Data Intensive Computing Project

Barth Niklas, Camerota Fabio, Castellano Giovanni, Santoro Matteo

October 2022

1 Introduction

Nowadays, electronic money transactions are becoming more and more widespread and this lead to the generation of large amount of data. Banks all over the world try to analyse this data to improve their service and provide a better experience to their customers.
It is clear that a network of transactions can be represented as a graph, which can easily be stored on a graph database such as Neo4j. One of the most interesting analysis which can be performed on the graph is called fraudsters detection, where we try to identify customers who operate for the purpose of scamming people.
The goal of this project is to explore how Neo4j can be used to perform such type of analysis. In particular, we try to replicate the analysis [1] performed by the Neo4j team on the PaySim dataset [2] [3], moreover, we study how working only on a subset of the graph affects the results. This last experiment is extremely useful to understand how much data is needed to get reliable results and if we can easily handle situations where computational resources are not enough to analyse the whole graph.

2 Dataset

The PaySim dataset contains data generated by a financial simulator that generates fake mobile money transactions based on an original dataset [3]. The original data was collected by an african company which provides mobile financial services.
In the graph there are mainly three types of nodes: **agents**, **transactions** and **identifiers**. The agents are the users of the system; they can be clients, merchants, or banks. Furthermore, we have different type of transactions depending on the type of payment. Usually, clients and merchants exchange money by means of banks. Figure 1 illustrates an example of a transaction.
Each client moreover, has some relationships with identifiers nodes; there are three types of identifiers nodes: phone number, email address and SSN.
For a detailed description of the dataset see Figures 2, 3 and 4.

3 Methods

Our initial idea was to store the data on Neo4j Aura and to interface with the database using Google Colab. However, Neo4j Aura does not allow to work on large amount of data in its free version. We considered the possibility to downsample the data, but to fulfill Neo4j Aura requirement we would need to drop 75% of the nodes in the graph. Therefore, we decided to store the database on our local machine and to use Python to interface with the database.
Most of the code we wrote is based on Neo4j GDS library [5], which provides a collection of graph analysis algorithms.

An important aspect of this library is that each algorithm requires as input a projection of the database, this means that before calling a GDS function we first need to project a subset of the database on the RAM memory.

3.1 First-party fraudsters identification

The first step of the analysis is to identify first-party fraudsters; i.e., a customer pretending to be someone else providing false information about himself.
The idea is that clients who share identifiers are likely to be first-party fraudsters. To keep track of this information we added a weighted relationship to the database which represents the number of identifiers shared between two clients. We then run the weakly connected component algorithm to identify clusters of clients who share identifiers. Afterwards, we compute the Jaccard similarity score between each pair of clients within each cluster, and store the results on a new weighted relationship called SIMILAR_TO. Finally, for each of these clients we compute the weighted degree centrality based on the SIMILAR_TO relationship. The nodes with a degree centrality higher than a certain threshold (95th percentile) are classified as first-party fraudsters.

3.2 Second-party fraudsters identification

We proceed with the identification of clients who could have supported first-party fraudsters; these nodes are called second-party fraudsters. The hypothesis is that clients who exchange a lot of money with first-type fraudsters are likely to be involved in the scam; in the following, we will refer to such clients as suspects.
First, we create a new directed and weighted relationship called TRANSFER_TO between suspects and fraudsters describing the amount of money transferred between the two nodes. Using once again the weakly connected components algorithm, we are able to identify clusters of clients who are connected to fraudsters. Finally, we run the PageRank algorithm to score each suspect in term of amount of money transferred to fraudsters or received from fraudsters.

3.3 How the amount of available data affects the analysis?

Considering the difficulties we faced at the beginning of the project with Neo4j Aura, we decided to study how the results of the analysis change as we decrease the amount of data. Such study can be useful to understand how reliable can be the results that we obtain in a scenario where we do not have a lot of data or, where we do not have enough computational capacity to handle the whole dataset.
Initially, we thought to downsample the graph removing a certain percentage of random nodes. This method however leads to an inconsistent database; for instance, we could end up in a situation where we have a transaction without any associated client. Instead, we downsampled the graph simply removing a certain percentage of transactions. In fact, as shown in figure 2, most of the nodes in the database represent transactions and this choice leads to a consistent database. We repeated this experiment many times dropping an always higher number of random transaction nodes (5%, 15% and 30%).

4 Results

The analysis led to the identification of 17 first-party fraudsters, this number is relatively small compared to the number of clients who share identifiers (see Figure 5). Moreover, we identified 46 second-party fraudsters. This number includes everyone who has a pagerank score higher than 0; looking at the pagerank score we can have more information about second-party fraudsters. The following table shows how the results change as we drop some of the transaction nodes.

As we can observe, the number of first-party fraudsters never changes. This happens because the identification of such fraudsters does not depend on the transactions. Even if the number of transactions on the dataset was zero we would still

Table 1: Number of first-party and second-party fraudsters after dropping a certain percentage of transactions

	firs-party	**second-party**
0%	17	46
5%	17	43
15%	17	37
30%	17	36

be able to identify first-party fraudsters. Considering the results illustrated in figure 2, we can conclude that we need the preserve only a very small percentage of the graph to identify these type of fraudsters.
However as we expected, the number of second-party fraudsters decreases as we decrease the number of transactions. This happens because we do not have enough data to identify all of them. It is interesting to notice that the number does not decrease linearly with the percentage of dropped transactions.

