Rogue behavior detection in NoSQL graph databases

Arnaud Castelltort, Anne Laurent

Rogue Behavior Detection in NoSQL Graph Databases

Arnaud Castelltort, Anne Laurent

LIRMM - Univ. of Montpellier - CNRS UMR 5506
Montpellier, France

Abstract

Rogue behaviors refer to behavioral anomalies that can occur in human activities and that can thus be retrieved from human generated data. In this paper, we aim at showing that NoSQL graph databases are a useful tool for this purpose. Indeed these database engines exploit property graphs that can easily represent human and object interactions whatever the volume and complexity of the data. These interactions lead to fraud rings in the graphs in the form of sophisticated chains of indirect links between fraudsters representing successive transactions (money, communications, etc.) from which rogue behaviours are detected. Our work is based on two extensions of such NoSQL graph databases. The first extension allows the handling of time-variant data while the second one is devoted to the management of imprecise queries with a DSL (to define flexible operators and operations with Scala) and the Cypherf declarative flexible query language over NoSQL graph databases. These extensions allow to better address and describe sophisticated frauds. Feasibility have been studied to assess our proposition.

Keywords: Rogue Behavior, Fraud Rings, NoSQL Graph Databases, Fuzzy DSL, Approximate Cypher Queries.

[☆] Fully documented templates are available in the elsarticle package on CTAN.

1. Introduction

Rogue behaviors are known to lead to important economic and political concerns. Frauds in banks and insurance companies represent billions of dollars lost every year. For instance, more than £52 billion was lost in the UK in 2013[1]. Fraud can be detected by considering abnormal patterns in the interactions. However, these anomalies are hidden and often difficult to retrieve because of their complexity. Fraud can be committed by one or more persons. It can impact on individuals or organizations (e.g., banks).

As rogue behaviors are characterized by the interactions, graphs can thus help for retrieving frauds. Graphs are indeed recognized to play an important role within the pattern recognition field [2], thus being a key technology for retrieving relevant information. Graphs efficiently represent the relationships between objects, should they refer to persons, organizations, or scientific data (e.g., chemistry). Techniques and algorithms can be distinguished in considering the fact that they are meant to mine relevant patterns or to retrieve them.

Detection is achieved through the modelization of fraud rings which are hidden within the graph of interactions. A fraud ring is a set of connections between actors. It can be found in many fraud frameworks [1].

Although graphs have been studied since the very beginning of computer science in the so-called graph theory field [3], their integration within database management systems is more recent. Some of the first systems have been proposed with the emergence of ontologies and RDF triplets queried through SPARQL [4]. More recently, NoSQL databases have proposed efficient engines devoted to graph databases: GraphDB, Neo4J, etc. [5] compares some of these engines and points the advantages of the Neo4J system, which is the one we consider.

In this paper, we propose a framework for defining fuzzy temporal pattern matching from NoSQL graph databases. For this purpose, we first recall the basic concepts of NoSQL graph databases, temporal queries and graph pattern matching in Section 2. We then detail the problem we address in Section 3,

before presenting a first attempt for addressing the problem using the NoSQL Neo4j graph database in Section 4. The proposition has been implemented. The main contribution of this paper is presented in Section 5. This contribution is mainly based on the use of generalized fuzzy queries. These queries can be user-defined and rely on a a Domain Specific Language (DSL) and on an extension of the declarative query language to better address and describe sophisticated frauds. Section 6 reviews the main contributions from the literature related to rogue detection. Section 7 sums up this paper and presents the future work we would like to address.

2. Preliminary Statements

2.1. Graphs

Graphs have been studied for a long time by mathematicians and computer scientists. A graph can be directed or not. It is defined as follows.

Definition 1 (Graph). *A graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq (V \times V)$.*

Definition 2 (Directed Graph). *A directed graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$. That is E is a subset of all ordered permutations of V element pairs.*

When used in real world applications, graphs need to be provided with the capacity to label nodes and relations, thus leading to the so-called labeled graphs, or property graphs as shown in Fig. 1 and defined bellow:

Definition 3 (Labeled Oriented Graph). *A labeled oriented graph G, also known as oriented property graph, is given by a quadruplet (V, E, α, β) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$, α stands for the set of attributes defined over the nodes, and β the set of attributes defined over the relations.*

Given such graphs, it is possible to retrieve subgraphs, as described below.

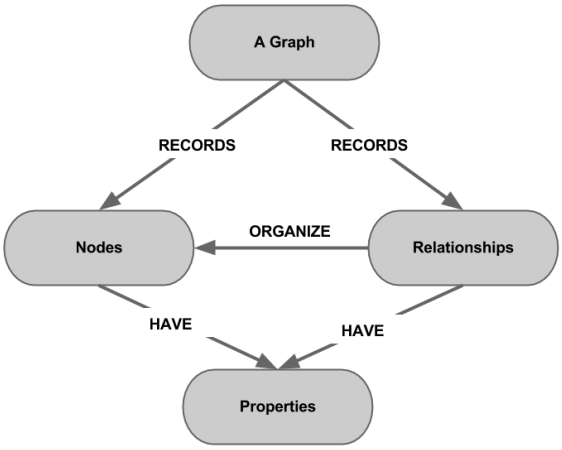

Figure 1: Labeled Graph

2.2. Graph Pattern Matching and Querying

Graph pattern matching is a very difficult algorithmic problem that has lead to the production of many works. We focus here on the definition and usage of pattern matching queries. Some related works are presented in Section 6.

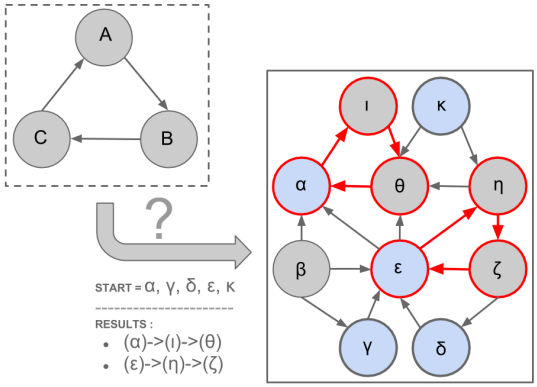

Figure 2: Pattern Matching

The goal of pattern matching is to retrieve a pattern from data. In graph pattern matching, the pattern and the source data are both organized as graphs, as illustrated in Fig. 2.

More formally, graph pattern matching amounts to retrieve all occurrences of a graph pattern Q from a source graph G. The problem of deciding whether a subgraph is included within another one is known as subgraph isomorphism, which is known to be NP-complete.

Definition 4 (Subgraph Isomorphism). *Let $Q = (V_Q, E_Q)$ and $G = (V, E)$ be graphs. A subgraph isomorphism from Q to G is a function $f : V_Q \rightarrow V$ such that if $(u, v) \in E_Q$, then $(f(u), f(v)) \in E$. f is an induced subgraph isomorphism if in addition if $(u, v) \notin E_Q$, then $(f(u), f(v)) \notin E$.*

Subgraph matching aims at retrieving the occurrences and does not only focus on the decision problem, as defined above from [6].

Definition 5 (Subgraph Matching). *For a graph G and a subgraph query Q, the goal of subgraph matching is to find every subgraph $S = (V_S, E_S)$ in G such that there exists a bijection $f : V_Q \rightarrow V_S$ that satisfies $\forall V \in V_Q$, $T_Q(v) = T_S(f(v))$ and $\forall e = (u, v) \in E_Q$, $(f(u), f(v)) \in E_S$, where $T_S(f(v))$ represents the label of the vertex $f(v)$ in S.*

Such pattern matching techniques are implemented in the so-called NoSQL graph databases together with many other features, thus allowing these frameworks to efficiently address rogue behavior detection.

2.3. NoSQL Graph Databases

NoSQL graph databases [7] are based on these concepts, attributes and values over the attributes being stored thanks to the $(key, value)$ paradigm which is very common in NoSQL databases.

