Computational Discovery of Transaction-Based Financial Crime via Grammatical Evolution: the Case of Ponzi Schemes

Peter Fratrič[1], Giovanni Sileno[1], Tom van Engers[1,2], and Sander Klous[1]

[1] Informatics Institute, University of Amsterdam, the Netherlands
[2] Leibniz Institute, TNO/University of Amsterdam, the Netherlands
p.fratric@uva.nl

Abstract. The financial sector continues to experience wide digitalization; the resulting transactional activity creates large amounts of data, in principle enabling public and private actors to better understand the social domain they operate on, possibly facilitating the design of interventions to reduce illegal activity. However, the adversarial nature of frauds and the relatively low amount of observed instances make the problem especially challenging with standard statistical-based methods. To address such fundamental issues to non-compliance detection, this paper presents a proof-of-concept of a methodological framework based on automated discovery of instances of non-compliant behaviour in a simulation environment via grammatical evolution. We illustrate the methodology with an experiment capable of discovering two known types of Ponzi schemes from a modest set of assumptions.

1 Introduction

Financial crime occurs at many levels of society, from credit card fraud, to tax fraud, money laundering, terrorist financing, financial market manipulation, up to the corruption of the highest representatives of individual countries or international political bodies. A unifying aspect of all these instances of non-compliance is that the transaction of assets with the aim of illegal profit is typically conducted in such a way that no suspicion of illicit activity arises. In order to detect non-compliant activity from available evidence, researchers and analysts have applied over the years various computational methods, ranging from rule-based systems, knowledge graphs, machine learning models, to executable models of social systems. Although these applications have shown various levels of success, several issues remain at present, still exploited by non-compliant actors [17,4].

Research background Synthetizing non-compliant behaviour into a set of patterns, either explicitly via a set of logical rules, or implicitly by some machine learning method, typically face difficulties as e.g. explainability (for ML-based methods), unavailability of data, high false positive rate, or overlooking the adaptability of non-compliant agents. All these issues make traditional approaches both ineffective and inefficient, particularly on the medium-longer term.

We will elaborate therefore on three other trends observed in the literature. According to a recent review [3], *network analysis* tools have been slowly finding their way into prominence. These tools capitalize on the ability of networks to represent complex relationships, and at the same time being both interpretable and easy to visualize. Once the transaction graph is formed, the main goal becomes essentially to detect non-compliant individuals, suspicious events, or anomalous structures [2,1]. Several methods of this type have already been proposed in the area of financial fraud detection [23]. Comparatively, approaches based on *modeling and simulation* are covered by a much smaller number of studies [18,6], most of which focus on the possibility of training detection models on simulated data, also to mitigate the issue of high false negatives. Only a few simulation environments were developed in the literature to generate illicit transaction activity. Instead, the issue of adaptivity is addressed mostly in the context of *adversarial machine learning* [25,12,14,16]. However, this approach targets the local fraud space defined by the parametric model determined by the dataset, so it can hardly generalize to illicit behaviour in a global sense, i.e. not included in the data or encoded in the classifier. If we target the design of intelligent agents autonomously learning frauds by interacting with the environment (and so capable of generating new illicit behaviours), the number of studies is even lower, e.g. co-evolutionary methods to discover tax frauds tested in a transaction tax network environment in [15]; reinforcement learning [19] to design an agent learning credit card fraud in an adversarial environment.

Generalizing to any kind of adversarial system where the detection model is tested against a model of an adaptive perpetrator, the research seems to be progressing faster in other areas. For instance, adversarial systems are more extensively studied in the area of artificial intelligence [7], although still on a relatively low scale level. The area of cybersecurity is advancing comparatively faster than the socio-legal domain, probably because the implementation of a model of cyber environment is less of a challenge compared to social systems, which means model-based testing methods can be effectively implemented [24].

Aims and contribution of the paper Fraud schemes target specific vulnerabilities of a socio-legal system and/or psychological weaknesses of its victims, and very often exhibit a modular structure: more complex schemes tend to be modifications of simpler ones. This short paper presents and elaborates on this intuition, focusing on Ponzi schemes (PSs) implemented on distributed ledger, i.e. smart contracts. The reason why we choose specifically smart contract PSs is that the complex legal terminological nuances involved in arbitrary contracts are mitigated with smart contracts because of their mechanistic transaction environment. This, and public availability of data, makes the distributed ledger suitable for the type of investigations. Moreover, due to popularization of this technology, the question of smart contract PS detection is a pressing issue [8,9].

