Anti-Money Laundering Visualization

Debopriyo Ghosh
Rutgers University
dg1114@scarletmail.rutgers.edu

Nish Patel
Rutgers University
nsp124@scarletmail.rutgers.edu

Jahnavi Manchala
Rutgers University
jm2658@scarletmail.rutgers.edu

***Abstract*—With massive graph networks, the analyst can not visualize or analyze the graph using a traditional visualization. Despite this, analysts will need to perform an ad-hoc review of accounts to identify potential accounts of interest. Visualizing the data with a graph city solves this problem by allowing analysts to see the entire graph and perform analysis one building at a time. We used the graph city architecture introduced by James Abello, H. Zhang, Daniel Nakhimovich, Chengguizi Han, and Mridul Aanjaneya.[1]**

***Keywords*—** Money Laundering Detection, Graph Cities, Data Visualization, Synthetic Laundering Transactions, Neo4j, Neovis, PySpark, Monitoring Transactions

I. Introduction

Money laundering detection is a critical issue in the financial sector, where the stakes are exceptionally high due to the potential for large-scale financial crimes. Real financial transaction data, which is pivotal in identifying and combating such illicit activities, is highly confidential and difficult to obtain for privacy reasons. This confidentiality is necessary to protect the personal and financial information of individuals and institutions but simultaneously poses a significant challenge for regulatory bodies and financial institutions striving to detect and prevent money laundering. As a result, developing effective detection systems often relies on sophisticated techniques that can work with limited or synthesized data, while still accurately identifying suspicious activities without violating privacy norms and regulations. IBM has developed a simulation model to generate synthetic transactions. The simulation includes simulated money laundering and these transactions are labeled in the dataset.

Our project aims to provide analysts with a visualization tool to perform an ad-hoc analysis of financial transaction data to identify possible accounts of interest. Once the accounts of interest are found, we aim to help the analyst review financial transactions conducted by the accounts and their related accounts.

II. Dataset Description

The dataset has been generated by IBM and is based on a virtual world inhabited by individuals, companies, and banks. Individuals interact with other individuals and companies. Likewise, companies interact with other companies and with individuals. These interactions can take many forms, e.g. purchase of consumer goods and services, purchase orders for industrial supplies, payment of salaries, repayment of loans, and more. These financial transactions are generally conducted via banks, i.e. the payer and receiver both have accounts, with accounts taking multiple forms from checking to credit cards to bitcoin. The data generator that created the data not only models illicit activity, but also tracks funds derived from illicit activity through arbitrarily many transactions. We use a higher illicit ratio (more laundering) for our project.[2] The dataset (17 GB in size) contains 180 million records (100 bytes per record), 2.1 million distinct bank accounts, 15 distinct currencies , and 7 distinct payment formats. The data was generated via the IBM simulation from August 1, 2022 to November 5, 2022.[3]

Timestamp	From Bank	Account	To Bank	Account	Amount Receive	Receiving Currency	Amount Paid	Payment Currency	Payment Format	Is Laundering
9/1/22 0:22	800319940	8004ED620	808519790	872ABC810	120.92	US Dollar	120.92	US Dollar	Credit Card	0
9/1/22 0:05	8021ADE00	8023BF220	9A7F59FA0	A23691240	33.97	US Dollar	33.97	US Dollar	Credit Card	1
9/1/22 0:14	801946100	8023F0980	83585F5A0	948893910	79.20	US Dollar	79.20	US Dollar	Credit Card	0
9/1/22 0:05	80010C840	800122AA0	80010C840	800122AA0	8,834.09	Euro	10351.64	US Dollar	ACH	0
9/1/22 0:05	80010C840	800122AA0	80010CF20	80012DA00	8,834.09	Euro	8834.09	Euro	ACH	0
9/1/22 0:08	80010CF20	80012DA00	80010CF20	80012DA00	9,682.16	US Dollar	8262.75	Euro	ACH	0
9/1/22 0:08	80010CF20	80012DA00	80010BD60	80011E460	9,682.16	US Dollar	9682.16	US Dollar	ACH	0
9/1/22 0:03	800319940	800466670	80029A010	8002F6F20	9,125.22	US Dollar	9125.22	US Dollar	ACH	0

Fig. 1. Dataset Record Structure

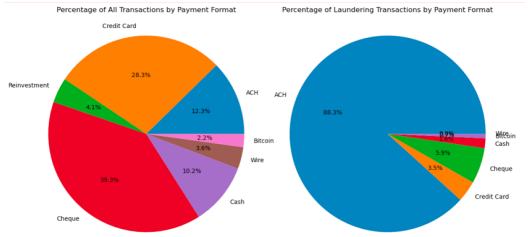

Fig. 2. Transactions in Payment Format

III. Questions to Answer

Currently, the industry relies heavily on manual analysis of transactions in order to flag and review fraudulent transactions with a high false positive rate. As data volume has increased and new digital currencies have been introduced, the challenge of identifying potential money-laundering transactions with legacy rules-based has increased exponentially. The three fundamental questions we are asking are:

1) What is the best data type for financial transaction data?
2) How can we help analysts perform ad-hoc analysis of financial transaction data to identify possible accounts of interest?
3) Once the accounts of interest are found, how can we help the analyst review financial transactions conducted by the accounts and their related accounts?