Fig. 3 shows a graph and its structure in $(key, value)$ pairs.

Studies have shown that these technologies present good performances, far better than classical relational databases, for representing and querying such large graph databases. There exist several NoSQL graph database engines (OrientDB, Neo4J,HyperGraphDB, etc.) [8]. Neo4J is recognized as being among the top engines in terms of performance [5]. It has been recently extended

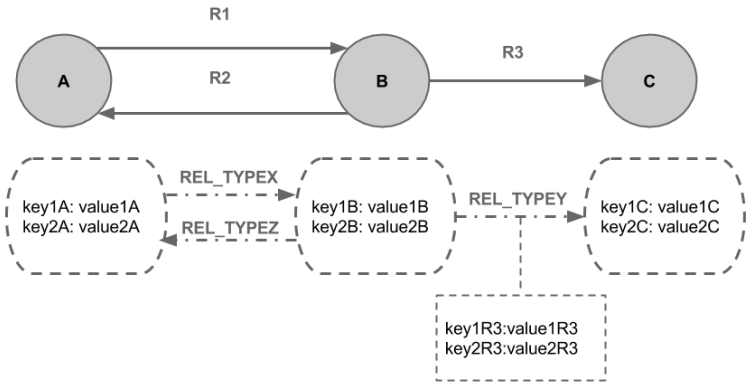

Figure 3: Properties of Nodes and Relations

for managing the successive versions of the graph database in the Mnemosyne system described below.

2.4. Mnemosyne: an Innovative Temporal Data Management System

[9] aims at proposing an innovative model of history management in NoSQL graph databases. There have been several proposals in the literature for managing evolving graphs [10]. [9] focuses on property graphs and is based on three key concepts: (i) Using the graph to manage time-variant data; (ii) Totally decoupling data historization from the representation of the data in the graph datasource; (iii) Using generic graph traversals regardless of the graph datasource.

One of the main specificity of the Mnemosyne system is that it uses two graphs (i) a DataGraph that is the graph currently in use; (ii) a VersionGraph that stores the history of different versions of a data graph. The VersionGraph model does not depend on the DataGraph.

Fig. 4 shows an example of a DataGraph and a VersionGraph.

In this system, every node and relation in the graph database is tracked in the VersionGraph with a node called *TraceElement*. From this node, all the modifications (insert, update, create) are kept in a list of *RevisionElements* that can be queried generically for elements (nodes or relations) of the graph

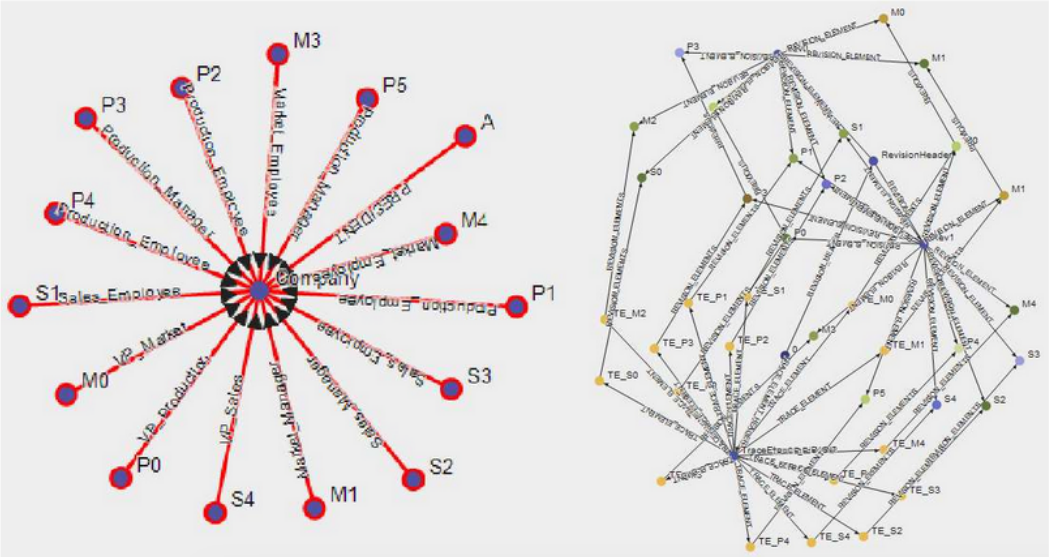

Figure 4: DataGraph (left) and VersionGraph (right)

source.

Fig. 5 illustrates this proposal. In this example, three nodes (a,b,c) and three relations (r1,r2,r3) from Fig. 6 are traced. a (res. b, c) is the node related to node A (B, C) from Fig. 6, while $r1$ (resp. $r2$, $r3$) is the node from the Version graph tracing the versions of relation $R1$ (resp. $R2$, $R3$).

$Rev.0$ stands for the initial version of all the elements, should they be nodes or relations.

When several modifications have occurred, the VersionGraph becomes more complex and several revisions appear. These revisions can be traversed with different points of view, depending on the user needs to trace nodes or elements, as illustrated by Fig. 7.

3. Problem Statement

In this paper, we focus on malicious human actions in the form of rogue behaviors. Such activities are known to lead to significant financial losses.

These behaviors can be detected as they differ from regular *honest* ones.

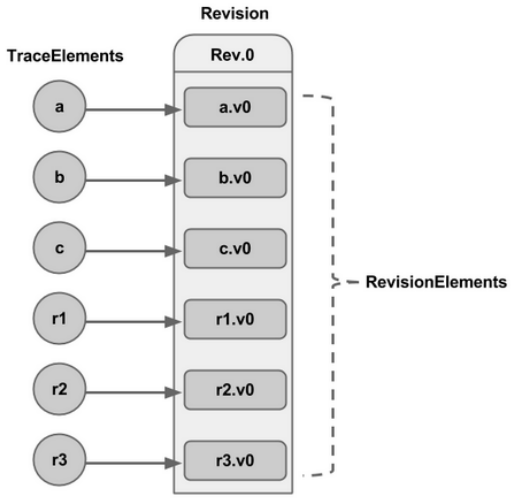

Figure 5: Version Graph Model

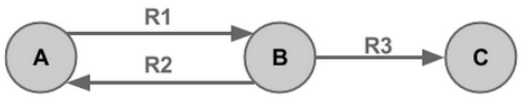

Figure 6: Data Graph Model

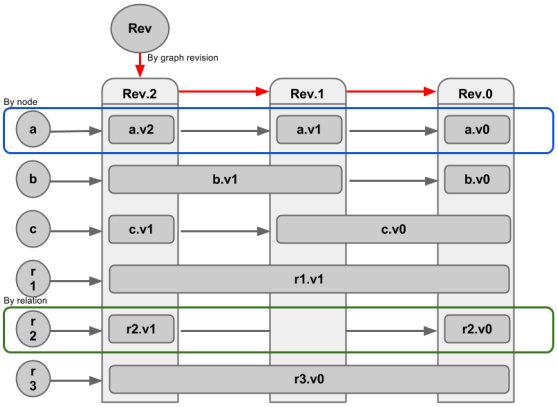

Figure 7: Points of view in VersionGraph

Fraud detection is thus often considered as a subtopic of anomaly detection to detect irregular activities. Anomaly detection has been widely studied [11].

In this work, we claim that fraud data are often fundamentally graphs. Fraudsters act in an indirect manner in order to remain hidden as long as possible. Some of them may rely on vehicles or fake corporations. Most of them operate in groups so as to introduce a complexity and a volume of interactions that prevent the fraud from being visible.

However, when focusing on graphs, few works have been produced [12]. In such problems, fraudsters try to remain hidden by acting in groups so that no action is then a direct one. Intermediate layers are meant to hide the association of fraudsters. It should be noted that, if the size of the group is too small, then the fraud can more easily be detected, but if the size is too big, there is a higher probability of an error occurring, whether it be caused by coincidence or caused by a weak link in the circle of members. The size of the group being organized for the fraud ranges from two to several.