In our study, we pursue a long term goal of developing a *fraud discovery assistant*, where illicit behaviour can be generated depending on presumed observables, socio-psychological modules of the simulation model, or potentially even the implemented countermeasures. At this point of research, we focus on

the first (generation) and briefly discuss the second (internal socio-psychological modules) and third (couuntermeasures) aspects. Depending on observables considered, two known types of PSs will be discovered using grammatical evolution. The illicit activity discovered and simulated in the model will be visualized as a series of snapshots of the transaction network.

Case study: types of Ponzi schemes Various types of smart contract PSs already exist on the Ethereum blockchain [5]. In its *basic* version, each time a new participant enters the scheme, the entry fee is redistributed equally among other participants. A modification that aims to create a community of highly profitable users can be implemented by imposing a preference ordering on the capital redistribution. For example, to exploit risk-appetite and deceivability of society, early-stage investors can be benefited by repaying the premium chronologically; therefore, the participants that joined later might not be repaid once the capital is depleted. This type of smart contract PS is known as *waterfall type*. Another type, that is in a way a modification of the previous two types, is the *array* type. In this case, the redistribution mechanism keeps track of which participant was paid last in order to equalize the frequency of payments, but at the same time pays the next participant only if there is enough capital to send a payment exceeding the entry fee of the participant. This means that it prioritizes the size of the user base that is already in profit, therefore it is more likely the scheme will be perceived as a valid investment in the society. Clearly, there can be more sophisticated variations of smart contract PSs, for example including a reward for participants recruiting new users; however, for our current aims, the two previous schemes are sufficient.

2 General Framework

In order to generate and evaluate possible fraud schemes, we propose a framework for (re)construction of non-compliant behaviour that requires four operational components. These are: (i) a *search space* defined by an action space (in which a fraud scheme can be constructed); (ii) a *simulation environment* to execute actions of agents including a *non-compliant* (or *fraudulent*) *agent*, in which (iii) a *fitness function* can be calculated to determine how good each scheme is by evaluating its outcome; (iv) a *search algorithm* to explore the search space, that we typically identify with the reasoning mechanism of the non-compliant agent. If all four instruments are well-defined, then it is possible to (re)create the fraudulent behaviour as illustrated on Figure 1.

Expert knowledge is used to formulate hypotheses about the functioning of the simulation environment and representation of the search space, including relevant observables for the non-compliant agent (search algorithm). The inner loop (continuous lines) searching the space of non-compliant schemes produces a dataset of transaction schemes. The set of hypotheses can be subsequently extended with assumptions that give rise to new instances of non-compliant behaviour (dashed lines).

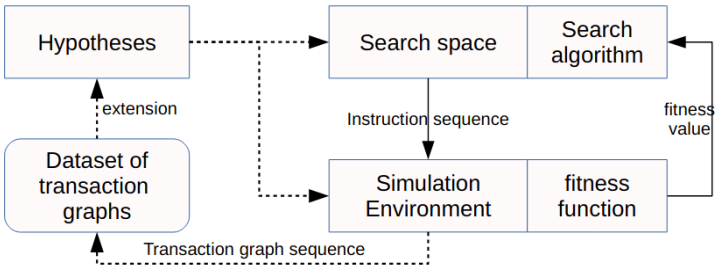

Fig. 1. Overview of methodology to (re)construct non-compliant behaviour

Arguably, a *detection algorithm* is also an important part of the theoretical framework. However, at the current stage of research, we consider it as essentially a different research question subsequent to the generation problem (see the discussion section for insights on future directions). Moreover, note that simulation environment can be also combined with additional goals as assessing algorithmic fairness (see e.g. [11]).

