account”) and not its label (“S<sub>1</sub>” or “A<sub>4</sub>”), then it might occur that some of these sub-graphs are isomorphic, meaning that they share the exact same structure Baesens et al. (2015). It is the case for example in Fig. 4 where the sub-graph composed by (S<sub>2</sub>, A<sub>2</sub>) and (S<sub>6</sub>, A<sub>5</sub>) are isomorphic.

**Table 2** Connectivity Matrix of Transaction Graph T1. Each column represents an account vertex. Each row represents a company vertex. Numbers are the number of time a transaction created an edge between the two vertices

	A1	A2	A3	A4	A5	A6	A7
<b>C1</b>	1	1	1	1	1	1	0
<b>S1</b>	1	0	0	0	0	1	1
<b>S2</b>	0	1	0	0	0	0	0
<b>S3</b>	0	0	1	0	0	0	0
<b>S4</b>	0	0	1	1	0	0	0
<b>S5</b>	0	0	0	1	0	0	0
<b>S6</b>	0	0	0	0	1	0	0



This set of features can also be translated in the connectivity matrix shown in Table 2. Removing the central node means ignoring the first row of the matrix. A connected sub-graph can be connected in two ways: when a supplier is connected to a specific number of accounts (which is the case for  $S_1$ ), or when an account is connected to a specific number of account (such as  $A_3$ ). The case of Pattern C is a special one where the pattern satisfies both of these conditions.

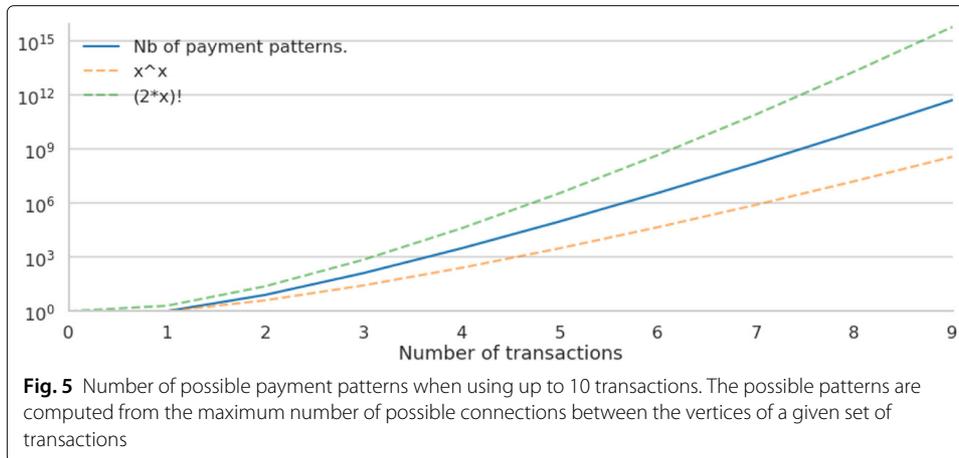
Functionally speaking, these connected sub-graphs indicate how the client interacts with its suppliers, and thus shows a “map” of the client’s activity in the set of  $T$  transactions used to create the transaction graph.

In Fig. 5, we calculate the possible number of unique payment patterns that can be created for a specific number of transactions based on Algorithm 1. This number seems to grow at an exponential speed, meaning that for  $x$  transactions used to build the transaction graph,  $x^x$  possible unique payment patterns can be found. This number corresponds to the number of features of an histogram. This fast growth in the number of features indicates that our data point might be placed in a very sparse high-dimensional space. Thus our system might fall prey to the curse of dimensionality Trunk (1979).

If the amount of the transactions were available, it could be used as a way to discriminate identical patterns of transactions by computing the cumulative amount found in a pattern and adding it as a feature for the anomaly detection model.

**Feature set creation**

In order to create an overview of a client’s behavior through time, we first use the transactions found in each  $N$  windows created in the feature engineering process to create the



$N$  corresponding graphs centered on the client  $C$ . A “test graph” is also created for the test window. We then transform the graph into a histogram with the technique previously described, thus creating  $N$  histograms where the features are the unique connected sub-graphs (i.e unique payment patterns) and the values are the number of occurrences of the pattern in the graph. Table 3 shows an example of such a process with T1 representing the graph shown in Fig. 3 and T2 and T3 representing other graphs. Similarly, the histogram corresponding to the test graph is also created using the same process. These histograms are then used as the basic features of our anomaly detection system.

Figure 6 shows an overview of the feature engineering process, proposed as a reminder. First, the transactions’ windows created in the pre-processing phase are converted into a transaction graph representing the relationships between a client and the accounts it uses to pay its supplier. Then the transaction graphs are in turn transformed into a set of feature vectors composed of the number of sub-graphs found in the transaction graphs.

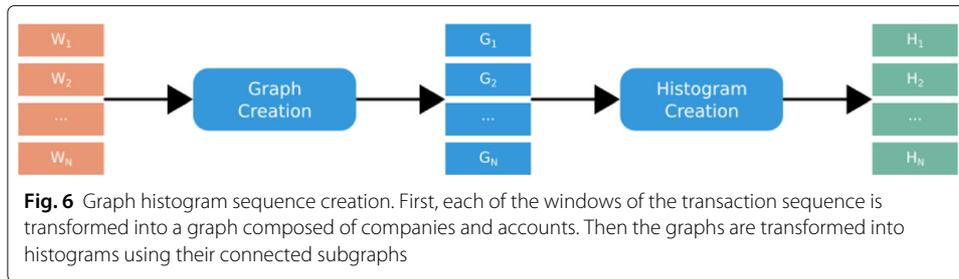
### Anomaly detection

In this section, we describe the anomaly detection system we use in GraphSIF in order to assert if the test graph created by the tested transaction and the most recent transactions of  $C$  is similar to the graphs found in  $C$ ’s behavior sequence created from  $C$ ’s historical transactions.

The anomaly detection system relies on two main building blocks: a parametric clustering algorithm (K-means Jain (2010)) that create clusters of transaction graphs represented as histograms according to their similarity, and a single-layer neural network called a Self-Organizing Map Bullinaria (2004) that projects multi-dimensional features such as the histograms into a two-dimensional space, in order to facilitate the computation of

**Table 3** Examples of featurized sets of transactions

Transaction set ID	Pattern A	Pattern B	Pattern C	Pattern D
T1	2	1	1	0
T2	3	3	0	1
T3	1	0	0	0

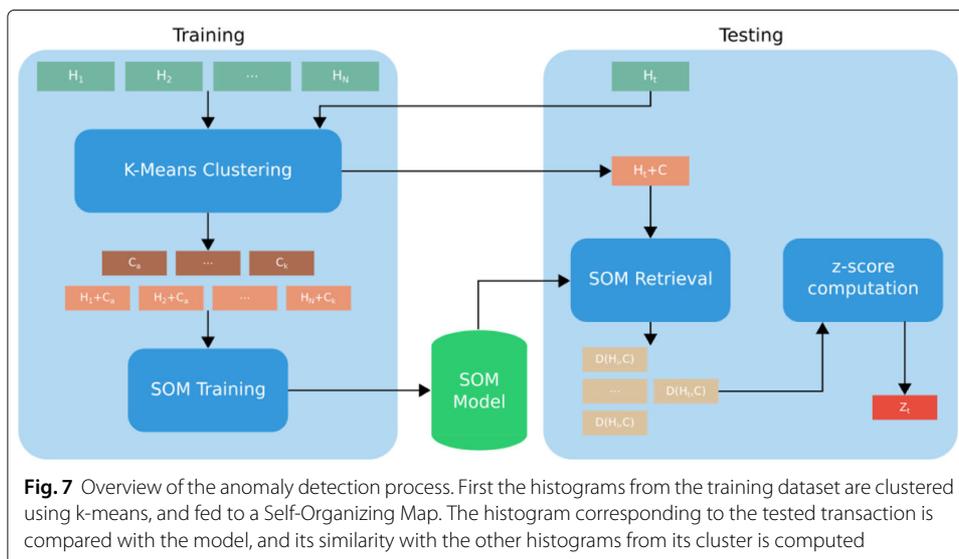


distance between feature vectors and to alleviate the curse of dimensionality (Trunk 1979) that states that the more dimensions, the more difficult it becomes to compute a meaningful distance between two feature vectors.