In our work, we focus on fraud rings [13] defined as follows.

Definition 6 (Fraud Ring). *Given a graph G, a fraud ring (also known as fraud cycle) can be defined as a subgraph $F \subseteq G$ where there exist at least two nodes $n_1, n_2 \in F$ that are indirectly connected in both directions in a period of time, with $n_1 - [*] -> n_2$ and $n_2 - [*] -> n_1$.*

A fraud ring betrays illegal links between the nodes n_1 and n_2.

Many problems are based on fraud rings, from corruption, insurance and bank fraud, to shell companies, etc. Even in human resources management, some people can take advantage of friendships or close relationships to unfairly attain positions of power. Many works and methods have addressed this problem but fraudsters have built innovative practices that are not easily discovered by existing tools.

For instance, for insurance fraud, some people claim millions of dollars after declaring fake accidents, fake passengers and fake witnesses. Fig. 8 shows how this appears on a graph.

Fraud rings can be retrieved by connected analysis when for instance one person acts once as a driver and then twice as a witness or passenger in another

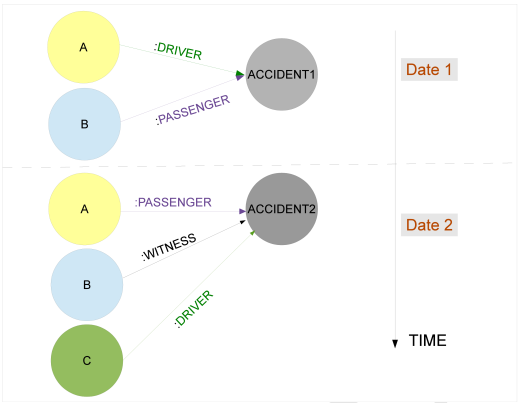

Figure 8: Fraud Ring - fake car accidents, passengers and witnesses

car accident, as shown by SJ Moody and illustrated by Fig. 9.

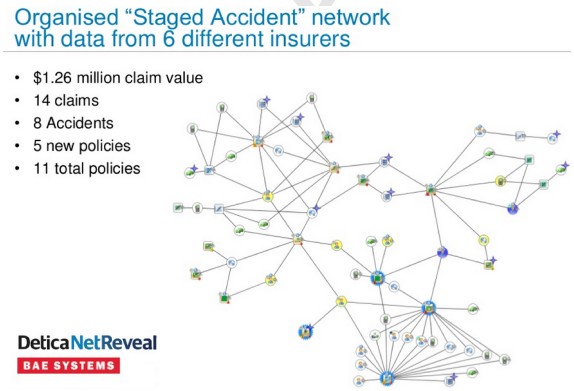

Figure 9: A Fraud Ring Example

3.1. Use Case

We focus on the bank fraud detection as discussed in [14]. In such frauds, a circle of individuals share some legal documents and create accounts. The credit lines and accounts are used, and gradually merged with unsecured lines.

As depicted in [14]: "One day the ring *busts out*, coordinating their activity, maxing out all of their credit lines, and disappearing. Sometimes fraudsters will go a step further and bring all of their balances to zero using fake checks

immediately before the prior step, doubling the damage".

For this purpose, link analysis and discrete data analysis have been proposed and applied, for instance using the Neo4j tools or in [15, 16]. For instance it can help to retrieve the "account holders who share more than one piece of legitimate contact information" which is very easy when data are represented using graph structures.

3.2. Problem

In this paper, we focus on the exploitation of the relations, of the history and of the labels contained in the graph for retrieving fraud rings. We claim that:

- history is a key feature for discovering such rings. Time is indeed important and fraudsters are playing with delays in order to gradually root their actions. However, time is not always represented in the systems used to store the information, which prevents organizations from efficiently wiping out fraud and corruption.
- fuzziness is required as such patterns are approximate.
- many algorithms have been designed, but their implementation on real world problems and engines is not always tractable. We thus consider the use of adapted tools with the capabilities to express such queries in declarative languages.

The next section introduces various means to address these points.

4. First Attempt

As shown above, fraud detection relies on the discovery of graph patterns in the successive states of the data. This section relies on a classic example of bank fraud as depicted in [14].

Banks and Insurance companies lose billions of dollars every year to fraud. Traditional methods of fraud detection play an important role in minimizing

these losses. However increasingly sophisticated fraudsters have developed a variety of ways to elude discovery, both by working together, and by leveraging various other means of constructing false identities.

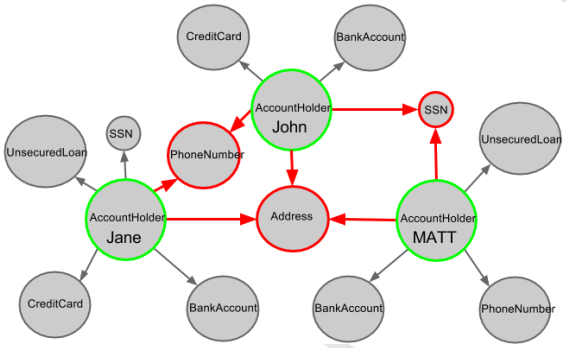

Figure 10: Fraud dataset

The dataset used in this section is defined in Fig. 10. Green circled nodes represent fraudsters and red circled nodes represent the connection points between these people.

In this section, we first introduce a standard pattern matching resolution to detect fraud rings in a graph. This kind of resolution works well when all the data are available (which means that no valuable information for resolution has been erased or replaced). To handle this problem, there are two main ways: (i) by embedding the history within the operational model. This creates difficulties as the patterns for fraud detection must then be written in an adhoc manner, depending on the specific model; (ii) by considering a generic model for describing the history that is compatible with pattern matching. This allows the user to write generic pattern matching queries for retrieving the fraud rings.

This latter case is explored by presenting how the Mnemosyne model can help.

4.1. Pattern Matching Resolution

4.1.1. Classic Resolution

The challenge is to determine if there is a ring in the dataset, and if so, what is its size and what is the finacial risk it represents. A classical solving resolution algorithm looks like the following Listing 1.

Listing 1: Classical pattern resolution

```
1  MATCH       (accountHolder:AccountHolder)-[]->(contactInformation)
2  WITH        contactInformation,
3              count(accountHolder) AS RingSize
4  MATCH       (contactInformation)<-[]-(accountHolder),
5              (accountHolder)-[r:HAS_CREDITCARD|HAS_UNSECUREDLOAN]->(
                   unsecuredAccount)
6  WITH        collect(DISTINCT accountHolder.UniqueId) AS AccountHolders,
7              contactInformation, RingSize,
8              SUM(CASE type(r)
9                  WHEN 'HAS_CREDITCARD' THEN unsecuredAccount.Limit
10                 WHEN 'HAS_UNSECUREDLOAN' THEN unsecuredAccount.Balance
11                 ELSE 0
12             END) as FinancialRisk
13 WHERE       RingSize > 1
14 RETURN      AccountHolders AS FraudRing,
15             labels(contactInformation) AS ContactType,
16             RingSize,
17             round(FinancialRisk) as FinancialRisk
18 ORDER BY    FinancialRisk DESC
```

First, it finds all the account holders that have at least one piece of information in common (line 1 to 5) and then it calculates the ring size (line 3) and the induced financial risk (lines 8 to 12).

The result of the execution of Listing 1 is shown in Fig. 11.

