Fuzzy Historical Graph Pattern Matching A NoSQL Graph Database Approach for Fraud Ring Resolution

Arnaud Castelltort, Anne Laurent

Fuzzy Historical Graph Pattern Matching A NoSQL Graph Database Approach for Fraud Ring Resolution

Arnaud Castelltort and Anne Laurent

LIRMM - Univ. of Montpellier - CNRS UMR 5506
161 rue Ada, 34 095 Montpellier, France

Abstract. Graphs have been studied and used for many years as they allow to represent in an efficient manner real data such as biological or social data. Graph databases have recently emerged within the NoSQL framework and are implemented in systems like Neo4J, OrientDB, etc. Recent works have shown that the management of history is crucial in such systems. In this paper, we show how such historical graph databases can be queried in order to retrieve fraud rings, also known as fraud cycles. Frauds are indeed often based on sophisticated chains of successive transactions (money, communications, etc.). We thus claim that the indirect link between fraudsters can be retrieved by considering historical NoSQL graph databases. We study how the model of historical NoSQL databases can be extended for better address this goal and we propose the associated queries that have been tested on a synthetical database.

Keywords: NoSQL Graph Databases, Time Information, Fraud Rings.

1 Introduction

Graphs are known to be efficient for representing data in many applications, from linguistics to chemistry and social networks. For instance, such a graph allows to draw in a very intuitive manner the relationships among people and between these people and the organizations they belong to.

Graphs are recognized to play an important role within the pattern recognition field [12], thus being a key technology for retrieving relevant information, as for fraud detection or in social/biological interactions. Techniques and algorithms can be distinguished depending on the fact that they are meant to mine relevant patterns or to retrieve patterns.

If considering frauds (bank and insurance), frauds represent billions of dollars lost by companies every year. For instance more than £52 billions have been lost in UK in 2013 [1]. Fraud can be organised or can be individual. It can impact individuals or organizations (e.g., banks) as in first-party and third-party fraud. Graphs can help for retrieving frauds through the modelization of fraud rings which are hidden within the graph of interactions [24]. A fraud ring is a set of connections between actors. It can be found in many fraud frameworks.

Although graphs have been studied since the very beginning of computer science in the so-called graph theory field [8], 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 [26]. More recently, NoSQL databases have proposed efficient engines devoted to graph databases: GraphDB, Neo4J,... [7] 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 historical pattern matching from NoSQL graph databases. For this purpose, we first recall the basic concepts of NoSQL graph databases, historical queries and graph pattern matching in Section 2. We then detail the problem we address in Section 3, before presenting our contribution and its resolution using the NoSQL Neo4j graph database in Section 4. Section 5 sums up this paper and presents the future work we would like to address.

2 Related Work

2.1 Graphs and NoSQL Graph Databases

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.*

NoSQL graph databases [23] 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. 2 shows a graph and its structure in $(key, values)$ pairs.

Studies have shown that these technologies present good performances, much better than classical relational databases for representing and querying such large graph databases. Their exist several NoSQL graph database engines (OrientDB, Neo4J,HyperGraphDB, etc.) [4]. Neo4J is recognised as being one of the top