**Overview**

Figure 7 shows an overview of the anomaly detection process. First, the  $N$  histograms created at the end of the feature engineering phase are regrouped into clusters thanks to the K-means algorithm. Each of the histograms are associated with their clusters, and the centroid of each cluster  $C_i$  with  $i \in K$  is also computed.  $K$  is the number of clusters set as the parameter for the K-means algorithm.

Then, the histograms are used to train a Self-Organizing Map (SOM). In order to do so, each of the histogram is fed to the SOM, where a unique neuron (also called “node”) is activated. The weights of this node and the eight neighboring ones are then updated in order to match the values of the histogram. The operation is repeated for each of the histograms until every one of them is associated with a node. Several histograms can be associated with the same node. Finally, once the SOM is trained, the centroid of each cluster is fed to the SOM and the node activated by it is retrieved. This creates the “SOM model” that is composed of the nodes trained with the histograms along with the nodes corresponding to the centroids.



Once the SOM model created, when an histogram corresponding to a test graph needs to be evaluated, it first goes through the clustering phase undergone by the other histograms in order to be assigned a cluster  $c$ . Then, it is fed to the SOM in order to find the node activated by it. The distance between this node  $n_t$  and the node activated by the centroid of the cluster  $c$  ( $n_c$ ) in the SOM ( $D_t$ ) is retrieved, along with all the distance between the nodes activated by the members of  $c$  and  $n_c$ . All of these distances are then used to compute the z-score Abdi (2007) of  $D_t$ . The z-score is a statistical measure that tells us how many variation away a data point is from the mean. The higher the z-score, the less similar an histogram is from the others.

In the remainder of this section we detail the different algorithms used to obtain the z-score from our input data.

### Training

In this subsection, we detail the training of the two models (the K-Means algorithm and Self-Organizing Map) used in the anomaly detection system. Training the models means that we use the historical histograms created at the end of the graph-based feature engineering phase to fit the models so that they accurately represent the past behavior of the considered client. The training phase is divided in three parts: the histogram clustering where the histograms are assigned a cluster, the SOM training where the weights of the nodes of the SOM are adjusted to match the value of the histograms, and finally the histograms projection where the SOM nodes corresponding to the histograms and centroids are determined in order to be used in the testing phase.

#### *Histograms clustering*

Clustering the histograms is the first step of the training process. It consists of using the K-Means Jain (2010) algorithm on the set of historical transactions in order to assign them a cluster based on their similarity according to a selected distance metric. K-Means is a well-known clustering algorithm that assigns clusters to feature vectors according to their proximity to a centroid that is the mean of the member of the clusters when the algorithm reaches convergence. This proximity is computed according to a distance metric such as the Manhattan distance or the Euclidian distance. We use the Euclidian distance in our current implementation.

Algorithm 2 shows an overview of the algorithm.

When using K-means, it is very important to carefully choose the number of clusters  $K$  so that it represents accurately the underlying distribution of the data. Due to time constraints a thorough analysis of the optimal number of parameters could not be performed. However, a preliminary experimental study conducted on a selected test company showed that  $K = 3$  seems to yield the best results in terms of stability of the cluster.

The motivation behind the use of a clustering algorithm as a first step in our training process is to partition the local profile of a user in order to detect re-occurring transactions graphs. As a graph represents the interaction of a client with its supplier, it is logical to think that more than one type of transaction graph might occur in the lifetime of the client company. For example, assuming that the transactions occur homogeneously every month, two suppliers might be paid every sixth month using a specific account each, while three other suppliers might use only one account to be paid every two months. These two examples will create different transaction graphs throughout the behavior sequence, all

---

**Algorithm 2:** Histograms clustering using K-Means

---

**Data:**

- $cH = (h_1, \dots, h_N)$ : Set of histograms (observations)
- $C^{(1)} = (C_1^{(1)}, \dots, C_k^{(1)})$ : Set of centroids at time 1

**Result:**

- $S_i^{(t)}$ : histograms for each clusters.
- $C^{(t)} = (C_1^{(t)}, \dots, C_k^{(t)})$ : Set of centroids at time  $t$

```

1 Init ( $C^{(1)}$ );
2 while Not convergence do
3    $S_i^{(t)} = \left\{ h_p : \|h_p - C_i^{(t)}\|^2 \leq \|h_p - C_j^{(t)}\|^2 \forall j, 1 \leq j \leq k \right\};$ 
   // assign the histograms to the closest centroid.
4    $C_i^{(t+1)} = \frac{1}{S_i^{(t)}} \sum_{h_j \in S_i^{(t)}} h_j;$ 
   // calculate the new position of the centroid as the mean
   of the observations in the new cluster.
5    $t = t + 1$ 

```

---

sharing similar properties. These transaction graphs will be assigned different clusters by the clustering algorithm, thus identifying two different component of the client's behavior. Thus, clustering the graphs into different clusters allows us to further decompose this behavior.

This research could be furthered by studying the different clusters found and determining if they relate to a real behavior for the client company (such as the acquisition of materials, taxes payments, and so on). However, in the context of fraud detection, we solely focus on the fact that the clusters can be used as a point of comparison for new transactions graphs.

Alternative clustering algorithms such as the k-medoids algorithm Park and Jun (2009) or x-means algorithm Tsunenori (2000) might be investigated in order to optimize the clustering process.

**Self-Organizing map training**

One of the issue with using the transaction patterns described in the feature engineering phase as the dimension for our feature vectors is that we do not have direct control on the dimension of said feature vectors. Furthermore, the number of possible patterns grows exponentially with the number of transactions that are found in the set used to create the underlying transaction graph. This fact leads to two major issues: firstly, the feature vectors representing the algorithm might be projected in a high-dimensional but sparse feature space, thus resulting in artificially high distance due to the curse of dimensionality. Secondly, it is hard for a human to interpret the notion of proximity in such a high-dimensional space. In order to alleviate these two issues, we use a Self-Organizing Map, and more particularly a Kohonen network. Kohonen and Self-learning musical gram (1989); Brockett et al. (2006); Bullinaria (2004). This type of network aims to organize all of the feature vectors in a two-dimensional plane according to their similarity.

A SOM is an unsupervised neural network based on competitive learning, in the form of a neural network where only one neuron (also called node) is activated at any one time. The specificity of the Kohonen network is that the single computational layer is arranged in rows and columns, and each node is fully connected to all the source nodes in the input layer. In order to train the SOM, after an initialization phase, three steps are performed until convergence: sampling, matching and updating.

In the initialization phase, each of the neurons of the computational layer of the SOM is assigned random activation weights. The values of the weights need to be reasonably close so that every neuron has a chance to be activated. The number of activation weights of a neuron is the same as the dimension of the feature space.

In the sampling phase, a sample  $x$  is drawn from the set of feature vectors  $X$ . This sample is randomly chosen so that the ordering of the set does not have any impact on the training process.