Such queries rely on the so called Cypher language which runs as a declarative language over NoSQL graph databases. These queries include the following clauses[1]:

[1] See `https://neo4j.com/docs/cypher-refcard/current/` for a complete reference card

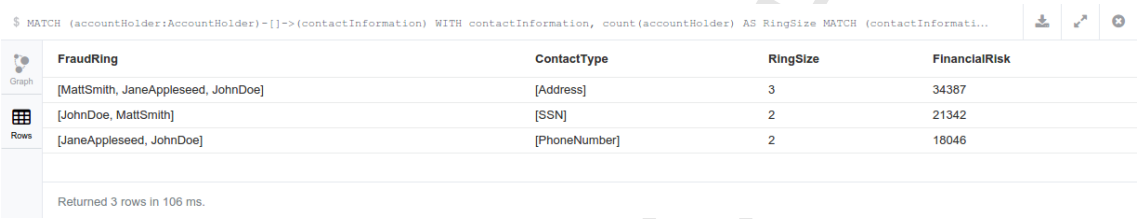

Figure 11: Classical pattern resolution results

- START: Starting points in the graph, obtained via index lookups or by element IDs;
- MATCH: The graph pattern to match, bound to the starting points in START;
- WHERE: Filtering criteria;
- RETURN: What to return;
- CREATE: Creates nodes and relationships;
- DELETE: Removes nodes, relationships and properties;
- SET: Set values to properties;
- FOREACH: Performs updating actions once per element in a list;
- WITH: Divides a query into multiple, distinct parts.

For instance, the following query returns all the nodes of type accountholder:

```
MATCH (A:ACCOUNTHOLDER)
RETURN A
```

The following query returns all the relations between two accountholders, whatever the type of their relation:

```
MATCH (A:ACCOUNTHOLDER)-[*]->(B:ACCOUNTHOLDER)
RETURN A,B
```

4.1.2. Resolution Generalization

In fact, resolution of these kind of problems can be as generalized as answering this question: "Do some people share some information?". Expressing the query for responding to such a question can be more generically answered with queries like:

```
MATCH (A:ACCOUNTHOLDER)-[*]->(B:ACCOUNTHOLDER)-[*]->(C:ACCOUNTHOLDER)
WITH count(accountHolder) AS RingSize
RETURN A,B,C,RingSize
```

We can then apply this algorithm in the same manner to other graphs that are not in the same business area.

4.2. Temporal Pattern Matching Resolution

In this section, temporal queries rely on the Mnemosyne model presented in Section 2.

4.2.1. Fraud Ring: Temporal Cheating

It should be noted that expressing pattern matching defined in Section 4.1 can be done on existing insurance/bank data as such applications are meant to store and trace the history. However, this is not always the case. Fig. 12 shows how, if history was not managed in such systems, fraudsters can cheat to stay hidden by creating, updating, and deleting information over time.

In this case, the above algorithms will not detect any fraud ring as they cannot manage history. To help organizations to find frauds, it is thus necessary to record historical tracks.

4.2.2. Mnemosyne Extension

The Mnemosyne system presented above offers an efficient manner to trace the history within graph data. If a node or a relationship is impacted by an operation, should it be an update, delete, or insert operation, then this is recorded in another graph called VersionGraph which handles the history of each node or relation.

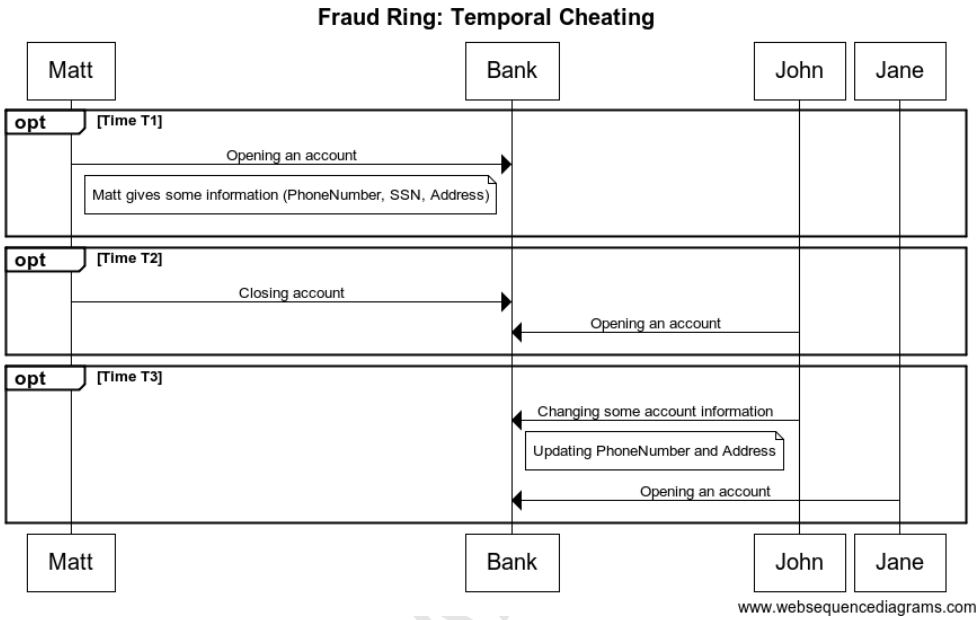

Figure 12: Example of temporal cheating applied to current dataset

It should be noted that the VersionGraph manages the history in the same manner no matter what the business area may be. The Mnemosyne system represents the history in the same way despite the heterogeneity of the DataGraph (in terms of types as well as structures of data). This is due to the generic nature offered by the system which does not focus on the semantics of data and which provides a strong dissociation between the data (in the sense of information) and the data structure.

However, in this model, the representation does not materialize the links between the impacted neighborhoods. For instance, if a relation is added, then the incoming and outgoing nodes are changed in the graph. All these operations will thus be traced in the Mnemosyne VersionGraph, but no link is traced between the elements (to manage performance optimization).

We thus propose to materialize these links in order to speed up the process. If a relation is updated, then the Mnemosyne will materialize (by a relationship in the VersionGraph) the impacted nodes and relations.

In our system, this materialization is triggered by any change on the data, following algorithm 1.

Algorithm 1: Materializing Change Impacts

Data: VG_i: Input Version Graph for Graph G, $R \in G$ revised relation

Result: VG_r : Resulting VersionGraph

$VG_r \leftarrow VG_i$;

foreach *Input Node* $N_i \in G$ *such that* $N_i - R- > *$ **do**
  CREATE materialized relation $n_i- > r$ in VG_r

foreach *Output Node* $N_O \in G$ *such that* $* - R- > N_O$ **do**
  CREATE materialized relation $r- > n_o$ in VG_r

Fig. 13 presents an example of such a situation, with the blue relationships representing the materialized relationships which are added.

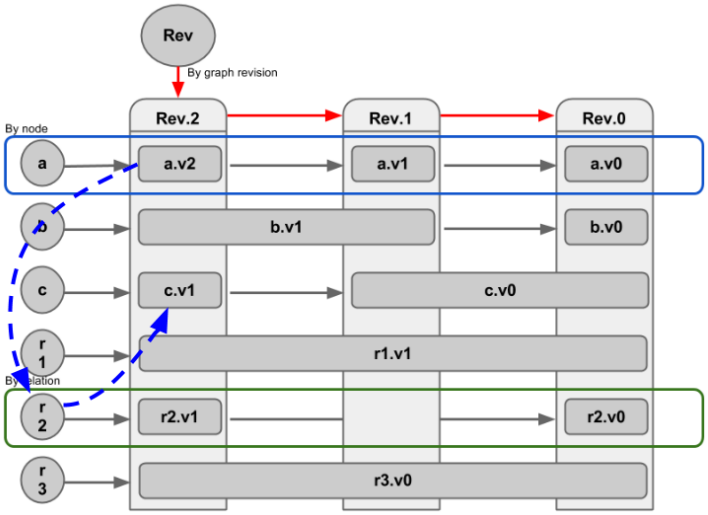

Figure 13: Materializing Elements impacted by an operation in the VersionGraph

4.2.3. Resolution

Resolving such patterns in NoSQL graph databases requires the definition of original queries. Such queries are built to navigate in the graph and historical graph (Mnemosyne VersionGraph) in order to retrieve the fraud rings. The efficiency of the system relies on the NoSQL engine.