3 Generation of Ponzi schemes

In general, a simulation model generates a sequence of transactional graph snapshots $G_0, ..., G_T$ as a record of agents interacting in the environment for a finite number of steps T. The fraudulent subgraph sequence $H_0, .., H_t$ for $t < T$ is generated during the simulation process, in association with non-compliant activity. For the sake of this study, the agent-based model will be designed to be minimalistic, which means that only the minimal set of assumptions necessary to approximate PS mechanism will be employed. Since PSs do not in principle depend on transactions that are happening outside the scheme, it is not needed to assume any direct transactional interactions between the agents. While this might intuitively seem an unsound manner to model social systems, it turns out to be an advantage, because it allows us to generate data related to illicit behaviour only by using the assumptions necessary for the illicit behaviour to arise. Obviously, it is true that other forms of interactions happen in the real world during a PS spread, but this extension is needed only for more sophisticated types of schemes (see p. 9).

In practice, no additional economic activity producing value is assumed in the model, which means except for trivial cases every transaction sequence is a PS. This simplifies the modelling as there is no need to define a PS either on a phenomenological or a logical basis. Yet, the adaptation mechanism to find profitable schemes will plausibly work for more complex settings (e.g. societal policies, physical constraints, additional economic actions). Consequently, by relaxing this assumption, more sophisticated schemes can be addressed.

Contract mechanism We assume that the initial transaction graph G_0 is an empty graph with $N + 2$ nodes; one node represents the contract, one node

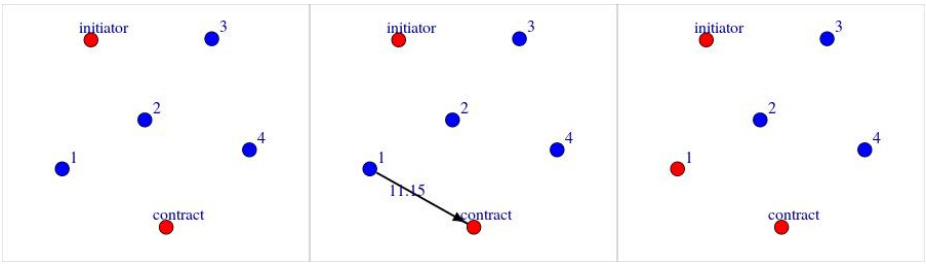

Fig. 2. Initial state is an empty transaction graph G_0 (blue nodes) and subgraph H_0 (red nodes). In G_1 a transfer of 11.15 cash is made between the node 1 and the contract node. In the state G_2 the subgraph H_2 has three nodes.

represents the initiator of the contract, and N nodes represent the agents that can join the contract. The contract and its members constitute the subgraph sequence $H_0, ..., H_T$ and therefore the initial subgraph H_0 will consist only of two nodes (the contract node and the initiator) and zero edges.

The contract has two attributes: a deposit account and a list of members. In order to join the contract, an agent needs to send an entry fee, that is initialized for each agent randomly from a prespecified distribution. If an agent decides to join the contract, the entry fee is sent to the contract deposit address. The mechanism, specified in the following excerpt, is illustrated on Figure 2.

```
if contract.isTrustworthy(agent.threshold):
    G.addEdge(agent, contract, agent.entryFee)
    G.executeTransactions()
```

In the simulation environment, the method `addEdge` adds a directed edge from the node in the first argument to the node in the second argument. The method `executeTransactions` is called on a graph object by an agent to execute transactions defined by the edges. The `threshold` attribute of an agent and the contract method `isTrustworthy` serve to model the agents' attitude, and will be explained in the next section.

Contract trustworthiness and agent trust During the simulation, the *trustworthiness* of the contract is calculated as a numerical value in the zero-one interval. There is no general agreement what exactly makes a contract trustworthy to people; for the sake of example we consider two plausible basic assumptions, and define a function based on these assumptions[3]. The trustworthiness Tr of a contract is (a) proportional to the relative amount of agents in profit n_+ compared to the number of agents n that have already joined; (b) inversely proportional to the root of the density of agents that joined the contract. Following these two

[3] The model of trust used in this study is simplistic and serves only for the purpose of demonstration. For overviews on trust models see, e.g. [22,10].

assumptions, we can define the function:

$$Tr(n_+, n) = \frac{1 + n_+}{n} \cdot \left(\frac{n}{N}\right)^{\frac{1}{K}} \tag{1}$$

where $K > 1$ is a societal coefficient, that controls the interplay between the two assumptions.