IV. Methodology

- To answer question 1, a literature review was conducted to identify the current trends in the space. Legacy rules-based analysis relied on the table data structure within relational databases and pre-defined thresholds for identifying potential fraudulent transactions. Transactions over ten thousand dollars must be reported to the IRS. The key component that was missing from these systems was the ability to easily view the context of the transactions and the accounts that money is being transferred from or to. As graph analytics[4] have evolved due to the rise of social networks, the financial industry has also embraced graphs to model financial transactions as a series of relationships to provide the missing context. Based on the literature review, we decided to utilize the graph data structure to model our data. Given the need to store, query, and visualize graph data, the data was converted [5]into nodes and edges via a Python script and imported into a graph database (Neo4j) that can natively handle the relationships and provide an efficient query mechanism.

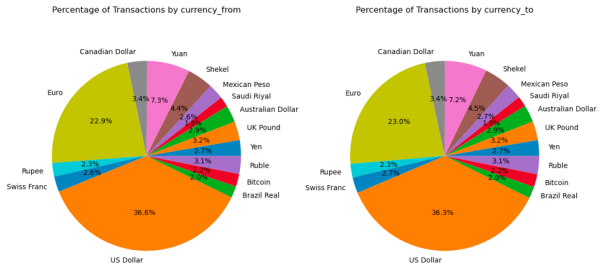

Fig. 3. Transactions in Currency Format

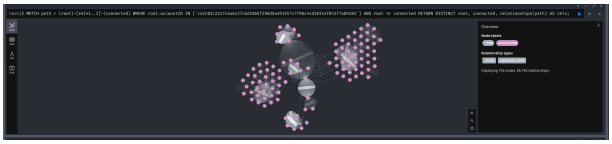

Fig. 4. Neo4j Database

- To answer question 2, the main issue to address was the screen bottleneck. With massive graph networks, the analyst can not visualize or analyze the graph using a traditional visualization. Despite this, analysts will need to perform an ad-hoc review of accounts to identify potential accounts of interest. Visualizing the data with a graph city solves this problem by allowing analysts to see the entire graph and perform analysis one building at a time. We used the graph city architecture introduced by James Abello, H. Zhang, Daniel Nakhimovich, Chengguizi Han, and Mridul Aanjaneya.[1][6][7][8]

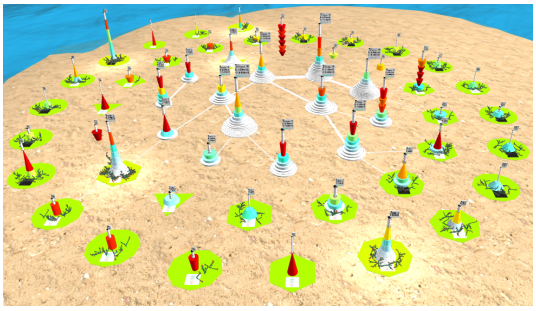

Fig. 5. Graph City Buildings

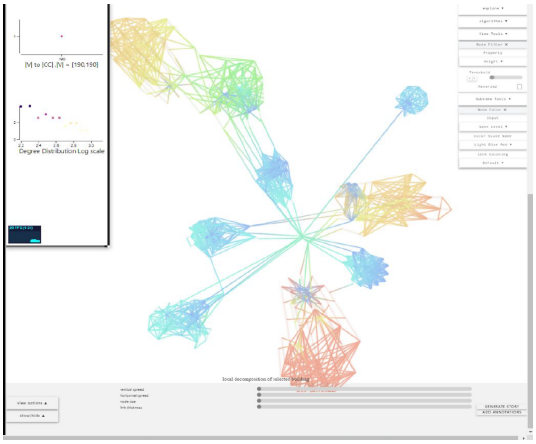

Fig. 6. Graph Nodes

- To answer question 3, we needed to figure out how to leverage Neo4j's advanced capabilities to allow analysts to view transactions for particular accounts and their related accounts. This was done by creating a search application that allows the user to filter on certain key attributes and visualize a graph of an account (or series of accounts).