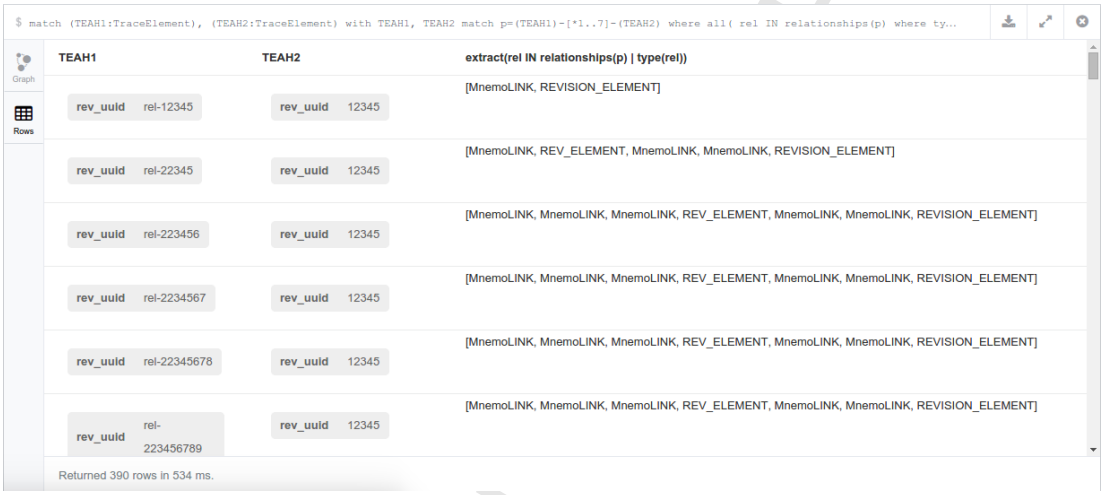

Figure 14: Mnemosing Ring: Exploiting Historical Revisions

Listing 2 shows how to query all the links from the history in order to retrieve the potential fraud ring.

Listing 2: Exploiting Historical Revisions

```
match (TEAH1:TraceElement), (TEAH2:TraceElement)
with TEAH1, TEAH2
match p=(TEAH1)-[*1..7]-(TEAH2)
where all( rel IN relationships(p) where type(rel) <> 'REVISION')
return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel))
```

As shown in this Cypher query, all the elements impacted on the graph can easily be retrieved by exploiting the materialized links of the history from the extension we propose. The result of this query is displayed in figure 14.

However, we claim that the patterns to be found cannot be defined in a crisp manner and propose below an extension to fuzzy historical patterns.

5. Fuzzy Historical Pattern Matching Resolution

In many cases, the parameters of the patterns cannot be defined in a strict manner. For instance, the minimum and maximum size of the fraud ring are

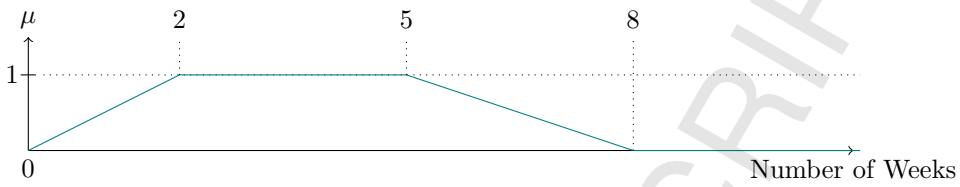

Figure 15: Trapezoidal Membership Function of the Fuzzy Set SeveralWeeks

fuzzy parameters. In the same manner, the time delays between the fraudsters' actions are fuzzy.

We thus propose using fuzzy clauses in the pattern matching queries.

5.1. Defining fuzzy clauses

Fuzzy queries have been extensively studied in the literature, mainly on relational databases and more recently on NoSQL gaph databases. The propositions can either address classical crisp databases ([17]) or fuzzy graphs [18, 19].

Temporal fuzzy clauses can be defined either by considering the fuzzy Allen intervals or by considering user defined clauses. User can define temporal graduation such as "some days" and "several weeks". Fig. 15 draws a membership function defining "several weeks". This membership function defines a degree of membership of a number of weeks to the term "several weeks". For instance, 3 weeks has a degree of membership of 1 but more than 5 weeks as a lower degree of membership (may be months could be a better term?).

An implementation of the membership function of Figure 15 has been made and named "fuzzyWeeks". This membership function is used in the query of Listing 3.

Listing 3: Fuzzy Pattern Matching on Fuzzy Weeks

```
match (TEAH1:TraceElement), (TEAH2:TraceElement)
with TEAH1, TEAH2
match p=(TEAH1)-[*1..7]-(TEAH2)
where all( rel IN relationships(p) where type(rel) <> 'REVISION') and ↩
    severalWeeks(p) > 0.7
return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel))
```

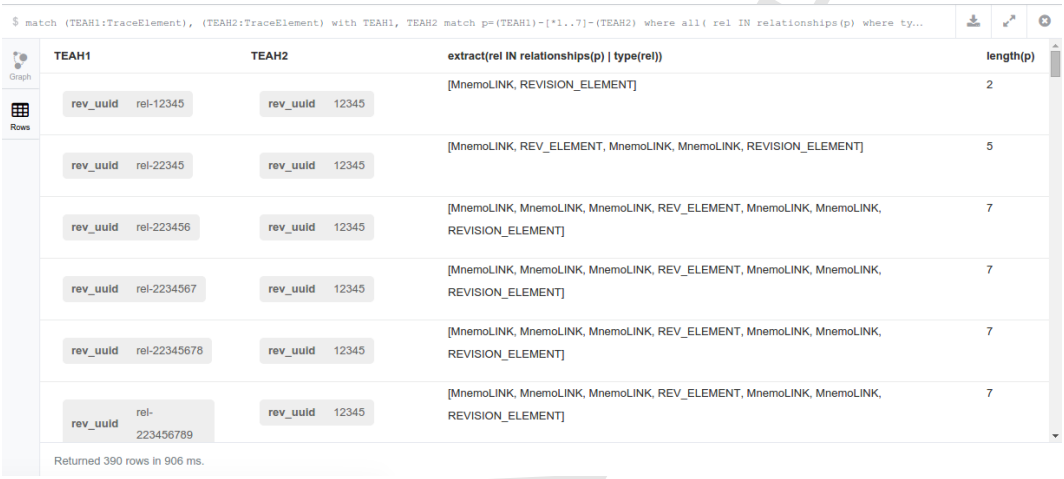

Figure 16: Mnemosing Ring : Exploiting the Length

Fuzzy pattern matching can also be applied on relationships. Rings can be characterized in the same way as the length of the paths forming the relations. Finding the size of a path can be accomplished as shown is Listing 4; the ring ranges within fuzzy bounds, as shown on Listing 5; the result is displayed in Figure 16.

Listing 4: Exploiting the Length

```
match (TEAH1:TraceElement), (TEAH2:TraceElement)
with TEAH1, TEAH2
match p=(TEAH1)-[*1..7]-(TEAH2)
where all( rel IN relationships(p) where type(rel) <> 'REVISION')
return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel)),
    length(p)
```

Listing 5: Fuzzy Pattern Matching on Fuzzy Lengths

```
match (TEAH1:TraceElement), (TEAH2:TraceElement)
with TEAH1, TEAH2
match p=(TEAH1)-[*1..7]-(TEAH2)
```

```
where all( rel IN relationships(p) where type(rel) <> 'REVISION')
return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel)),↩
    fuzzyDist(p)
```

5.2. Generalizing Fuzzy Clause to Handle any Rogue Behavior

The main drawback of the solution presented above is that for each specific need a new function has to be developed (e.g. fuzzyWeek, fuzzyDistance). This means that for each fuzzy clause, it is necessary to define a hard coded membership function.

This lack of genericity has highlighted the need for improvements that can adapt the detection to each rogue behavior.

To do so, it is necessary to offer the user the ability (i) to propose high level definition of fuzzy terms (ii) to use this definitions inside a declarative graph pattern matching query language.