Each agent has an internal threshold that determines if an agent joins the contract. The method `isTrustworthy` returns `True` if the trustworthiness of the contract is higher than the threshold of the agent, and `False` otherwise. We assume that the internal threshold parameter of each agent is proportional to the entry fee the agent is willing to pay to join the contract, because we deem a plausible assumption that the agents considering to pay more will be equally more skeptical of their investment.

Fitness Clearly, the profit attained by the PS heavily depends on the equation (1), and therefore the scheme initiator needs to decide the optimal redistribution of capital such that the scheme is attractive for the agents in the environment. This means that the initiator needs to balance out short-term profit with long-term sustainability of the scheme. This is a core parameter for this type of non-compliant behaviour. The fitness of a PS is then defined as the amount of capital generated for the initiator, that is, the amount of redistributed capital that ends up in the deposit address of the initiator node.

Search space representation Once the graph G_0 and the subgraph H_0 are initiated, the scheme is defined by its specific capital redistribution structure. This redistribution structure consists of a set of logical rules that evolve the transaction graph, deciding which members should be paid. In our framework, the characteristic form of the PS is expressed as illustrated below:

```
if contract.FeeReceived(new_user):
    H.addNode(new_user)
    H.addEdge(new_user, contract, new_user.entryFee)
    G.executeTransactions()
    try:
        H.evaluate(instructionSequence)
        G.executeTransactions()
    else:
        exit()
```

The set of instructions `instructionSequence` consists of instruction that modify the payment scheme H_{t-1}. Then the payments defined by the modified graph H_t are carried out by the `executePayments()` method[4]. Note that an entry condition can be considered for potential new users, e.g. a minimal entry fee. For simplicity, we assume no special conditions are in place: anyone can join.

[4] It can be argued whether `executePayments` should be called after every instruction, or after graph modifications, e.g. `AddNode` already applied during the evaluation of `instructionSequence`; however, this choice does not affect the model profoundly.

The syntax of the instructions `instructionSequence` will be defined by a context-free grammar, defined in Backus-Naur form below:

```
<instruction> ::= <clause> ; <instruction> | <clause>
<clause>  ::= if (<premise>) <action>
<action>  ::= H.addNode(<node>) | H.addEdge(contract, <node>, <weight>)
  | H.removeEdges(contract, <node>)
```

The actions `addNode`, `addEdge`, and `removeEdges`, add node, edge, and remove edges of the subgraph H_t respectively. The rest of the terminal symbols will be formulated later to illustrate how specific assumptions of observables depend on what kind of scheme is generated. In the trivial case displayed on Figure 2, the instruction sequence would consist of three instructions: `H.addNode(1)`, `H.addEdge(contract, 1, 11.15)`, and `H.removeEdges(contract, 1)`, where 11.15 is the weight of an edge that corresponds to the amount of currency transferred. As will be defined later, the symbol `<node>` can be replaced by a variable, which means the instruction sequence of the initiator essentially acts as an open formula that is grounded in an event of a new agent joining the PS[5].

Note that not all words generated in the exploration are semantically correct (e.g. adding a node that was already added), which is why the code above requires `try` method to call the `exit()` method if an error is detected on runtime.

4 Experiments and Results

The present work empirically demonstrates how two PS types can be discovered based on the introduction of hypothetically relevant observables, as following the methodology described in Figure 1. In practice, the context-free grammar presented above is extended with further terminal symbols (standing for the hypothetical relevant predictors), i.e. dedicated *query-methods* (used by agents to perceive some property from the environment), and *premises* (used by agents to condition performance).

In our experiments, the search algorithm used by the agent to discover new instances of illicit schemes from the given set of predictors is *grammatical evolution* [21].[6] For the simulation environment, we will consider $N = 100$ agents and the societal coefficient K of the trustworthiness function will be set to 10. The distribution of the entry fees follows a Beta distribution with both first and second shape parameter equal to two, which means the distribution approximates a Gaussian. The sampled value from the Beta distribution is scaled by a factor of 10 for better readability.

[5] This reflects the event-driven architecture integrated into the smart contract programming language Solidity. In general, it captures the cyclic characteristic of fraudulent business models.