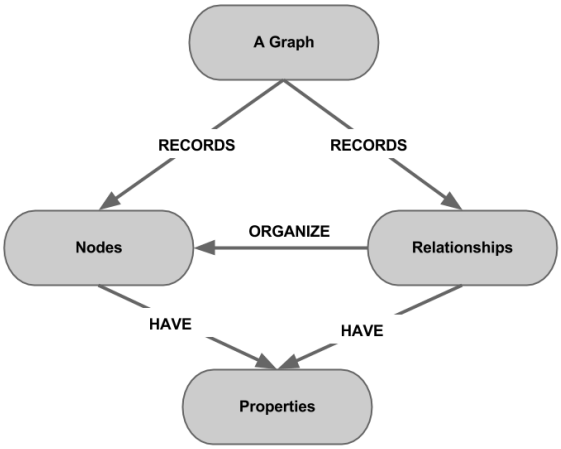

Fig. 1: Labeled Graph

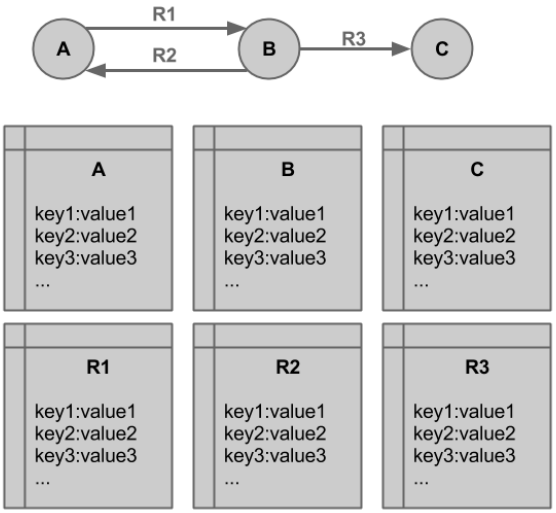

Fig. 2: Properties of Nodes and Relations

ones regarding performance [7]. It has been recently extended for managing the successive versions of the graph database in the Mnemosyne system described below.

2.2 Mnemosyne: an innovative historical data management system

[9] aims at proposing an innovative model of history management in NoSQL graph databases. This contribution is based on three key concepts:

– Use the graph to manage historical data;
– Data historization must be totaly decoupled from the representation of the data in the graph datasource;
– Use generic graph traversals regardless of the graph datasource.

One of the main specificity of the Mnemosyne system is that it uses two graphs :

– DataGraph: the current graph in use
– VersionGraph: a VersionGraph storing the history of different versions of a data graph. The VersionGraph model does not depend on the DataGraph.

Fig. 3 shows an example of a DataGraph and a VersionGraph.

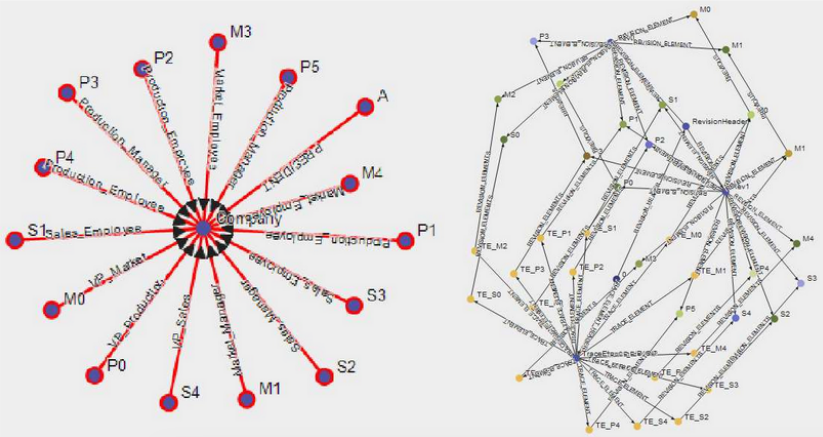

Fig. 3: DataGraph (left) and VersionGraph (right)

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 source.

Fig. 4 illustrates this proposal. In this example, three nodes (a,b,c) and three relations (r1,r2,r3) from Fig. 5 are traced.

When several modifications have occured, the versiongraph becomes more complex and several revisions appear. These revisions can be traversed with different points of view, depending on the fact that the user needs to trace nodes or elements, as illustrated by Fig. 6.

2.3 Pattern Matching and Querying

Graph pattern matching is a very difficult algorithmic problem that has attracted many works that cannot be all recalled here.