5.2.1. Fuzzy Linguistic Terms Definition in Behavioral Anomalies Detection

Searching for anomalies detection means searching for links between some data. There are different kinds of links such as relational links (is there some relation, may it be direct or indirect, that can rely this person A with person B?), temporal links (have event e1 and event e2 happened in the same temporal frame?), geographical links, etc. Each of this dimension has its own vocabulary and its own graduation.

What has been previously proposed in this article is a high level pattern matching system that uses the genericity of the underlying versioning system to detect anomalies in suspicious patterns. This approach has been enriched in the beginning of this section with the use of fuzzy functions that account for the fact that parameters cannot always be defined in a strict manner, especially when searching for behavioral anomalies.

In the remainder of this section, the goal is to define and provide a system that offers to the user the ability to define his/her own terms definition according to his/her functional specific domain.

We propose the use of linguistic terms and more generally linguistic variables to express domain concerns. Every term will be associated to a membership function offering the ability to define fuzzy linguistic terms and variables.

Definition 7 (Fuzzy linguistic variables). *Fuzzy linguistic variables can be defined using fuzzy sets. A fuzzy linguistic variable V is defined as a quadruple of the form V=(X, U, T, MF), where X is the name of V, U is the domain (universe) of V, T represents the set of fuzzy subsets defined in U, and MF represents the membership functions that characterize each fuzzy subset defined in T.*

5.2.2. Fuzzy linguistic terms grammar

To express linguistic terms, we rely on our previous defined work on fuzzy logic and our previous implementations based on functional programming.

In functional programming, a common approach is to model parsers as functions and to define higher-order functions (also called combinators) that implement grammar constructions such as sequencing, choice, and repetition. As explained in [20], the basic idea dates back to 1970s [21] and has become popular since the 1980s in a variety of functional programming languages [22, 23, 24].

The parsing expression grammar we chose to express fuzzy linguistic variale is defined in Listing 6.

Listing 6: Fuzzy DSL syntax PEG

```
1
2   def number: Parser[Double] = """-?\d+(\.\d*)?"""
3   def point: Parser[Point] = "(" ~> number ~ "," ~ number <~ ")"
4   def functionName: Parser[String] = """trian|trape|gauss|gbell|sigm↩
        """"
5   def parameters = rep1(point) | rep1(number)
6   def memberfunction = functionName.? ~ parameters
7   def termName = """[a-zA-Z]\w*"""
8   def term =  "TERM" ~> termName ~ ":=" ~ memberfunction <~ ";"
9   def variableName = """[a-zA-Z]\w*"""
10  def fuzzyVariable = "FUZZIFY" ~> variableName ~ rep1(term) <~ "↩
        END_FUZZIFY"
```

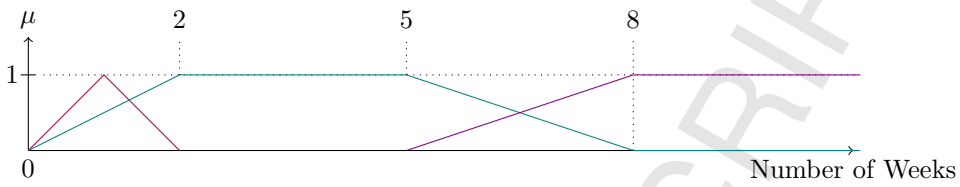

Figure 17: Representation of Listing7

Listing 7 and 8 gives the definition of two linguistic terms "temporal" and "distance" that will be used in the next section.

Listing 7: Fuzzy linguistic definition of week

```
1
2  FUZZIFY week
3          TERM one:=  trian 0 1 2;
4          TERM several:= trape 0 2 5 8;
5          TERM many:= (5,0) (8,1);
6  END_FUZZIFY
```

Listing 8: Fuzzy linguistic definition of depth

```
1  FUZZIFY depth
2          TERM low:=  trian 0 1 2;
3          TERM middle:= trian 1 3 5;
4          TERM high:= (3,0) (5,1);
5  END_FUZZIFY
```

5.2.3. Using a Rogue Definition

The Cypher language does not allow the use of fuzzy definitions. To do so, it is necessary to extend the language. Discussion about extending a graph pattern matching declarative language has been done in [17]. Cypherf, standing for Cypher fuzzy, is an extension of the cypher language that allows the user to use fuzzy linguistic terms such as those defined in the previous section. The available functions are listed bellow:

- Fuzzy(μ_f, value): returns the degree of membership to μ_f function
  - μ_f is expressed as a String that describes the membership function with the Fuzzy DSL
  - value is expressed as a Double
- FuzzyLT(fuzzyVariable, value): returns a collection that contains for every fuzzy term of the fuzzy linguistic variable two properties: the name of the term and the degree of membership of "value" to the term. For instance, for a value X and a fuzzy linguistic variable $serveralWeeks = (Week, [0, \infty], \{one, several, many\}, \{\mu_{one}, \mu_{several}, \mu_{many}\})$ the result will be:

```
1 {
2     {name:"one",degree:μ_one(x)},
3     {name:"several",degree:μ_several(x)},
4     {name:"many", degree:μ_many(x)}
5 }
```

  - fuzzyVariable: is expressed as a String that defines the fuzzy variable. This definition is composed of the name of the fuzzy variable and a set of fuzzy terms. Every fuzzy term is defined by its name and its membership function
  - value is expressed as a Double
- TNorms(tnormName, expression1, expression2): applies a TNorms of name "tnormName" on expression1 and expression2
- TCoNorms(type, expression1, expression2): same as TNorms but this time for t-conorms.

Two examples are provided in Listings 9 and 10. Listing 9 returns the couples of nodes (TEAH1 and TEAH2) and the relation characterizing with the highest score (HEAD) their link. TEAH1 and TEAH2 are linked with p (line 3). For

this purpose, it computes the membership degree for every term of the fuzzy variable *depth* using the fuzzyLT function.

Listing 10 returns the same type of result but only if the membership degree is greater than 0.7 (line 6) which prevents unrelevant results to be displayed.

Listing 9: Fuzzy Pattern Matching on Fuzzy Lengths with fuzzy linguistic terms

```
1 match (TEAH1:TraceElement), (TEAH2:TraceElement)
2 with TEAH1, TEAH2
3 match p=(TEAH1)-[*1..7]-(TEAH2)
4 where all( rel IN relationships(p) where type(rel) <> 'REVISION')
5
6 return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel)), ↩
    head(fuzzyLT("/tmp/depth.fl", size(p)))
```

Listing 10: Fuzzy Pattern Matching on Fuzzy Lengths with selection on fuzzy linguistic terms degree

```
1 MATCH (TEAH1:TraceElement), (TEAH2:TraceElement)
2 WITH TEAH1, TEAH2
3 MATCH p=(TEAH1)-[*1..7]-(TEAH2)
4 WITH p, head(fuzzyLT("/tmp/distance. fl", size(p))) as dist
5 WHERE all( rel IN relationships(p) where type(rel) <> 'REVISION')
6 AND dist.degree > 0.7
7 return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel)),
```

In this section fuzzy historical pattern matching has been introduced. Both a first implementation and a generalization have been done with some running examples. A grammar has been presented and the relevance of the use of fuzzy historical pattern matching in fraud ring detection has been addressed.

The next section presents the related work on frauds, fuzzy and temporal graph data.

6. Related Work

Fraud detection has been a hot topic for many years as its impact is extensive in many fields and sectors such as industry, finance, bank, insurance, organized

and corporate crime, health care, etc.

Rogue behaviors can be detected using several methods. The most common ones from the literature are based on statistical methods, pattern recognition, expert systems and machine learning [25]. These techniques can be applied on several kinds of data such as time series or transactional data, depending on the type of context being considered: computer intrusion, credit card fraud, telecommunication fraud, etc. However there are only few works on graph data.

6.1. Frauds and Detection

Rogue behaviors and frauds are defined in the literature in various ways. The main characteristics are related to the fact that such behaviors refer to criminal deceptions through intentional human practices and interactions aiming at dishonest advantages.

Fraud detection relies on methods for anomaly detection. Such anomalies can be seen as outliers and are thus retrieved by comparing regular and fraudulent situations.