[6] In general, grammatical evolution is an evolutionary algorithm where words of a grammar are mapped to integer vectors, and an evolutionary optimization procedure is used to optimize the fitness function. Then integer vectors are mapped back to words.

Waterfall-type topology The basic Waterfall type of PS can be found by including a set of rather trivial terminal symbols into the grammar. In the instruction sequence, it must be indicated who is the initiator of the scheme and who is the new user that wants to join. The `getFee` method returns the entry fee of the agents that already joined. Further on, the possible percentages of either the entry fee or the contract balance to be paid to the contract participants are assumed. A handy piece of information to include is the `NUsers` query-method, that takes as an argument an integer and returns true if the number of market participants is equal to the argument.

```
<premise> ::= TRUE | NUsersEq(<int>)
<node> ::= new_user | initiator
<weight>  ::= <percentage>·getFee(<node>) | <percentage>·contract.balance
<percentage> ::= 0.06 | 0.1 | 0.2 | 0.5 | 1.2 | 2
<int> ::= 1 | 2 | 3 | 4 | 5 | 10 | 50
```

By visually analysing the best 20 generated transaction graph sequences, we have observed that all of them had a star graph structure, that is typical for the waterfall type (Figure 3). The only deviation from this pattern occurred when the algorithm decided to send capital to the initiator only after a sufficiently high number of contract participants was reached. This means that the evolutionary algorithm discovered that the spread of the PS is greater, and therefore also the amount of capital accumulated, if the capital is redistributed more generously at the beginning.

Array topology As already discussed, the waterfall type can be made more efficient if the scheme will have a concept of who was paid last and who should be paid next. Indeed, each time the method `executeTransactions` is called for

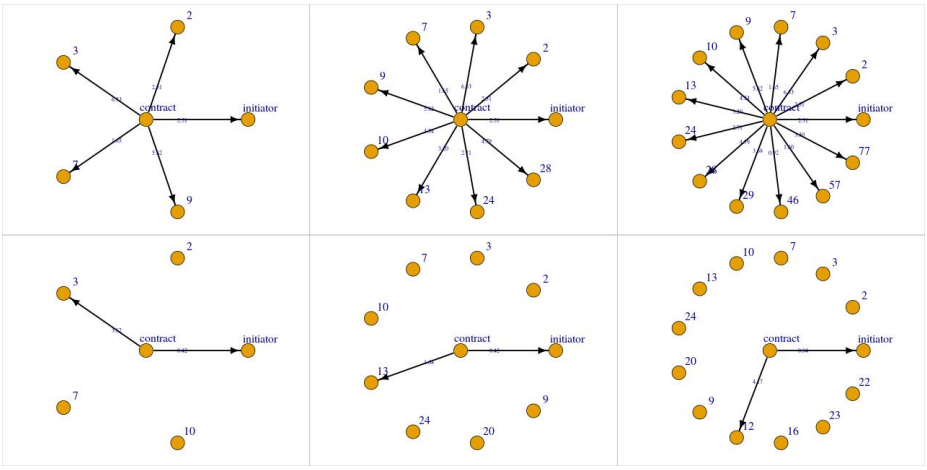

Fig. 3. Snapshots of the transaction graph for the waterfall type (upper row) and the array type (lower row) of Ponzi Schemes (PS).

the waterfall scheme, many of the transactions cannot be executed because the capital of the contract is depleted. This issue can be resolved by including three more query-methods into the grammar, modifying it as such:

```
<premise> ::= TRUE | NUsersEq(<int>)
  | BalanceFeeHigherAndNotConnected(<node>, <weight>)
<node> ::= new_user | initiator | getNextToPay() | getLastPaid()
```

where `getNextToPay` and `getLastPaid` keep track of which agent was paid last and which agent is next in order; `BalanceFeeHigherAndNotConnected` returns true if the contract node is not connected to the node provided by the first argument and the capital balance of the contract is higher than the second argument. Running the search procedure over the extended search space yields the *array* type of PS. The typical "clock" pattern is depicted on Figure 3. As in the waterfall case, a number of different transaction patterns were discovered. These mutants differ slightly in the transaction topology, usually adding one or two instructions more into the instruction sequence.