In the matching phase, the winning neuron  $I(x)$  is found by comparing the weight vectors of each neuron and finding the one closest to the values of the input vector. More specifically, the similarity of the vector to a neuron  $j$ 's weights is computed using a discriminant function  $d_j(x) = \sum_{i=1}^D (x_i - w_{ji})^2$ . Closely related input vectors might activate the same winning neuron  $I(x)$ .

In the updating phase, the winning neuron and its neighbors are updated using the equation

$$\Delta w_{j,i} = \eta(t) T_{j,I(x)} (x_i - w_{ji}). \quad (1)$$

In this equation,  $T_{j,I(x)}$  symbolizes the topological neighborhood of the winning neurons, and is defined as  $T_{j,I(x)} = e^{-\frac{s_{ji}^2(x)}{2\sigma^2}}$  where  $\sigma$  is the size of the neighborhood of the winning neuron.  $\eta(t)$  is the learning rate that dictates how the weights of the winning neuron and its neighbors gets to be updated to a value closest to the input vector. This learning rate is defined as

$$\eta(t) = \eta_0 e^{-\frac{t}{\tau}} \quad (2)$$

where  $\eta_0$  is the initial learning rate and  $\tau$  a parameter for the exponential decay function that decrease the learning rate over time.

The sampling phase, matching phase, and updating phase are repeated until the SOM reaches convergence, meaning that no significant modification in the weights occurs.

In our setting, the set of feature vectors  $X$  corresponds to the set of histograms  $H$  created by the feature engineering process.

### **Histograms projection**

Once the SOM is training and convergence is reached, an additional step is performed: each histogram  $h$  found in the set of created histograms  $H$  is fed to the SOM, and the neurons  $n(h)$  activated by the histogram is associated to it. Similarly, each centroid  $c$  of  $k$  centroids created by the clustering phase are also fed to the SOM and the neurons  $n(c)$  are associated with the centroids.

This phase allows us to project the histograms from their high-dimensional feature space to the 2-dimensional space of the SOM nodes, in a way that preserves their similarity. A valuable effect of this projection is that it enables a human expert to read and

interpret the created map, and thus allows us to use human knowledge to understand the transactions patterns uncovered.

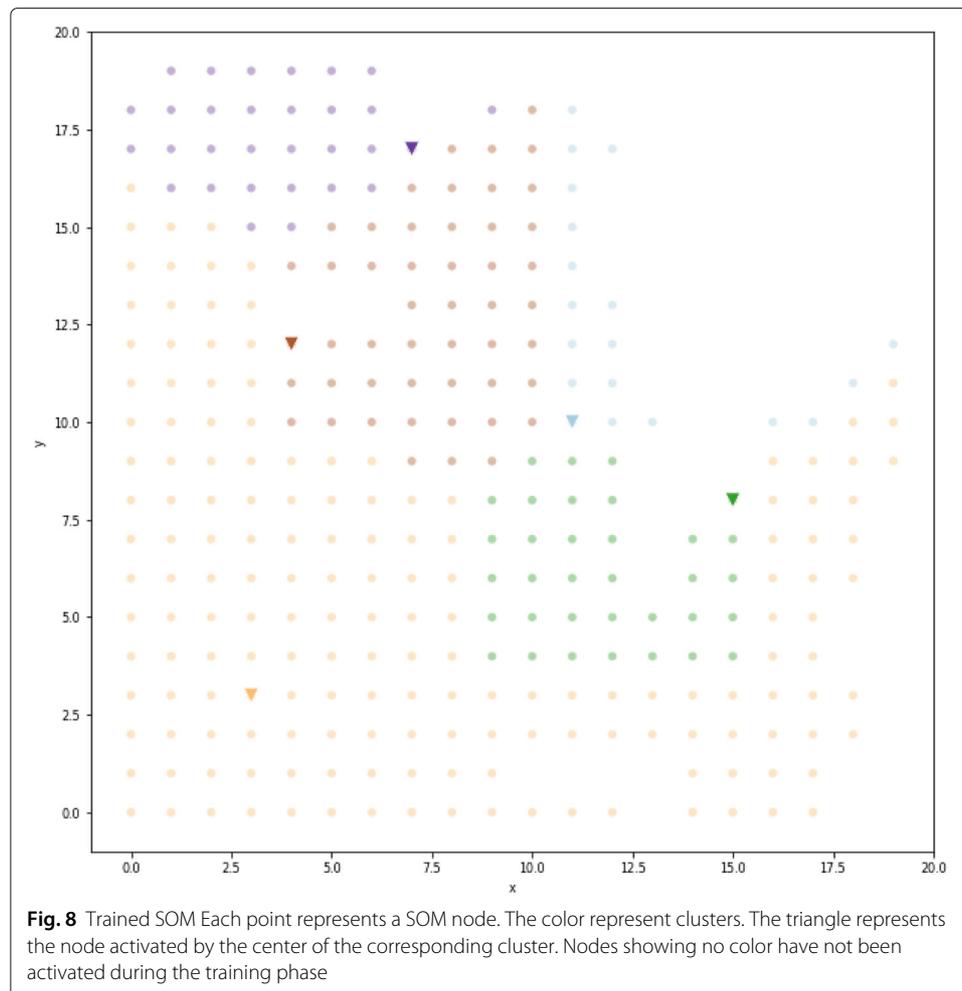
**Testing**

In the testing phase, a transaction is given to the anomaly detection system in order to assert its similarity with *C*'s historical transactions. Before being submitted to the anomaly detection system, the test transaction is integrated to the most recent transactions of *C* and turned into a transaction histogram following the steps of the feature engineering process previously described. The result of this phase is a legitimacy score summing up how distant the transaction is from the historical ones. This phase is divided in 3 steps: the test histogram clustering, then the SOM distance retrieval, and finally the similarity computation.

**Test histogram clustering**

In this step, the test histogram  $h_t$  is assigned a cluster based on the centroids found in the training phase. It is assigned the cluster whose Euclidian distance is the closest, following the equation

$$c_{h_t} = \min(c_i : \{ \|h_t - C_i\|^2 \leq \|h_t - C_j\|^2 \forall j, 1 \leq j \leq k \} \quad (3)$$



Assigning a cluster to  $h_t$  allows us to compare it to the historical transactions closest to it. Furthermore, as the centroids determined by K-means algorithm represent the mean of all the historical transactions of the cluster, it represents the representative member of the cluster. Thus, the farthest  $h_t$  is from the centroid, the less similar to the other member of the cluster it is. In other words, the further away  $h_t$  is from the centroid, then the closer from the edge of the cluster it is, and thus is dissimilar to the other members of the cluster. However, as mentioned previously, the curse of dimensionality might hinder the computation of a meaningful distance in our case. Thus, we use the trained SOM in order to compute the similarity of  $h_t$  with the member of its cluster. Figure 8 shows a trained SOM with the each node colored with its corresponding cluster, along with the nodes activated by the cluster center. Figure 9 shows an overview of the labeling process.

#### **Self-organizing map distance retrieval**

In this phase, we feed  $h_t$  to the previously trained SOM and thus retrieve  $n_{h_t}$  the neuron activated by  $h_t$ , and we compute  $D(n_{h_t}, n_{c_{h_t}})$  the Euclidian distance between the neuron activated by  $h_t$  and the neuron activated by  $c_{h_t}$  that is the centroid of the cluster assigned to  $h_t$ .

Once this distance is acquired, it might be tempting to use it directly as a way to determine  $h_t$ 's legitimacy score, by for example assigning a threshold distance from which  $h_t$  would be considered anomalous. However, assigning a value to the threshold might prove a challenge as the distance corresponds to a neuron-to-neuron distance and not a vector-to-vector distance. Thus, it is not clear what the meaning of such a threshold would be. We propose a solution to this issue in "Label attribution" section.