This comparison can be achieved using supervised learning [26] or cases can be separated using statistical and clustering methods [27].

Algorithms have been defined for several types of data sources. Transactional and stream data have been extensively addressed, especially for dealing with bank and finance frauds and can be regarded as outlier detection in stream data [28].

As frauds are generated by humans, their characteristics are often impossible to define in a crisp manner. For this reason, many works have addressed the use of soft computing and formal representations of imprecision and uncertainty.

[29] reviews the main methods from intelligent systems that can help, from fuzzy neural networks to genetic algorithms. [30] proposes the use of fuzzy clustering in the particular context of corporate fraud. [31] focuses on auction frauds and proposes the use of fuzzy rules that are optimized by genetic algorithms.

Regarding graph-based data, outlier detection has been less addressed as it is much more complex. Recent works have addressed the detection of outliers in graph streams [32] and temporal data and graphs [33].

However, no work has addressed fuzzy fraud ring detection from temporal graph data.

In order to detect frauds from such graph data, graph pattern must be considered.

6.2. Graph Pattern Matching

Graph pattern matching is distinguished from graph mining where frequent subgraphs are searched for [34, 35].

[36] addresses several topics within the framework of the approximate graph matching problem, as for instance the computation of the distance between two graphs.

Temporal queries have been studied since the beginning of databases, in relational databases [37], data warehouses [38] and Web data [39, 40]. However, very few works have been proposed to represent and query history in graph-oriented NoSQL databases [41].

Many works on temporal queries are based on the Allen interval algebra [42] recalled in Fig. 18. In this framework, $A = [a^-, a^+]$ and $B = [b^-, b^+]$ are two intervals of dates (for instance, $A_1 = [January2013, March2013]$ and $B_1 = [February2013, June2013]$) and some predicates are defined on A and B which can hold up or not depending on tests over the boundaries a^-, a^+, b^-, b^+ (for instance $before(A_1, B_1)$ does not hold as $March2013 > February2013$).

Temporal SQL has been extensively studied, and has been fuzzified [43] to describe for instance that the event B occurs *long before* event A, or that event A occurs *before or approximately at the same date as* the event B.

For all the above-mentioned topics, graphs can be queried through languages. Some of them have been studied in the literature. Some languages have been proposed by scientists and some other ones have been issued by the editors.

ALLEN'S TEMPORAL INTERVAL RELATIONS BETWEEN INTERVALS
$A = [a^-, a^+]$ AND $B = [b^-, b^+]$

Name	Definition		
before	$b(A,B)$	$\equiv$	$a^+ < b^-$
overlaps	$o(A,B)$	$\equiv$	$a^- < b^-$ and $b^- < a^+$ and $a^+ < b^+$
during	$d(A,B)$	$\equiv$	$b^- < a^-$ and $a^+ < b^+$
meets	$m(A,B)$	$\equiv$	$a^+ = b^-$
starts	$s(A,B)$	$\equiv$	$a^- = b^-$ and $a^+ < b^+$
finishes	$f(A,B)$	$\equiv$	$a^+ = b^+$ and $b^- < a^-$
equals	$e(A,B)$	$\equiv$	$a^- = b^-$ and $a^+ = b^+$
after	$bi(A,B)$	$\equiv$	$b(B,A)$
overlapped-by	$oi(A,B)$	$\equiv$	$o(B,A)$
contains	$di(A,B)$	$\equiv$	$d(B,A)$
met–by	$mi(A,B)$	$\equiv$	$m(B,A)$
started–by	$si(A,B)$	$\equiv$	$s(B,A)$
finished–by	$fi(A,B)$	$\equiv$	$f(B,A)$

Figure 18: Allen's Temporal Interval Relations Between Intervals

[44] proposes a survey of all the languages defined over the last 25 years, where subgraph matching appears to be one of the most powerful and necessary query, including approximate matching.

[45] proposes a propositional dynamic logic that extends several graph database languages.

[46] introduces *GraphQL*, a graph algebra capable of taking into account nodes, relations, and attributes on both nodes and relations. In this language, the authors define the so-called *graph pattern* as a pair $\mathcal{P} = (\mathcal{M}, \mathcal{F})$ where $\mathcal{M}$ is a graph motif and $\mathcal{F}$ is a predicate on the attributes of the motif. The algebra contains several operations, including selection, cartesian product, join and composition. It also demonstrates that their algebra is contained into Datalog, thus allowing for rewriting their queries in Datalog.

7. Conclusion and Further Work

Rogue behaviors often amount to human abnormal interactions also called *fraud rings*. We thus discuss in this paper how property graphs can help for rogue behavior detection by focusing on interactions. We focus on NoSQL graph databases that offer a useful framework to deal with large graph-based datasets. This paper extends [47] which has introduced a first attempt to NoSQL graph queries for rogue detection. It puts the emphasis on the use of fuzzy

queries in order to better address the approximate definitions inherent in such human activities. This paper introduces the use of a Domain Specific Language and a declarative fuzzy query language (Cypherf) to better address and resolve sophisticated cases.

The use of NoSQL graph databases is a key element of our work. We especially claim that representing the successive versions of the graph data allows to better retrieve the chains of successive transactions that represent a fraud. For this purpose, we consider using the Mnemosyne system that has been extended for materializing temporal relations between objects. This allows us to directly apply pattern matching on the graph for retrieving the fraud rings.

Further works aim at integrating the contribution from this paper to alert systems, especially on stream data. For instance, it would then be possible to build follow-up systems for organized crime or terrorist rings which have particular behaviors such as "gathering-dispersal-reorganization" dynamics. Such chains appear for example when fraudsters assemble for illicit activities and then suddenly disperse to face either an internal or external threat. Inside threats occur in case of betrayal, the subsequent regrouping often being reorganized in order to destroy the betrayal (settling of scores). External threats may be due to police investigations.

Such a system may then be used for early fraud detection in order to prevent frauds before they occur.

For all these works, the volume and complexity of data and treatments may challenge the limits of the current implementation. For this reason, we aim at investigating the use of in-memory graph processing systems.

References

[1] G. Sadowski, P. Rathle, Fraud detection: Discovering connections with graph databases, in: White Paper - Neo Technology - Graphs are Everywhere, 2014.

[2] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty Years Of Graph Matching In Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence.

[3] J. A. Bondy, Graph Theory With Applications, Elsevier Science Ltd, 1976.

[4] W3C, Allegrograph rdfstore web 3.0's database (2009).

[5] T. T. A. Board, Technology radar, http://thoughtworks.fileburst.com/assets/technology-radar-may-2013.pdf (May 2013).

[6] Z. Sun, B. Shao, H. Wang, J. Li, H. Wang, Efficient subgraph matching on billion node graphs, in: In PVLDB, 2012.

[7] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly, 2013.

[8] R. Angles, C. Gutiérrez, Survey of graph database models, ACM Comput. Surv. 40 (1).

[9] A. Castelltort, A. Laurent, Representing history in graph-oriented nosql databases: A versioning system, in: Proc. of the Int. Conf. on Digital Information Management, 2013.

[10] A. Kosmatopoulos, K. Giannakopoulou, A. N. Papadopoulos, K. Tsichlas, An overview of methods for handling evolving graph sequences, in: I. Karydis, S. Sioutas, P. Triantafillou, D. Tsoumakos (Eds.), Algorithmic Aspects of Cloud Computing - First International Workshop, ALGOCLOUD 2015, Patras, Greece, September 14-15, 2015. Revised Selected Papers, Vol. 9511 of Lecture Notes in Computer Science, Springer, 2015, pp. 181–192. doi:10.1007/978-3-319-29919-8_14.
URL http://dx.doi.org/10.1007/978-3-319-29919-8_14

[11] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (2009) 15:1–15:58. doi:10.1145/1541880.1541882.
URL http://doi.acm.org/10.1145/1541880.1541882

[12] C. C. Noble, D. J. Cook, Graph-based anomaly detection, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03, ACM, 2003, pp. 631–636.