5 Discussion

Convergence Computation-wise, the evolutionary algorithm for a population of size 100 with other parameters kept default [21] can easily find profitable PSs of Waterfall type in less than 200 iterations. In order to discover the Array-type, the algorithm had to be extended with an adaptive mutation chance parameter to avoid getting trapped in local optima[7] of the Waterfall-type. Moreover, the *maximum depth* parameter, required to ensure termination of the recursive grammar defined above, is increased due to the higher complexity of the scheme.

Extending to other types of Ponzi schemes Other types of PS can be explored similarly. For instance, to include a PS spread mechanism as those observed in social systems [20], a simple network spread model [13] can be implemented, extending the grammar accordingly. Participants to the scheme has to be rewarded by how many new members they have recruited. Adding to the grammar a query-method that returns the number of new contract participants recruited by an agent would be sufficient in terms of predictors. However, a mechanism that motivates the individual agent to recruit new participants would also need to be present.

Detection supported by generation Without loss of generality, assume we use a certain neural network to decide whether a certain behaviour is compliant or not. This detection model can be trained on labelled data records of a given socio-economic system, and then tested also on labelled data obtained via the simulation environment. The training dataset can be subsequently extended by generated instances of non-compliant behaviour to enhance the performance of

[7] See the source code: `https://github.com/fratric/Ponzi-Scheme-Discovery`

the classifier, mitigating the issue of unbalanced datasets as motivated in the introduction. Note that these instances can correspond to previously unobserved types of non-compliant behaviour. More interestingly, since the noncompliant agents are assumed to be capable to adapt, the detection model can be also used to incentivize the discovery of new schemes, producing co-evolutionary adversarial dynamics (see e.g. [15]).

Beyond grammatical evolution Grammatical evolution can be challenging to use for more complex applications, both regarding the computational complexity and the representation by a context-free grammar. In systems consisting of several transactional sub-systems, where a variety of transaction operations can occur, the search space represented by a context-free grammar would be too large to be explored using evolutionary operators. In such case, the search algorithm associated with the noncompliant agent (or a group of agents) would require a more sophisticated type of reasoning, e.g. bringing some context into the grammar, such that the noncompliant agent(s) are capable to plan ahead depending on the environment and the actions of other agents, thus allowing modularization of the search space. Moreover, the behaviour of the noncompliant agent ought not to be deterministic, which is also important for generation of rich synthetic data. However, this challenge is similar to planning and cooperation in complex, diverse and stochastic environments which remain still open questions.

6 Conclusions

Our present research deals with exploration of non-compliant behaviours in the context of policy-making. The paper sketched a general computational framework to generate instances of transaction-based financial crime and illustrated its application on a well known case of smart contract Ponzi schemes. It was demonstrated that with only a modest set of assumptions it is possible to generate a sequence of transaction graphs that captures the functional and modular aspects of two well-known types of Ponzi schemes, that differ in their dynamic topology defining the redistribution of capital. We argue that the lines of research revisited in this paper are relatively unexplored and deserve much more attention, as they have the potential to successfully address certain important issues present in the contemporary research on fraud detection. However, more examples of fraud generated in simulation environment needs to be provided before creating a sound basis for deployment into real socio-economical systems.

References

1. Aggarwal, C.C.: An introduction to outlier analysis. In: Outlier analysis, pp. 1–34. Springer (2017)
2. Ahmed, M., Naser Mahmood, A., Hu, J.: A survey of network anomaly detection techniques. Journal of Network and Computer Applications **60**, 19–31 (2016)