#### **Similarity computation**

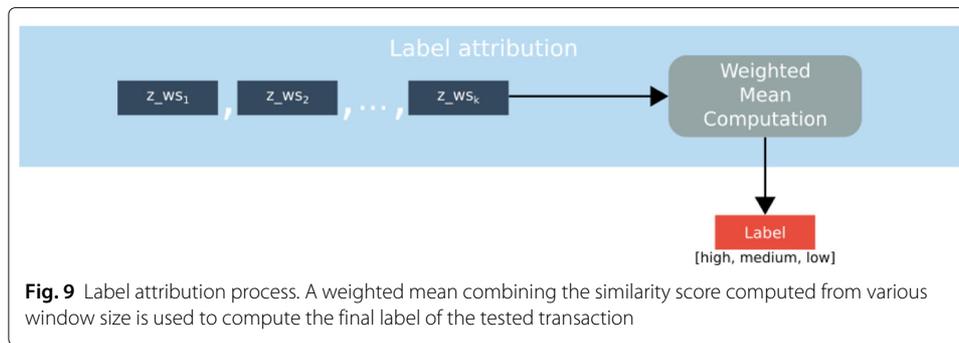
In order to provide a more robust way to assert  $h_t$ 's similarity, the distances from the other histogram members of the cluster  $C_{h_t}$  and its centroid ( $\{D(n(h_i), n(c_{h_t})) \forall h_i \in S_{h_t}\} = \mathbf{D}$ ) are also retrieved. Once retrieved, the z-score of  $D(n(h_t), n(c_{h_t}))$  with respect to  $\mathbf{D}$  is computed. The z-score, as described in Abdi (2007), is a statistical metric that corresponds to how many standard deviation away a data point is from the mean of an ensemble. In our case, it gives us insight about how far away  $h_t$  is from the centroid, with respect to all the other members of the cluster. The higher the z-score is, the more dissimilar  $h_t$  is from the other members of the clusters, and thus the more likely it is to be an anomaly.

#### **Label attribution**

The previously described training and testing algorithm uses a set of historical histograms to assert the similarity of a test histogram, through the computation of a z-score. However, the computation of these histograms is dependent on the size of the window of transaction  $ws$  that is used in the pre-processing algorithm. Indeed, the window size directly impacts the graph extracted from the window, and thus the histogram describing the graph.

#### **Impact of window size**

A small window size will create smaller graphs that encompass the short-time behavior of a client  $C$ . Furthermore, a small window size will also provide more behavior histograms to perform the clustering and SOM training, at the expense of a decrease in the



complexity of patterns found in a graph, as less transactions means less possible relationships between account and supplier. Lastly, anomaly detection using graphs created from small windows are less stable as adding a single transaction can create a huge topological difference between two graphs.

On the contrary, a large window size will create larger graphs, that sums up a large number of transaction and thus the long-term behavior of the client. However, as the number of transactions needed to create the graphs will be higher, less data points will be created. As a certain amount of data point (depending on the size of the SOM) is needed for the training algorithm, very high windows size are thus not suitable for the detection system. However, larger transaction graphs means that more complex patterns might be formed, which would have been missed if smaller windows were used. Lastly, using a large window size means that the topological modification following the introduction of a single transaction might not be enough for the anomaly detection system to pick up an anomaly.

### Optimizing windows size

A straightforward way to determine which windows size is more suitable for anomaly detection for a specific client  $C$  would be to perform an optimization method such as grid search Bergstra and Bengio (2012) or random search Bergstra and Bengio (2012). However, these optimization methods rely on the assumption that target labels are available, which is not the case in our use-case. Thus, an alternate method has to be found.

Instead of trying to optimize the size of the window, a possible solution would be to perform the anomaly detection in a range of different windows size, and aggregate the results in a way similar to bagging Dietterich (2000). This way, the anomaly detection system would be able to draw its conclusion from both small size transaction graphs and high size transactions graph when assigning the legitimacy score of a transaction.

As a way to perform this aggregation, the following algorithm is proposed. For every size  $ws$  in  $\mathcal{W}$  of length  $l(\mathcal{W})$ , the pre-processing phase, feature engineering phase and anomaly detection phase of the anomaly detection system are performed, effectively creating  $l(\mathcal{W})$  anomaly detection systems, and a z-score is computed for a transaction  $t$  from each of them. Then, the following process is applied:

- 1 The z-scores undergo a discretization process when the score is turned into one of the three legitimacy labels (“high”, “medium”, “low”). In order to do so, two risk thresholds ( $0 < r_1 < r_2 < 1$ ) are used as parameters for the threshold function. These risk thresholds represent the fraction of the maximal z-score corresponding

to each of the legitimacy labels. As a rule of thumb, a z-score of 3 indicates that a value is an outlier with respect to a given data set. Thus, a risk score  $r_1 = 0.2$  indicates that if the z-score of a given value is smaller than  $0.2 * 3 = 0.6$  then it is considered legitimate by the anomaly detection system.

The risk thresholds are defined by the investigation team, as it relies on the costs implied by a false positive (legitimate transaction mislabeled as fraud) or a false negative (fraud mislabeled as legitimate transaction). These costs depend on factors that reside outside of the scope of the anomaly detection system, and thus need to be asserted by a team of experts.

- 2 Each of the label is assigned a weight ( $w_h, w_m, w_l$ ) that represent a bonus in the overall legitimacy score. These weights can be parameterized by the investigator in order to control the sensitivity of the system. Usually,  $w_h > w_m > 0 > w_l$  so that "high" legitimacy labels pull the score up and "low" legitimacy label decrease the overall legitimacy score.
- 3 The sum of the  $l(\mathbf{W})$  weights is computed and normalized so that an overall legitimacy score  $0 < L_{\mathbf{W}} < 1$  is calculated.

$L_{\mathbf{W}}$  can also be discretized using a threshold function if the need to provide labels is found. This threshold function can use as input a list of threshold risks  $\delta_1, \delta_2, \dots, \delta_n$  corresponding to the operational needs of the fraud detection team. Alternatively, a voting system might be used in order to aggregate the value of each of the anomaly detection systems. This label attribution phase thus alleviates the need to search for an optimal value of  $w$ s and allows the anomaly detection to be performed on both short-term and long-term transaction behavior of the investigated client  $C$ .

If the amount of transactions were available, it could be used to create a cost function that would impact the thresholding parameters so that more attention would be given to high-value transactions.

## Experimental results

In this section, we discuss the experimental setting used to assert the performance of GraphSIF, and show the experimental results obtained by running GraphSIF on a set of transactions previously labeled by SiS-id expert system. While these labels provide a baseline for GraphSIF, the labels of the expert system does not represent the ground truth about fraud, as the "medium" legitimacy label represents a lack of knowledge from the expert system, while the "low" legitimacy label can be issued in both case of fraud or invalid transaction. Thus, our first goal in this experiment is to assert the consistency of GraphSIF with human knowledge, as the ground truth concerning fraud detection is unknown. The second goal of this evaluation is to compute the operational efficiency of GraphSIF.

This experimental evaluation is divided in four parts: first, the experimental settings are described. Then, a single client is analyzed in order to detail the consistency between GraphSIF and SiS-id expert system in a case study. Then the global performances of GraphSIF are discussed. Finally a discussion about the results is provided.

### Experimental settings

In order to produce these results, GraphSIF has been run on a laptop running Ubuntu 18.04.4 with an Intel Core i7-10510U CPU and a 16 GB memory. The data used to train

the model is the History dataset previously described in “[SiS-id fraud detection platform](#)” section, consisting in a set of unlabeled transactions between client companies and suppliers companies. The data used to test our model is the Audit dataset described in “[SiS-id fraud detection platform](#)” section.