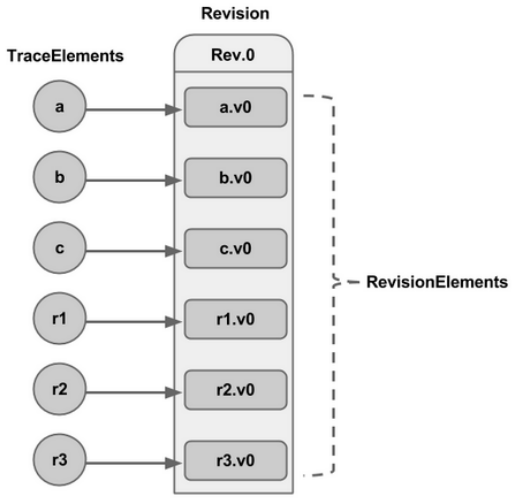

Fig. 4: Version Graph Model

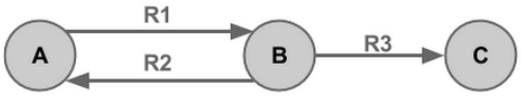

Fig. 5: Data Graph Model

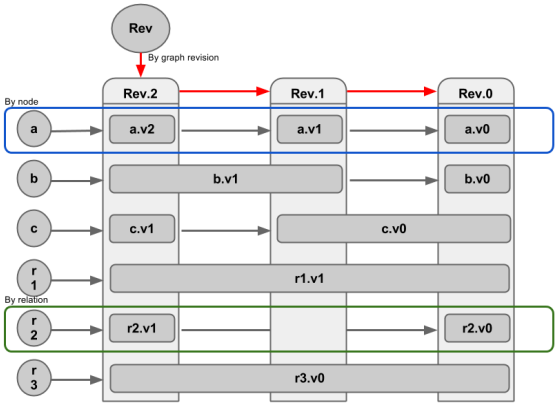

Fig. 6: Points of view in VersionGraph

Graph pattern matching is distinguished from graph mining where frequent subgraphs are searched for [29, 17].

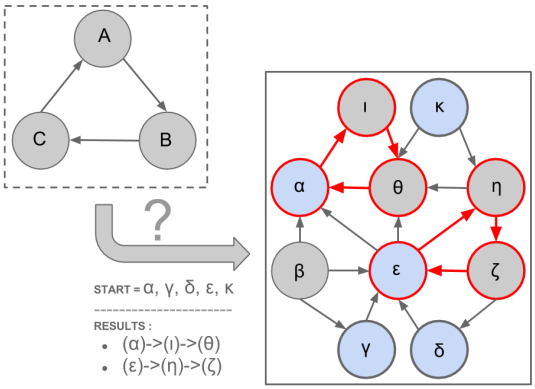

Fig. 7: Pattern Matching

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

Historical queries have been studied since the beginning of databases, in relational databases [11], datawarehouses [18] and Web data [5, 22]. However, very few works have proposed to represent and query history in graph-oriented NoSQL databases [16].

Many works on temporal queries are based on the Allen interval algebra [2] describing the temporal relations between two intervals $A = [a^-, a^+]$ and $B = [b^-, b^+]$.In this framework, A and 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 hold or not depending on some tests over the boundaries a^-, a^+, b^-, b^+ (for instance $before(A_1, B_1)$ does not hold as $March2013 \not< February2013$). Temporal SQL has been extensively studied, and has been fuzzified [25] as for instance to describe 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.

[28] 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.

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

[13] 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, composition and demonstrate that their algebra is contained into Datalog, thus allowing for rewriting their queries in Datalog.

We show below how these works can be used for fuzzy pattern matching in the framework of fraud ring detection and what their limits are.

3 Problem Statement

3.1 Preliminary statements

Fraud detection is often considered as a subtopic of anomaly detection. Anomaly detection has been widely studied [10]. Fraud data are often intrinsically graphs. However, when focusing on graphs, few works have been done [21]. In such problems, fraudsters try to stay 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 discovered, but if the size is too big, then there are more probabilities that some problem occurs, should it be caused by coincidence or caused by a weak link in the circle of persons. The size of the group being organized for the fraud ranges from 2 to several.

In our work, we focus on fraud rings [15] defined as follows.

Definition 4 (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, betraying 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 managing human resources, it can be the case that a few people make benefit their brotherhood from unfair positions. Many works and methods have addressed this problem but fraudsters have built innovative manners that are not easily discovered by existing tools.