[13] C. Jedrzejek, J. Bak, M. Falkowski, Graph mining for detection of a large class of financial crimes, in: 17th International Conference on Conceptural structure (ICCS'09), 2009.

[14] K. Bastani, in: Bank Fraud Detection, 2014. [link].
URL https://github.com/neo4j-contrib/gists/blob/master/other/BankFraudDetection.adoc

[15] A. Leontjeva, K. Tretyakov, J. Vilo, T. Tamkivi, Fraud detection: Methods of analysis for hypergraph data., in: ASONAM, IEEE Computer Society, 2012, pp. 1060–1064.
URL http://dblp.uni-trier.de/db/conf/asunam/asonam2012.html#LeontjevaTVT12

[16] T. Horvth, T. Grtner, S. Wrobel, Cyclic pattern kernels for predictive graph mining, in: W. Kim, R. Kohavi, J. Gehrke, W. DuMouchel (Eds.), Proceedings of the 10t ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004), August 22-25, 2004, Seattle, WA, USA, ACM Press, New York, NY, USA, 2004, pp. 158–167.
URL http://doi.acm.org/10.1145/1014052.1014072

[17] A. Castelltort, A. Laurent, Fuzzy queries over nosql graph databases: Perspectives for extending the cypher language, in: International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems, Springer, 2014.

[18] O. Pivert, O. Slama, G. Smits, V. Thion, SUGAR: A graph database fuzzy querying system, in: Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, June 1-3, 2016, IEEE, 2016, pp. 1–2. doi:10.1109/RCIS.2016.7549366.
URL http://dx.doi.org/10.1109/RCIS.2016.7549366

[19] O. Pivert, V. Thion, H. Jaudoin, G. Smits, On a fuzzy algebra for querying graph databases, in: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014, IEEE Computer Society, 2014, pp. 748–755. doi:10.1109/ICTAI.2014.116.
URL http://dx.doi.org/10.1109/ICTAI.2014.116

[20] G. Hutton, E. Meijer, Monadic parser combinators.

[21] H. Burge William, Recursive programming techniques.

[22] P. Wadler, How to replace failure by a list of successes a method for exception handling, backtracking, and pattern matching in lazy functional languages, in: Functional Programming Languages and Computer Architecture, Springer, 1985, pp. 113–128.

[23] G. Hutton, Higher-order functions for parsing, J. Funct. Program. 2 (3) (1992) 323–343.

[24] J. Fokker, Functional parsers, in: Advanced functional programming, Springer, 1995, pp. 1–23.

[25] Y. Kou, C.-T. Lu, S. Sirwongwattana, Y.-P. Huang, Survey of fraud detection techniques, in: Networking, Sensing and Control, 2004 IEEE International Conference on, Vol. 2, 2004, pp. 749–754 Vol.2.

[26] P. Dua, S. Bais, Supervised learning methods for fraud detection in healthcare insurance, in: S. Dua, U. R. Acharya, P. Dua (Eds.), Machine Learning in Healthcare Informatics, Vol. 56 of Intelligent Systems Reference Library, Springer, 2014, pp. 261–285. doi:10.1007/978-3-642-40017-9_12.
URL http://dx.doi.org/10.1007/978-3-642-40017-9_12

[27] R. J. Bolton, David, Statistical fraud detection: A review, Statistical Science 17.

[28] L. Cao, Q. Wang, E. A. Rundensteiner, Interactive outlier exploration in big data streams, Proc. VLDB Endow. 7 (13) (2014) 1621–1624. doi:10.14778/2733004.2733045.
URL http://dx.doi.org/10.14778/2733004.2733045

[29] J. Kingdon, Intelligent systems for fraud detection, in: E. Sanchez, T. Shibata, L. Zadeh (Eds.), Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives, Advanced Series in Electrical and Computer Engineering, 1997, pp. 133–154.

[30] M. Lenard, P. Alam, Application of fuzzy logic to fraud detection, in: V. Sugumaran (Ed.), Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications, IGI Global, 2007, pp. 135–139.

[31] C.-H. Yu, S.-J. Lin, Fuzzy rule optimization for online auction frauds detection based on genetic algorithm 13 (2) (2013) 169–182. doi:10.1007/s10660-013-9113-4.
URL http://dx.doi.org/10.1007/s10660-013-9113-4

[32] C. C. Aggarwal, Y. Zhao, P. S. Yu, Outlier detection in graph streams, in: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany, 2011, pp. 399–409. doi:10.1109/ICDE.2011.5767885.
URL http://dx.doi.org/10.1109/ICDE.2011.5767885

[33] M. Gupta, J. Gao, C. C. Aggarwal, J. Han, Outlier detection for temporal data: A survey, IEEE Trans. Knowl. Data Eng. 26 (9) (2014) 2250–2267. doi:10.1109/TKDE.2013.184.
URL http://dx.doi.org/10.1109/TKDE.2013.184

[34] X. Yan, J. Han, gspan: Graph-based substructure pattern mining, in: Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 721–.
URL http://dl.acm.org/citation.cfm?id=844380.844811

[35] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 313–320.
URL http://dl.acm.org/citation.cfm?id=645496.658027

[36] J. Wang, K. Zhang, G. Chirn, The approximate graph matching problem, Pattern Recognition, Proceedings of the 12th International Conference on IAPR 94 (1994) 284–288.

[37] S. S. Chawathe, S. Abiteboul, J. Widom, Representing and querying changes in semistructured data, in: Proc. of the ICDE Conf., 1998.

[38] K. Y. Lee, Y. D. Chung, M. H. Kim, An efficient method for maintaining data cubes incrementally, Inf. Sci. 180 (6) (2010) 928–948. doi:10.1016/j.ins.2009.11.037.
URL http://dx.doi.org/10.1016/j.ins.2009.11.037

[39] S. Auer, H. Herre, A versioning and evolution framework for rdf knowledge bases, in: Proceedings of the 6th international Andrei Ershov memorial conference on Perspectives of systems informatics, PSI'06, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 55–69.
URL http://dl.acm.org/citation.cfm?id=1760700.1760710

[40] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, V. Christophides, High-level change detection in rdf(s) kbs, ACM Trans. Database Syst. 38 (1) (2013) 1:1–1:42. doi:http://dx.doi.org/10.1145/2445583.2445584.
URL http://doi.acm.org/http://dx.doi.org/10.1145/2445583.2445584

[41] U. Khurana, A. Deshpande, Efficient snapshot retrieval over historical graph data, in: C. S. Jensen, C. M. Jermaine, X. Zhou (Eds.), ICDE, IEEE Computer Society, 2013, pp. 997–1008.

[42] P. J. Andrei Krokhin, P. Jonsson, Reasoning about temporal relations:

The tractable subalgebras of allen's interval algebra, Journal of ACM 50(5) (2003) 591–640.

[43] S. Schockaert, M. D. Cock, E. E. Kerre, Fuzzifying allen's temporal interval relations., IEEE T. Fuzzy Systems 16 (2) (2008) 517–533.
URL http://dblp.uni-trier.de/db/journals/tfs/tfs16.html#SchockaertCK08

[44] P. T. Wood, Query languages for graph databases, SIGMOD Rec. 41 (1) (2012) 50–60. doi:10.1145/2206869.2206879.
URL http://doi.acm.org/10.1145/2206869.2206879

[45] R. Angles, P. Barcel, G. Ros, A practical query language for graph dbs, in: 7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW), 2013.

[46] H. He, A. K. Singh, Graphs-at-a-time: Query language and access methods for graph databases, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 405–418. doi:10.1145/1376616.1376660.
URL http://doi.acm.org/10.1145/1376616.1376660

[47] A. Castelltort, A. Laurent, Fuzzy historical graph pattern matching a nosql graph database approach for fraud ring resolution, in: Artificial Intelligence Applications and Innovations, Springer, 2015, pp. 151–167.