In order to perform our experimentation, each of the client company found in the Audit dataset is selected. The corresponding subset of transaction from the History dataset is used to train GraphSIF, and then a sample of at most 1000 transactions from the Audit dataset is used to assert the consistency of the results with SiS-id’s expert system.

The different parameters used in the experiment are described in Table 4.

### Case study

In this section we evaluate the performance of GraphSIF on the subset of transaction issued by a single client  $C$ . In this evaluation, only the transaction issued by  $C$  in the Audit dataset are considered. This subset contains 1000 test transactions to evaluate the performances and 218,325 historical transactions for the History Dataset to train the model. First, we analyze the consistency of GraphSIF with the expert system, and then we present the operational efficiency of our system.

### Consistency

We first consider how close the results of GraphSIF are with the results of the expert system. While these results can not be considered ground truth, as the expert system does not distinguish between fraudulent and invalid transactions, they still provide a baseline for the analysis of GraphSIF results.

**High Legitimacy Label** First, we consider the results given by both the expert system and GraphSIF concerning the high legitimacy labels. Figure 10 shows a Venn diagram indicating the distribution of the high legitimacy label. The consistency of the two set is not high, only 19 out of the 167 transactions given a “high” legitimacy label by the expert systems are given the same label by GraphSIF. However, this inconsistency might be explained by the fact that the expert system relies on knowledge internal to the platform, in the form of a registration of secured suppliers, that is not available for the graph-based model.

**Table 4** Parameters used in the experience

Parameter	Value
Window size range	[2, 5, 8, 11, 14, 17, 20, 23]
K	3
Z-score thresholds	[0.2, 0.5]
Std max	3
Label weights	[20, 10, 0]
Risk thresholds	[0.8, 0.5]
<b>SOM Parameter</b>	<b>Value</b>
Map Size	10x10
Lattice	rectangular
Normalization	var
Initialization	PCA
Neighborhood	Gaussian
Training	batch

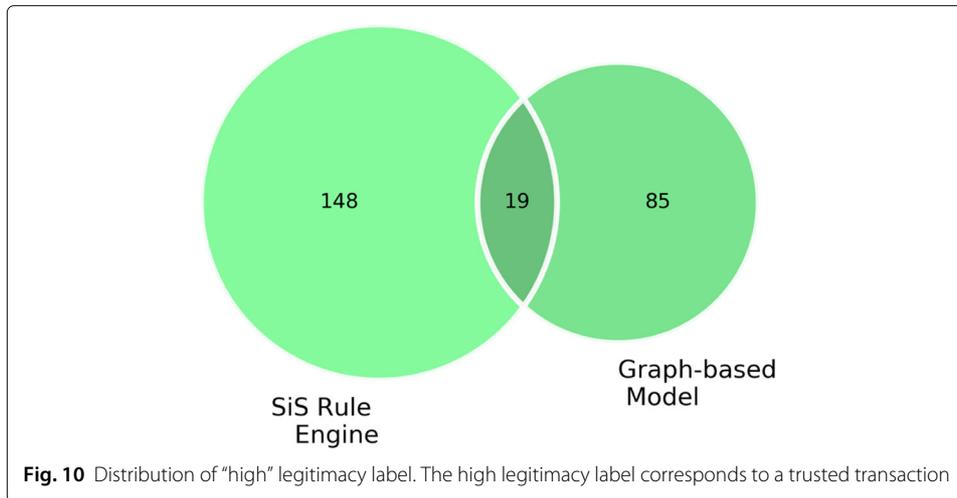


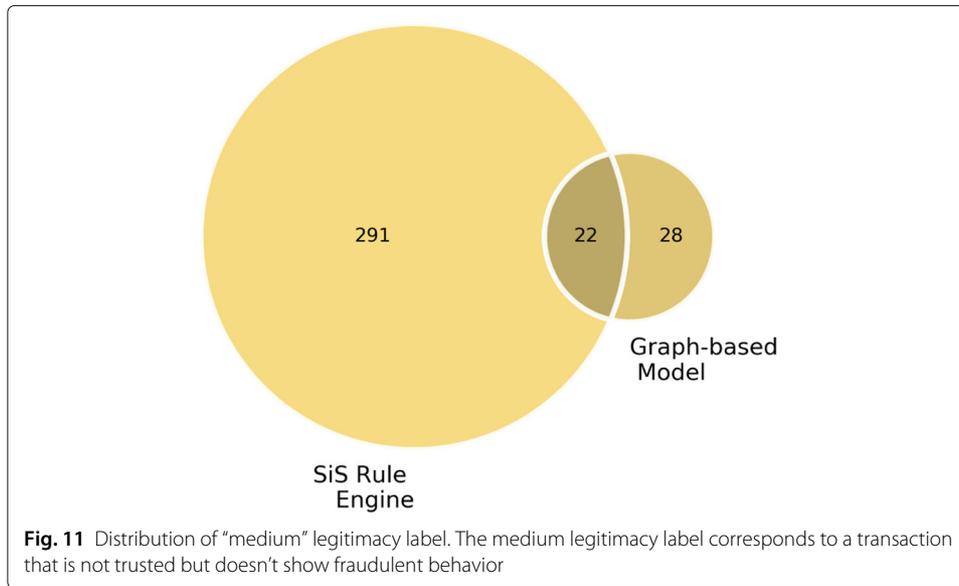
Table 5 shows the confusion matrix indicating the complete distribution of each label. Most of the high legitimacy labels (128 out of 167) have been labeled as low legitimacy transactions by GraphSIF. This behavior is consistent with the hypothesis of the expert system using additional knowledge to perform its classification.

**Medium Legitimacy Label** Then, we consider the results given by both the expert system and GraphSIF concerning the medium legitimacy labels. Figure 11 shows the distribution of the medium legitimacy label. For the expert system, a medium label indicates a lack of knowledge. Almost a third (313 out of 1000) of the transactions have been assigned this label by the expert system. On the contrary, a medium label doesn't indicate a lack of knowledge for GraphSIF, but rather informs the user that a specific transaction is slightly unusual. Thus the consistency between the two sets is not really expected. However, the low number of transactions labeled with the medium label by GraphSIF (50 out of 1000) seems to indicate a higher assertiveness of the proposed model. Table 5 shows that most (217 out of 291) of the medium labels given by the expert system were assigned a low legitimacy label by GraphSIF.

**Low Legitimacy Label** Finally, we consider the results given by both the expert system and GraphSIF concerning the low legitimacy labels. Figure 12 shows the distribution of the low legitimacy label. There is a high consistency between the expert system and GraphSIF concerning this set of suspicious transactions, as 501 out of 520 transactions were given the low legitimacy label by both of the detection systems. The 345 transaction assigned a low legitimacy transaction by GraphSIF and not the expert system come from the two other sets. Furthermore, the last row of Table 5 shows that only a handful of transactions labeled as low by the expert system have been given another label by