For instance, for insurance fraud, some people are claiming 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 people acts once as a driver and then twice as a witness or passenger in another car accident, as shown by [20] and illustrated by Fig. 9.

3.2 Use Case

We focus on the bank fraud detection as depicted in [6]. In such frauds, a circle of persons share some legal documents and create accounts. The credit lines and accounts are used, and gradually merged with unsecured lines.

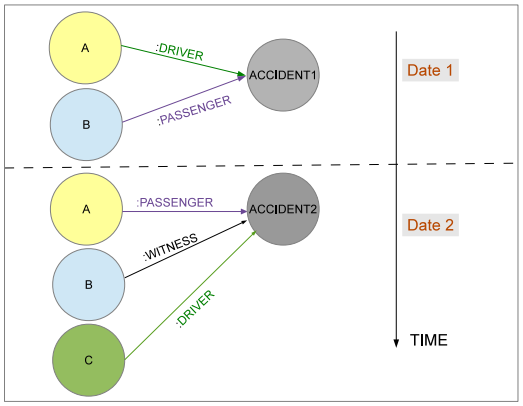

Fig. 8: Fraud Ring - fake car accidents, passengers and witnesses

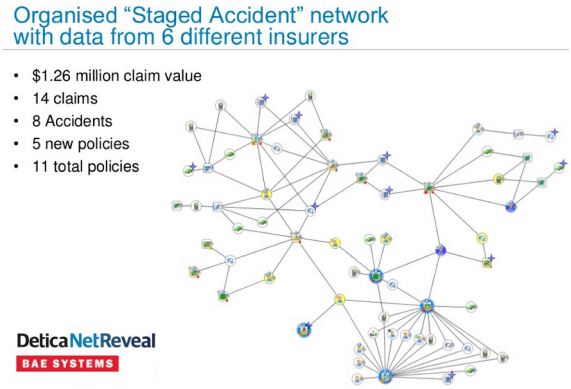

Fig. 9: A Fraud Ring Example [20]

As depicted in [6]: "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 has been proposed and applied, for instance using the Neo4j tools or in [19, 14]. 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.

3.3 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 Resolution and Experiments

As shown above, fraud detection relies on the discovery of graph patterns in the successive states of the data. This section relies on a classical example of bank fraud.

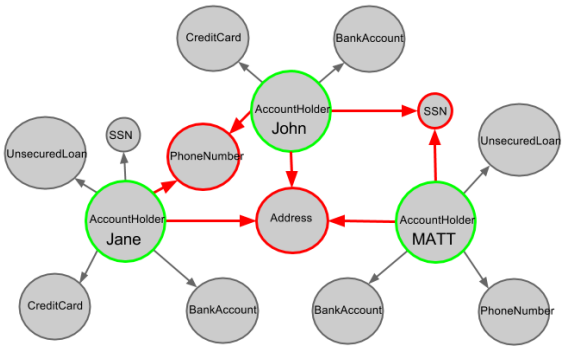

Fig. 10: Fraud dataset

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 persons.

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 have been erased or replaced). To handle this problem, there are two main ways:

– either 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

– either by considering a generic model for describing the history that is compatible with pattern matching. This allows to write generic pattern matching queries for retrieving the fraud rings.

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

Finaly, to go further, we show that fuzziness can help to address approximate pattern matching.

4.1 Pattern Matching Resolution

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 looks like Listing 1.1:

Listing 1.1: 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
```

That is to say, finding all the account holders that have at least one piece of information in common (line 1 to 5) and then calculate the ring size (line 3) and the induced financial risk (lines 8 to 12).

The result of the execution of Listing 1.1 declarative request is shown in Fig. 11.

Resolution Generalisation In fact, resolution of this kind of problems can be generalized as answering this question: "Do some people share some information?". Expressing the query for answering such a question can be more generic-aly answered with queries like:

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

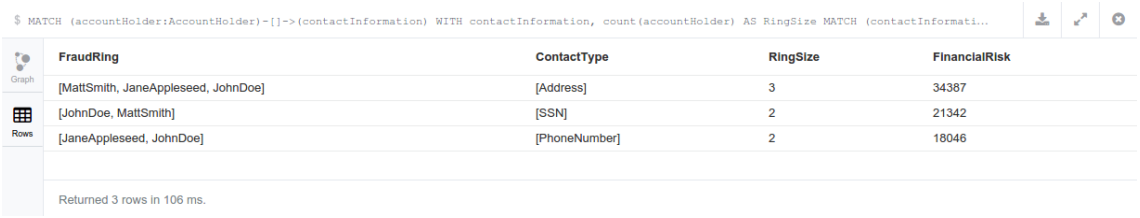

Fig. 11: Classical pattern resolution results

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

4.2 Historical Pattern Matching Resolution

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

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, the fraudsters can cheat to stay hidden by creating, updating and deleting some 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.

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

It should be noted that the VersionGraph manages the history in the same manner whatever the business area should be. The Mnemosyne system represents the history in the same way whatever 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 care about 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 impacts on the neighborhood. 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).

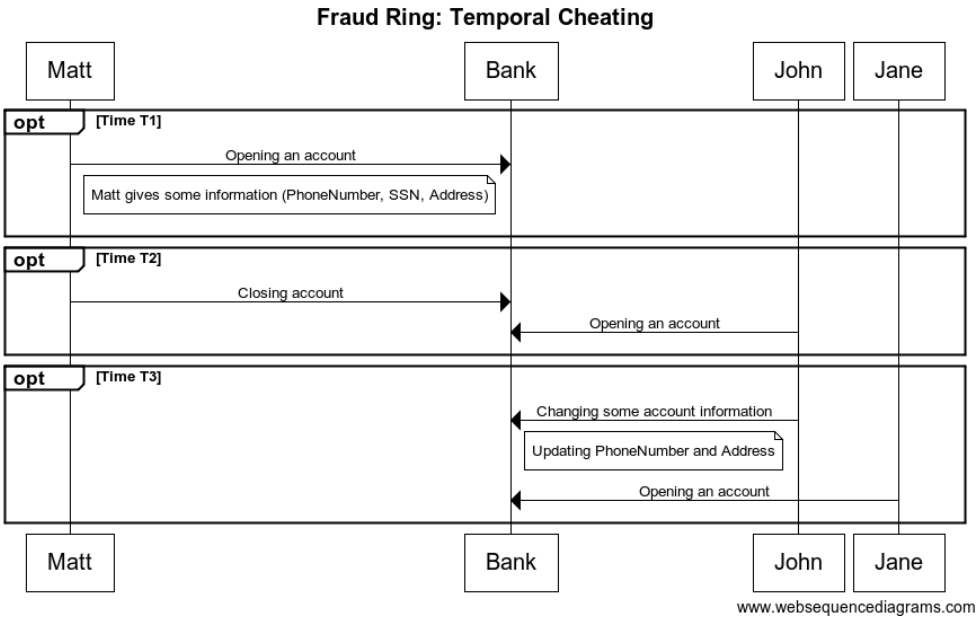

Fig. 12: Example of temporal cheating applied to current dataset