3. Al-Hashedi, K.G., Magalingam, P.: Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019. Computer Science Review **40**, 100402 (2021)
4. Bao, Y., Hilary, G., Ke, B.: Artificial Intelligence and Fraud Detection. SSRN Electronic Journal **7**(2), 223–247 (2020)
5. Bartoletti, M., Carta, S., Cimoli, T., Saia, R.: Dissecting Ponzi schemes on Ethereum: Identification, analysis, and impact. Future Generation Computer Systems **102**, 259–277 (2020)
6. Brito, J., Campos, P., Leite, R.: An Agent-Based Model for Detection in Economic Networks. Communications in Computer and Information Science, vol. 887, pp. 105–115. Springer International Publishing, Cham (2018)
7. Caminero, G., Lopez-Martin, M., Carro, B.: Adversarial environment reinforcement learning algorithm for intrusion detection. Computer Networks **159**, 96–109 (2019)
8. Chen, W., Zheng, Z., Cui, J., Ngai, E., Zheng, P., Zhou, Y.: Detecting Ponzi Schemes on Ethereum. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18. pp. 1409–1418. ACM Press, New York, New York, USA (2018)
9. Chen, W., Zheng, Z., Ngai, E.C., Zheng, P., Zhou, Y.: Exploiting Blockchain Data to Detect Smart Ponzi Schemes on Ethereum. IEEE Access **7**, 37575–37586 (2019)
10. Cho, J.H., Chan, K., Adali, S.: A survey on trust modeling. ACM Computing Surveys (CSUR) **48**(2), 1–40 (2015)
11. D'Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., Halpern, Y.: Fairness is not static. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. pp. 525–534. ACM, New York, NY, USA (2020)
12. Delecourt, S., Guo, L.: Building a Robust Mobile Payment Fraud Detection System with Adversarial Examples. In: 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE). pp. 103–106. IEEE (2019)
13. Fu, P., Zhu, A., Ni, H., Zhao, X., Li, X.: Threshold behaviors of social dynamics and financial outcomes of ponzi scheme diffusion in complex networks. Physica A: Statistical Mechanics and Its Applications **490**, 632–642 (2018)
14. Fursov, I., Morozov, M., Kaploukhaya, N., Kovtun, E., Rivera-Castro, R., Gusev, G., Babaev, D., Kireev, I., Zaytsev, A., Burnaev, E.: Adversarial Attacks on Deep Models for Financial Transaction Records. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. vol. 1, pp. 2868–2878. ACM, New York, NY, USA (2021)
15. Hemberg, E., Rosen, J., Warner, G., Wijesinghe, S., O'Reilly, U.M.: Detecting tax evasion: a co-evolutionary approach. Artificial Intelligence and Law **24**(2), 149–182 (2016)
16. Kumar, N., Vimal, S., Kayathwal, K., Dhama, G.: Evolutionary Adversarial Attacks on Payment Systems. In: 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 813–818. IEEE (2021)
17. Kurshan, E., Shen, H., Yu, H.: Financial crime & fraud detection using graph computing: Application considerations & outlook. In: 2020 Second International Conference on Transdisciplinary AI (TransAI). pp. 125–130. IEEE (2020)
18. Lopez-Rojas, E.A., Axelsson, S.: A review of computer simulation for fraud detection research in financial datasets. In: 2016 Future Technologies Conference (FTC). pp. 932–935. No. December, IEEE (2016)
19. Mead, A., Lewris, T., Prasanth, S., Adams, S., Alonzi, P., Beling, P.: Detecting fraud in adversarial environments: A reinforcement learning approach. In: 2018

Systems and Information Engineering Design Symposium (SIEDS). pp. 118–122. IEEE (2018)
20. Nash, R., Bouchard, M., Malm, A.: Investing in people: The role of social networks in the diffusion of a large-scale fraud. Social Networks **35**(4), 686–698 (2013)
21. Noorian, F., de Silva, A.M., Leong, P.H.W.: gramEvol: Grammatical evolution in R. Journal of Statistical Software **71**(1), 1–26 (2016)
22. Pinyol, I., Sabater-Mir, J.: Computational trust and reputation models for open multi-agent systems: a review. Artificial Intelligence Review **40**(1), 1–25 (2013)
23. Pourhabibi, T., Ong, K.L., Kam, B.H., Boo, Y.L.: Fraud detection: A systematic literature review of graph-based anomaly detection approaches. Decision Support Systems **133**, 113303 (2020)
24. Utting, M., Legeard, B.: Practical model-based testing: a tools approach. Elsevier (2010)
25. Zeager, M.F., Sridhar, A., Fogal, N., Adams, S., Brown, D.E., Beling, P.A.: Adversarial learning in credit card fraud detection. In: 2017 Systems and Information Engineering Design Symposium (SIEDS). pp. 112–116. IEEE (2017)