**Table 5** Confusion Matrix - SiS Rule Engine's and GraphSIF

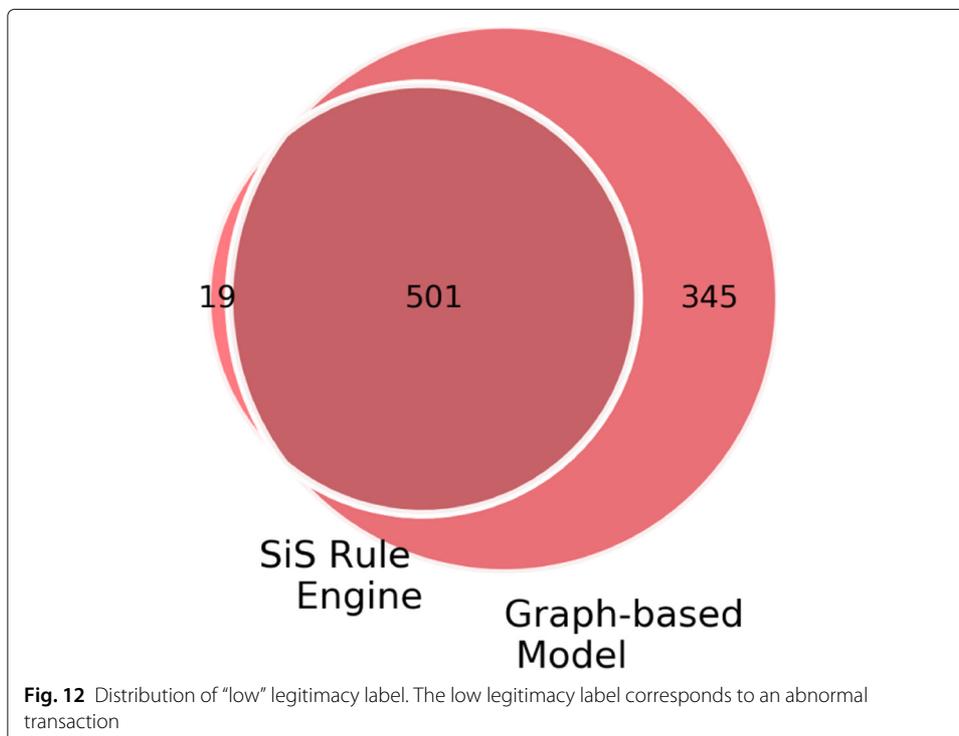
GraphSIF →			
Expert system ↓	High	Medium	Low
High	<b>19</b>	20	128
Medium	74	<b>22</b>	217
Low	11	8	<b>501</b>

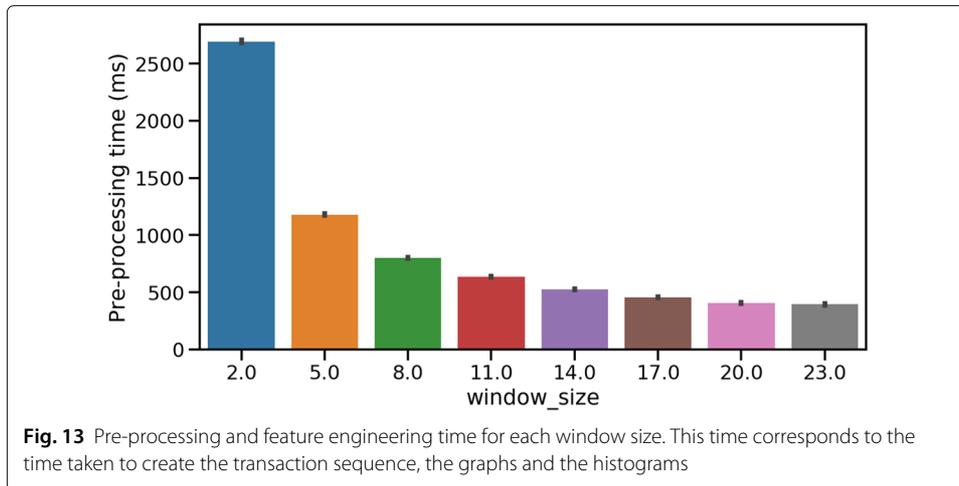


GraphSIF. It is possible that the 11 transactions given a low legitimacy label by the rule engine could use knowledge not available for GraphSIF, such as when a supplier closes an account.

**Operational efficiency**

In this section, we consider the operational efficiency of GraphSIF. GraphSIF, as opposed to the rule engine used by the expert system, requires a pre-processing phrase and a





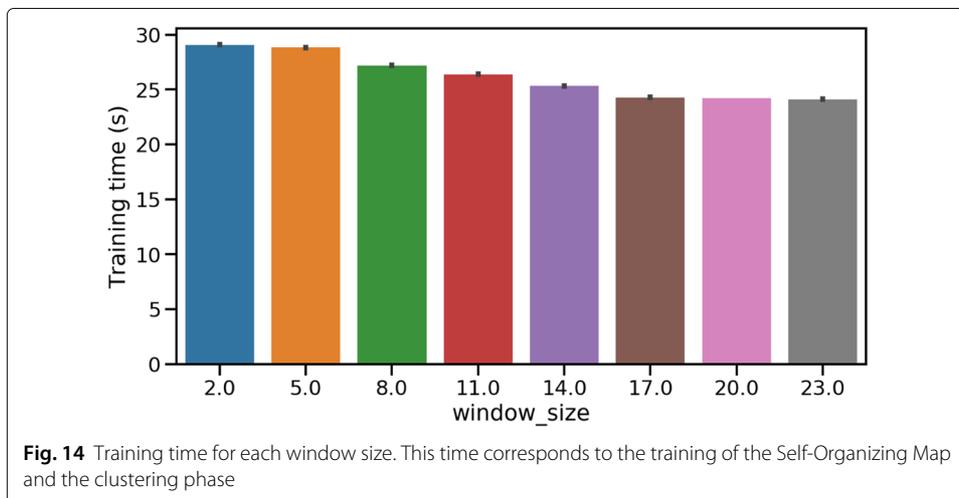
training phase to be completed in an off-line fashion. Then a transaction can be assigned a label.

**Pre-processing time** Figure 13 shows the time taken by GraphSIF to pre-process the transaction. The results are separated by window size, as this parameter has a huge impact on the quantity of windows created, and the amount of transactions in each window.

The pre-processing time seems to vary exponentially between 500 ms and 2500 ms depending on the window size. A shorter window size leads to higher pre-processing time, probably due to the fact that more windows are created.

In the case of the expert system, a pre-processing phase is also performed, requiring calls to internal and external APIs (Application Programming Interface). This pre-processing phase can take up to 3 seconds, which is significantly larger than GraphSIF pre-processing time.

**Training time** Once the pre-processing phase is complete, the training phase can begin. Figure 14 shows the time taken to perform the training phase. We can see that this time is also correlated with the window size, and varies from 28 seconds to 25 seconds as the



window size becomes larger. Two reasons might impact the decrease in training time: the number of windows used to train the SOM is reduced and thus less data is available for training, or, as the window grow larger, more distinctive patterns emerges and thus the training is more easily performed.

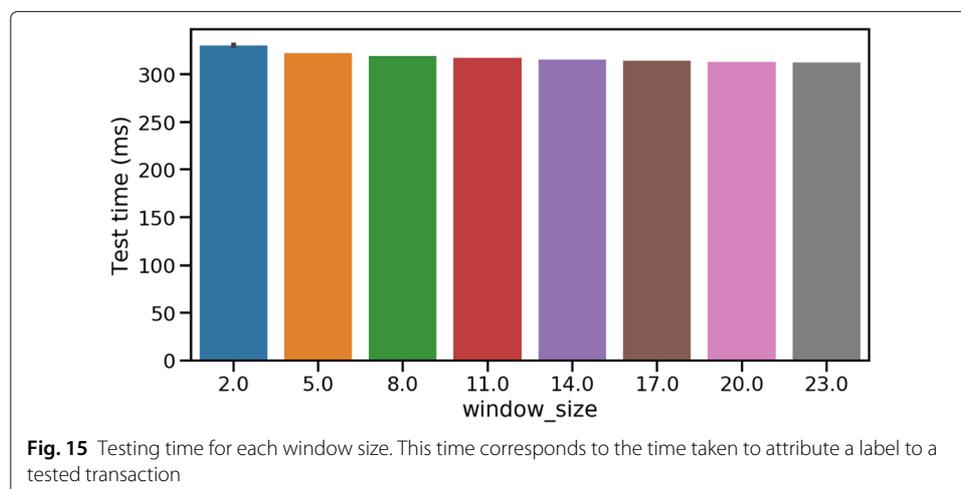
In the case of the expert system, there is no training phase, however the rule engine has to be updated in order to adapt to new fraud cases. This process is long and costly, as it relies heavily on human input. GraphSIF data-driven evolution is thus an improvement.