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

Resolution Resolving such patterns in NoSQL graph databases requires to define 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.

Listing 1.2 shows how to query all the links from the history in order to retrieve the potential fraud ring.

Listing 1.2: 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 on this Cypher query, all the elements impacted in the graph can easily be retrieved by exploiting the materialized links of the history from the extension we propose. However, we claim that the patterns to be found cannot

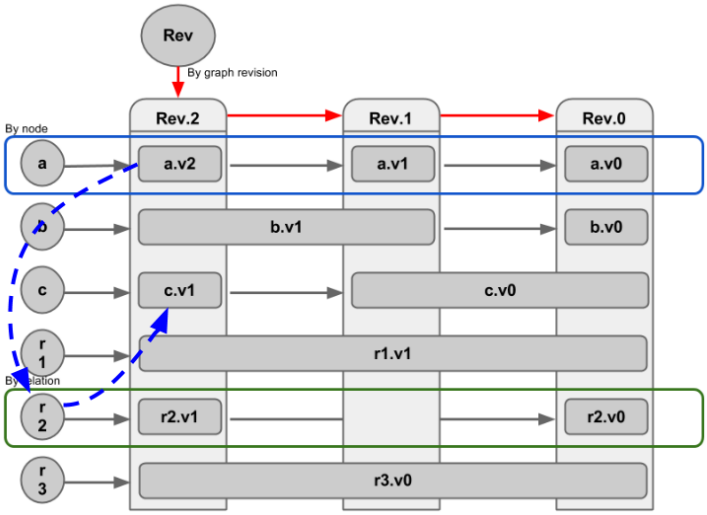

Fig. 13: Materializing Elements impacted by an operation in the VersionGraph

be defined in a crisp manner and propose below an extension to fuzzy historical patterns.

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

We thus propose to use fuzzy clauses in the pattern matching queries.

Temporal fuzzy clauses can be defined either by considering the fuzzy Allen intervals or by considering user defined clauses. Fig. 15 displays a membership function defining fuzzy weeks that is exploited in Listing 1.5. Rings can be characterized in the same way as the length of the paths forming the relations and the ring ranges within fuzzy bounds, as shown on Listing 1.4.

Listing 1.3: 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 1.4: Fuzzy Pattern Matching on Fuzzy Lengths

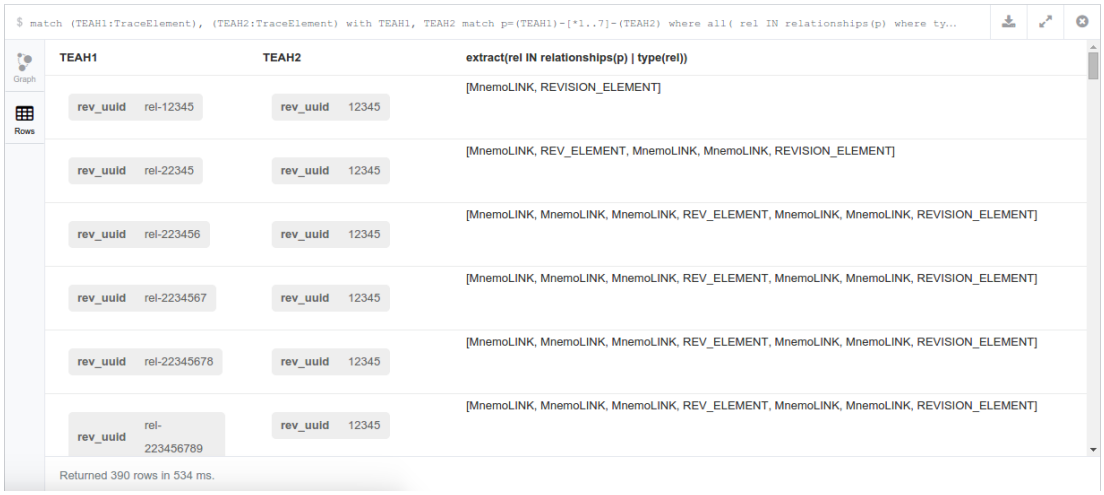

Fig. 14: Mnemosing Ring: Exploiting Historical Revisions

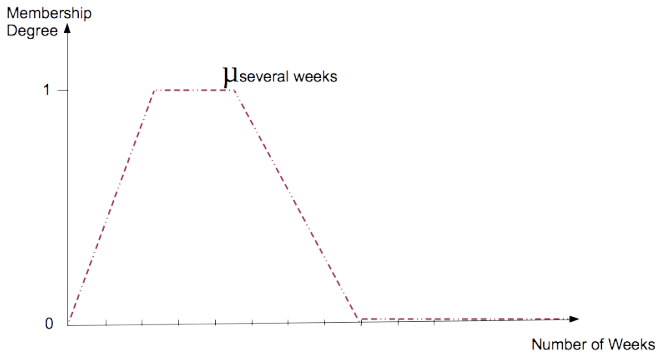

Fig. 15: Trapezoidal Membership Function of the Fuzzy Set FuzzyWeeks