5 How to run the code

1. Install Neo4j Desktop [4]
2. Download the PaySim dataset [2]
3. Drop the file into the Files section of a project in Neo4j Desktop
4. Choose *create new DBMS from dump* from the file options
5. Install the following plugins: *APOC* and *Graph Data Science Library*
6. Start the DBMS
7. Open the Python file and install Neo4j Python Driver
8. Execute *runAll()* function
9. The result is the PaySim dataset where first-party fraudsters and second-party fraudsters have been identified and labelled

The same analysis can be performed also on downsampled datasets:

1. Repeat steps 4-6
2. Open the Python file and modify the variable *percentage_transactions* inside *remove_transactions()* function, changing the percentage of transactions to be deleted (default is 5%)
3. Run *remove_transactions()* function
4. Execute *runAll()* function
5. The result is the PaySim downsampled dataset where first-party fraudsters and second-party fraudsters have been identified and labelled

References

[1] *Fraud Detection Using Neo4j Platform and PaySim Dataset*. URL: https://guides.neo4j.com/sandbox/fraud-detection/index.html.

[2] Michael Hunger. *PaySim Dataset, github*. 2021. URL: https://github.com/neo4j-graph-examples/fraud-detection/blob/main/data/fraud-detection-43.dump.

[3] Edgar Alonso Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. "PAYSIM: A FINANCIAL MOBILE MONEY SIMULATOR FOR FRAUD DETECTION". In: Sept. 2016.

[4] *neo4j*. URL: https://neo4j.com/.

[5] *Neo4j GDS*. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/.

[6] sisu.io. *Simulating Mobile Money Fraud*. 2020. URL: https://www.sisu.io/posts/paysim/.

Figures

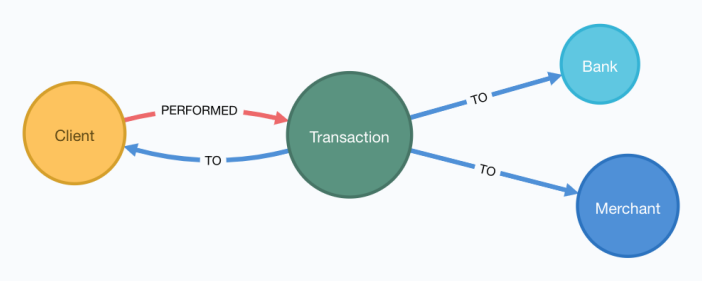

Figure 1: An example of how a transaction is represented in the graph [6].

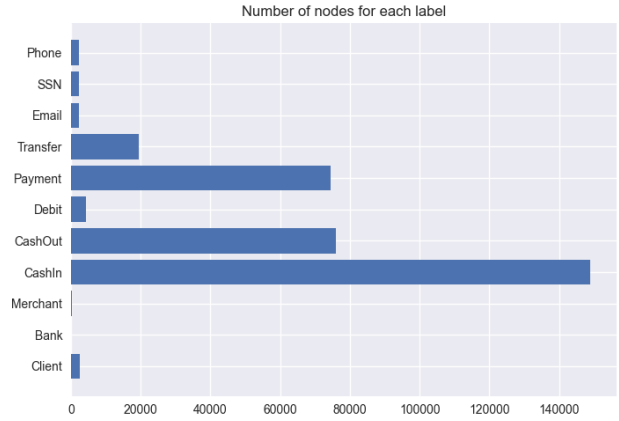

Figure 2: Number of nodes for each label present in the dataset.

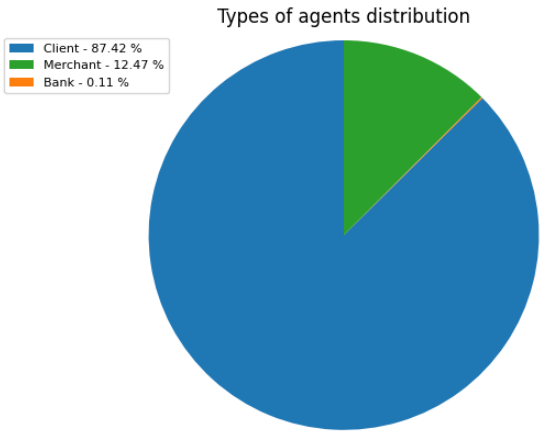

Figure 3: Distribution of the different agents over the graph

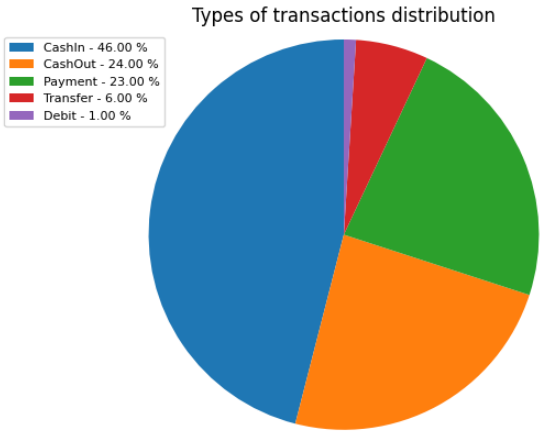

Figure 4: Distribution of the different transactions over the graph

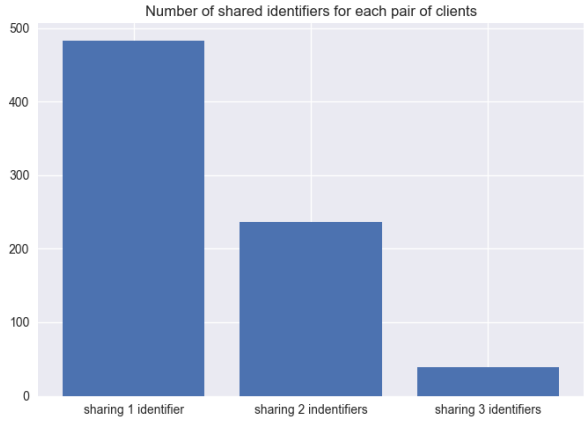

Figure 5: Number of nodes sharing identifiers.