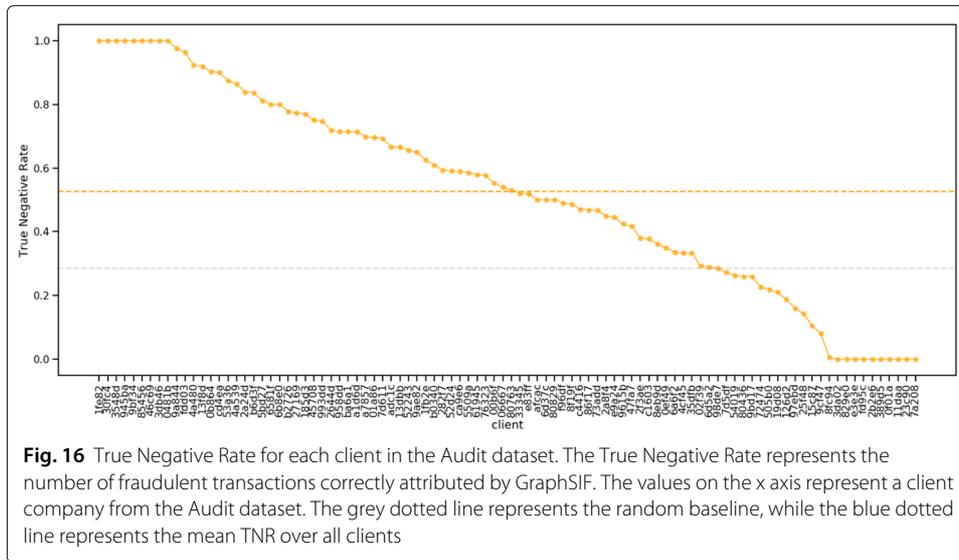
**Test time** Figure 15 shows the time taken to perform the classification of each transaction in the test set. While a slight increase is visible for a window size of 2, the test time is mostly uniform and close to 300 ms. The expert system classification time is closer to 3 seconds, as the pre-processing has to be performed for each tested transaction. Thus GraphSIF is faster than the expert system.

**Global results**

In this section we focus on GraphSIF’s behavior with the other clients of the Audit dataset, in terms of consistency and operational efficiency. The experiment presented in this section is conducted with every transaction of the Audit dataset. As opposed with the case study proposed in the previous subsection, this experiment analyses the behavior of each client company of the dataset in a separate manner, in order to assert the behavior of GraphSIF in various settings, with an emphasis on the number of potential frauds discovered by GraphSIF.

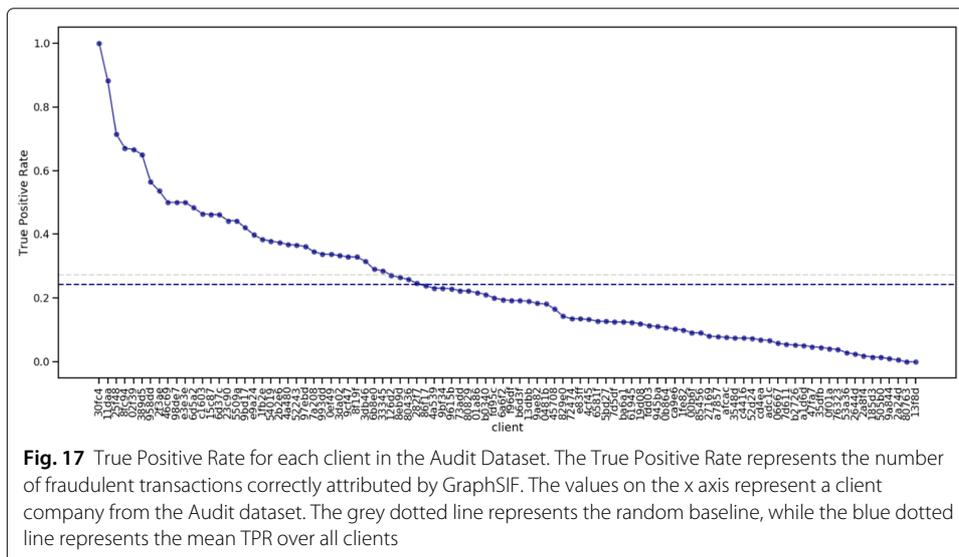
**Global Consistency** Figure 16 shows the True Negative Rate (TNR) for each client of the Audit dataset. We define True Negative Rate as the number of transactions given a low legitimacy label by both GraphSIF and the expert system, divided by the number of transactions given a low legitimacy label by the expert system. It shows how consistent GraphSIF is with the expert system regarding suspicious transactions. The dotted orange line shows the mean TNR over all the clients (0.53), while the grey dotted line shows the mean TNR of a random classification machine (0.29). We can see that while there is still room for improvement, GraphSIF shows promising results in terms of consistency.





Furthermore, Fig. 16 allows us to pinpoint a set of client on which GraphSIF is not accurate (rightmost ones). These clients are a promising starting point for further improvements.

Figure 17 shows the True Positive Rate (TPR) for each client in the Audit dataset. TPR is akin to TNR, but instead of counting the number of transactions given the low legitimacy label, the number of transactions given the high legitimacy number is computed. The TPR in our setting is fairly low (0.24), even lower than the one produced by randomly guessing a label for the transactions (0.27). This highlights the fact that GraphSIF tends to assign a low number of high legitimacy labels overall. Furthermore, the expert system’s usage of outside information might also explain this discrepancy.



However, in the context of fraud detection, it is more costly to assign a high legitimacy label to a fraudulent transaction than to assign a low legitimacy label to a legitimate transaction. Thus, while improving the TPR is important for the future, this discrepancy does not have a tremendous impact on GraphSIF usage.

### Discussion

While results shows that GraphSIF is able to provide consistent results in terms of consistency with the expert system, the most pressing issue is the lack of ground truths about actual legitimate and fraudulent transaction. While SiS-id's expert system provides an alternative to expert knowledge regarding the labeled transaction, this system is prone to error, most notably because fraud is dynamic and often changes faster than the rules of an expert system. Furthermore, comparing GraphSIF graph-based approach and SiS-id expert system is difficult, as they both rely on different types of knowledge (expert-based vs data-driven). However, in the absence of a dataset containing trusted labels, comparing our systems with SiS-id's expert system is the only way to propose an evaluation of their performance.

Although GraphSIF has been designed to handle Supplier Information Fraud, it could be used to detect other kinds of frauds in various domains. A possible application might be the detection of impersonation frauds over the telephone network, where a shift in the patterns of calls is observed when a fraudster takes control of the telephone line of a legitimate user. Similarly, GraphSIF could be used to detect unusual connection patterns in a computer network, that may be markers of a cyber-attack. Furthermore, the SOM - K-means model is agnostic to the specific features of the considered application, and thus can be used in other use-cases.

In this work, GraphSIF only uses the relationship between one client and its suppliers to perform its fraud detection. However, an interesting research direction would be to consider a graph modeling the relationships between all the clients and all the suppliers of the platform. Adapting some graph-based anomaly detection techniques to this graph might lead to interesting results.

### Conclusion

In this paper, we introduce GraphSIF, a novel feature-engineering process that creates a feature vector based on the relationships between a client company and the accounts it used to pay its supplier companies, providing a new tool to describe the underlying transaction mechanism involved in their interaction.