```
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)
```

Listing 1.5: Fuzzy Pattern Matching on Fuzzy Weeks

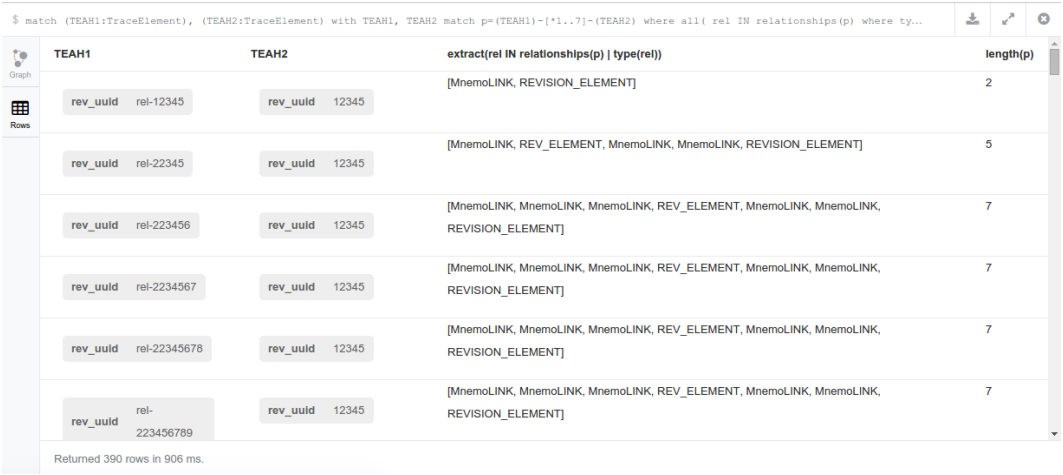

Fig. 16: Mnemosing Ring : Exploiting the Length

```
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') and ↩
      fuzzyWeeks(p) > 0.7
5 return TEAH1, TEAH2,  extract(rel IN relationships(p) | type(rel))
```

5 Conclusion and Further Work

In this paper, we address the problem of fraud detection by considering fraud rings in graph databases. 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 the Mnemosyne system that is extended here for materializing temporal relations between objects. This allows us to directly apply pattern matching on the graph for retreiving the fraud rings.

Fuzziness allows us to take better account of the fraud characteristics. The patterns may indeed differ from one case to another and considering crisp values when scanning the data may prevent from retrieving the relevant patterns.

Our further works will especially address the comparison of various strategies for materializing the history links in the Mnemosyne system. We also aim at testing our methods on various datasets, for instance for financial crime detection and counter-terrorism.

References

1. Annual fraud indicator. In: National Fraud Authority (2013)

2. Andrei Krokhin, P.J., Jonsson, P.: Reasoning about temporal relations: The tractable subalgebras of allen's interval algebra. Journal of ACM 50(5) pp. 591–640 (2003)
3. Angles, R., Barcel, P., Ros, G.: A practical query language for graph dbs. In: 7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW) (2013)
4. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. 40(1) (2008)
5. Auer, S., Herre, H.: A versioning and evolution framework for rdf knowledge bases. In: Proceedings of the 6th international Andrei Ershov memorial conference on Perspectives of systems informatics. pp. 55–69. PSI'06, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1760700.1760710
6. Bastani, K.: In: Bank Fraud Detection (2014), https://github.com/neo4j-contrib/gists/blob/master/other/BankFraudDetection.adoc
7. Board, T.T.A.: Technology radar, http://thoughtworks.fileburst.com/assets/technology-radar-may-2013.pdf (May 2013)
8. Bondy, J.A.: Graph Theory With Applications. Elsevier Science Ltd (1976)
9. Castelltort, A., Laurent, A.: Representing history in graph-oriented nosql databases: A versioning system. In: Proc. of the Int. Conf. on Digital Information Management (2013)
10. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41(3), 15:1–15:58 (Jul 2009), http://doi.acm.org/10.1145/1541880.1541882
11. Chawathe, S.S., Abiteboul, S., Widom, J.: Representing and querying changes in semistructured data. In: Proc. of the ICDE Conf. (1998)
12. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty Years Of Graph Matching In Pattern Recognition. International Journal of Pattern Recognition and Artificial Intelligence (2004)
13. He, H., Singh, A.K.: 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. pp. 405–418. SIGMOD '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1376616.1376660
14. Horvth, T., Grtner, T., Wrobel, S.: Cyclic pattern kernels for predictive graph mining. In: Kim, W., Kohavi, R., Gehrke, J., DuMouchel, W. (eds.) Proceedings of the 10t ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004), August 22-25, 2004, Seattle, WA, USA. pp. 158–167. ACM Press, New York, NY, USA (2004), http://doi.acm.org/10.1145/1014052.1014072
15. Jedrzejek, C., Bak, J., Falkowski, M.: Graph mining for detection of a large class of financial crimes. In: 17th International Conference on Conceptural structure (ICCS'09) (2009)
16. Khurana, U., Deshpande, A.: Efficient snapshot retrieval over historical graph data. In: Jensen, C.S., Jermaine, C.M., Zhou, X. (eds.) ICDE. pp. 997–1008. IEEE Computer Society (2013)
17. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proceedings of the 2001 IEEE International Conference on Data Mining. pp. 313–320. ICDM '01, IEEE Computer Society, Washington, DC, USA (2001), http://dl.acm.org/citation.cfm?id=645496.658027
18. Lee, K.Y., Chung, Y.D., Kim, M.H.: An efficient method for maintaining data cubes incrementally. Inf. Sci. 180(6), 928–948 (Mar 2010), http://dx.doi.org/10.1016/j.ins.2009.11.037

19. Leontjeva, A., Tretyakov, K., Vilo, J., Tamkivi, T.: Fraud detection: Methods of analysis for hypergraph data. In: ASONAM. pp. 1060–1064. IEEE Computer Society (2012), http://dblp.uni-trier.de/db/conf/asunam/asonam2012.html#LeontjevaTVT12
20. Moody, S.: Advanced techniques for detecting complex fraud schemes in large datasets. In: BAE Systems Detica 2012 COMMERCIAL IN CONFIDENCE Date/reference/classification 1 (2013)
21. Noble, C.C., Cook, D.J.: Graph-based anomaly detection. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 631–636. KDD '03, ACM (2003)
22. Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V.: High-level change detection in rdf(s) kbs. ACM Trans. Database Syst. 38(1), 1:1–1:42 (Apr 2013), http://doi.acm.org/http://dx.doi.org/10.1145/2445583.2445584
23. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly (2013)
24. Sadowski, G., Rathle, P.: Fraud detection: Discovering connections with graph databases. In: White Paper - Neo Technology - Graphs are Everywhere (2014)
25. Schockaert, S., Cock, M.D., Kerre, E.E.: Fuzzifying allen's temporal interval relations. IEEE T. Fuzzy Systems 16(2), 517–533 (2008), http://dblp.uni-trier.de/db/journals/tfs/tfs16.html#SchockaertCK08
26. W3C: Allegrograph rdfstore web 3.0's database (2009)
27. Wang, J., Zhang, K., Chirn, G.: The approximate graph matching problem. Pattern Recognition, Proceedings of the 12th International Conference on IAPR 94, 284–288 (1994)
28. Wood, P.T.: Query languages for graph databases. SIGMOD Rec. 41(1), 50–60 (Apr 2012), http://doi.acm.org/10.1145/2206869.2206879
29. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: Proceedings of the 2002 IEEE International Conference on Data Mining. pp. 721–. ICDM '02, IEEE Computer Society, Washington, DC, USA (2002), http://dl.acm.org/citation.cfm?id=844380.844811