Several recent papers such as Wachs and Kertész (2019); Dubols et al. (2007) and Mongiovì et al. (2013) propose a human interpretation of the patterns uncovered by their approach and how they might suggest illegal behavior. The focus of our work is to emphasize on the variation of behavior, instead of the behavior itself. However, the relation between the uncovered patterns and fraud attempts is currently under investigation.

In conclusion, we used the temporal information contained in the transactions of the History dataset to create a behavior sequence composed of the transactions emitted by a client aggregated in several bounded time windows. We showed how to use this behavior sequence to create a data model based on Self-Organizing maps representing the behavior of a client company through time. We then used this data model to infer the legitimacy of new transactions using the K-means clustering algorithm, along with an aggregation

algorithm allowing us to combine the results obtained for different window sizes in a comprehensive score.

We presented the result of our classification system, first on a single company to investigate its performance locally, and then for a large set of companies from a large real-life dataset. We investigated the consistency of GraphSIF's results with the ones from the SiS-id expert system, and analyzed its operational efficiency, showing that GraphSIF is a fast alternative to the expert system.

#### Abbreviations

SIF: Supplier impersonation fraud; ADS: Anomaly detection system; SOM: Self-organizing map; B2B: Business-to-business; API: Application programming interface; TPR: True positive rate; TNR: True negative rate

#### Acknowledgements

The authors would like to acknowledge the help of the development team of SiS-id : François Agier & Kévin Lainte.

#### Authors' contributions

RC performed the experiments and wrote the article. OH proofread the article content. LB and LS provided supervision.

#### Funding

This research was performed using CIFRE funding 2016/0665.

#### Availability of data and materials

The datasets generated and/or analyzed during the current study are not publicly available due to the sensitivity of their economical content, but are available from the corresponding author on reasonable request.

#### Competing interests

The authors declare that they have no competing interests.

Received: 28 February 2020 Accepted: 7 July 2020

Published online: 22 July 2020

#### References

- Abdi H (2007) Z-scores. *Encycl Meas Stat* 3:1055–1058
- AIG (2019) Impersonation Fraud Claims Scenarios. American International Group, Inc, May 2015. [www.aig.com/content/dam/aig/america-canada/us/documents/business/management-liability/impersonation-fraud-claims-scenarios-brochure.pdf](http://www.aig.com/content/dam/aig/america-canada/us/documents/business/management-liability/impersonation-fraud-claims-scenarios-brochure.pdf)
- Akoglu L, McGlohon M, Faloutsos C (2010) OddBall: Spotting anomalies in weighted graphs. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp 410–421. [https://doi.org/10.1007/978-3-642-13672-6\\_4](https://doi.org/10.1007/978-3-642-13672-6_4)
- Akoglu L, Tong H, Koutra D (2015) Graph based anomaly detection and description: a survey. *Data Min Knowl Disc* 29(3):626–688. <https://doi.org/10.1007/s10618-014-0365-y>, <http://arxiv.org/abs/arXiv:1404.4679v2>
- Baesens B, Van Vlasselaer V, Verbeke W (2015) Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection. John Wiley & Sons
- Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. *J Mach Learn Res* 13(1):281–305
- Bolton RJ, Hand DJ, et al (2001) Unsupervised profiling methods for fraud detection. *Credit scoring and credit control* VII:235–255
- Bolton RJ, Hand DJ, Provost F, Breiman L, Bolton RJ, Hand DJ (2002) Statistical fraud detection: A review. *Stat Sci* 17(3):235–249. <https://doi.org/10.1214/ss/1042727940>
- Brockett PL, Xia X, Derrig RA (2006) Using Kohonen's self-organizing feature map to uncover automobile bodily injury claims fraud. *J Risk Insur* 65(2):245. <https://doi.org/10.2307/253535>
- Bullinaria JA (2004) Self organizing maps: fundamentals. Introduction to Neural: 1–15
- Canillas R, Hasan O, Sarrat L, Brunie L (2019) Supplier Impersonation Fraud Detection in Business-To-Business Transaction Networks Using Self-Organizing Maps. In: International Conference on Complex Networks and Their Applications. Springer. pp 599–610
- DFCG E-H (2019) Etude Fraude 2019 : Pour 6 Entreprises Sur 10, La Lutte Contre La Fraude N'est Pas Une Priorité. Euler-Hermes. <https://www.eulerhermes.fr/actualites/etude-fraude-2019.html>
- Dietterich TG (2000) Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems. Springer. pp 1–15
- Dubols LA, Gray DK, Tweedie EJ (2007) Surgical images: soft tissue: Calcinosis cutis. *Canadian journal of surgery* 50(3):217
- Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: A survey. *Knowl-Based Syst* 151:78–94. <https://doi.org/10.1016/j.knosys.2018.03.022>, <https://doi.org/1705.02801>
- Jain AK (2010) Data clustering: 50 years beyond K-means. *Pattern Recognit Lett* 31(8):651–666
- Kim Y, Sohn SY (2012) Stock fraud detection using peer group analysis. *Expert Syst Appl* 39(10):8986–8992. <https://doi.org/10.1016/j.eswa.2012.02.025>
- Kohonen T, A Self-learning musical grammar or 'associativememoryofthesecondkind' (1989). In: International Joint Conference on Neural Networks, Washington Vol. 1. pp 1–5. <https://doi.org/10.1109/ijcnn.1989.118552>
- Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. *Science* 298(5594):824–827. <https://doi.org/10.1126/science.298.5594.824>

- Mongiovi M, Bogdanov P, Ranca R, Papalexakisy EE, Faloutsos C, Singh AK (2013) NetSpot: Spotting significant anomalous regions on dynamic networks. In: Proceedings of the 2013 Siam international conference on data mining. pp 28–36
- Park H-S, Jun C-H (2009) A simple and fast algorithm for K-medoids clustering. *Expert Syst Appl* 36(2):3336–3341
- Priebe CE, Conroy JM, Marchette DJ, Park Y (2005) Scan statistics on enron graphs. *Comput Math Organ Theory* 11(3):229–247
- Qian Q, Xin M (2007) Research on hidden Markov model for system call anomaly detection. In: Pacific-Asia Workshop on Intelligence and Security Informatics, Springer. pp 152–159. [http://link.springer.com/chapter/10.1007/978-3-540-71549-8\\_17](http://link.springer.com/chapter/10.1007/978-3-540-71549-8_17)
- Sadowksi G, Rathle P (2014) Fraud detection: Discovering connections with graph databases. White Paper-Neo Technology-Graphs are Everywhere 13
- Trunk GV (1979) A problem of dimensionality: A simple example. *IEEE Trans Pattern Anal Mach Intell PAMI-1*(3):306–307. <https://doi.org/10.1109/tpami.1979.4766926>
- Tsunenori I (2000) *Lect Notes Comput Sci* 1983:17–22. [https://doi.org/10.1007/3-540-44491-2\\_3](https://doi.org/10.1007/3-540-44491-2_3)
- Van Vlasselaer V, Eliassi-Rad T, Akoglu L, Snoeck M, Baesens B (2016) Gotcha! Network-based fraud detection for social security fraud. *Manag Sci* 63(9):3090–3110. <https://doi.org/10.1287/mnsc.2016.2489>
- Wachs J, Kertész J (2019) A network approach to cartel detection in public auction markets. *Sci Rep* 9(1):1–10. <https://doi.org/10.1038/s41598-019-47198-1>. <https://doi.org/arXiv:1906.08667v1>

### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Submit your manuscript to a SpringerOpen<sup>®</sup> journal and benefit from:**

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

---

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)

---