Search Application

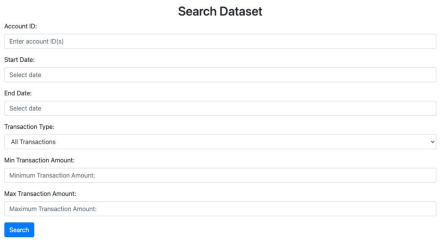

Fig. 7. Search Application

Once the query is submitted, a graph visualization is rendered that shows all transactions by that account and also any transaction by accounts that the account sent money to (depth 2).[9]

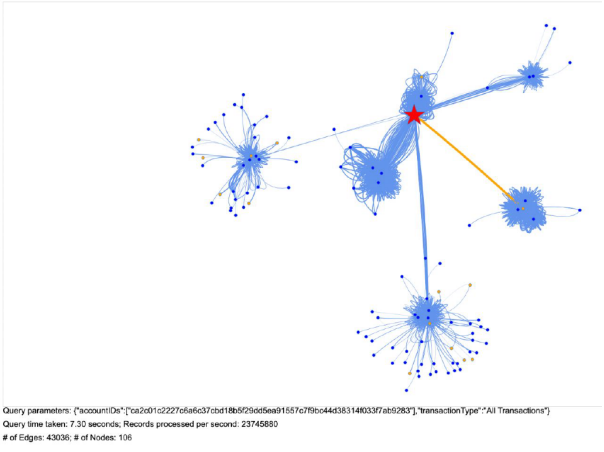

Fig. 8. Query Result

An interesting finding is that communities of interest are readily apparent when visualizing the accounts through the search application.

V. Implementation

- We performed data cleaning and preprocessing steps to prepare a CSV to be used in the analysis.
  - Check for nulls (no null values in any column)
  - Calculate conversion rates from each currency to USD and prepare a new column of payments in US Dollars
  - Create a unique id by hashing bank + '_' + account
  - Convert timestamp columns into integer year, month, day, hour, minute columns to save space
  - Output a 'clean' CSV for use in visualizations and modeling
- We use the clean data to create two CSVs with the account data (accounts.csv) and the transaction data (transactions.csv)
- We import these account and transaction information data into Neo4j and build a database.
- We build a web application that primarily uses client-server architecture with MVT (Model, View, Template) design pattern.

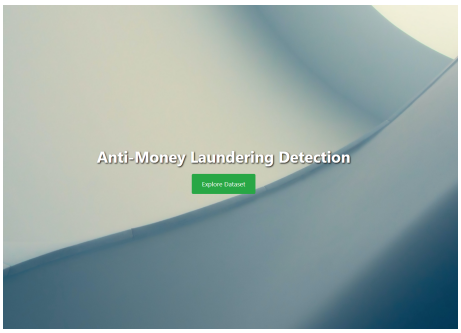

Fig. 9. Web App home page

- We format the dataset as aml.txt and aml_label.csv for importing into the Graph City backend.
- We use the Graph City Architecture to generate the Graph City for our dataset.

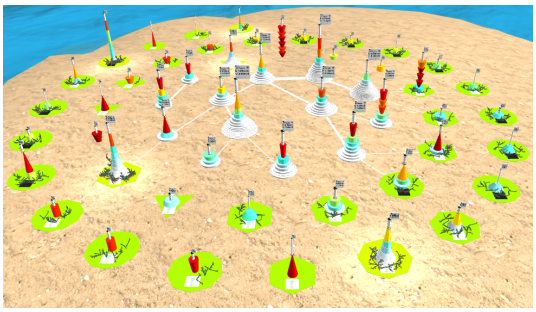

Fig. 10. Generated Graph City for AML

- We link the Graph City Front-End to our application.

The application is used to connect to Graph City and provide a high-level visual representation of the data. The user can now explore the network clusters that can be used to initiate an analysis of the data:

- Explore a cluster in Graph City
- Click on a node to redirect to our app to query Neo4j
- Generate the cluster visualization of connected nodes

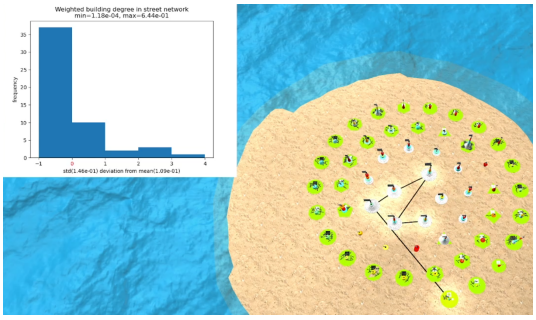

Fig. 11. Data Exploration

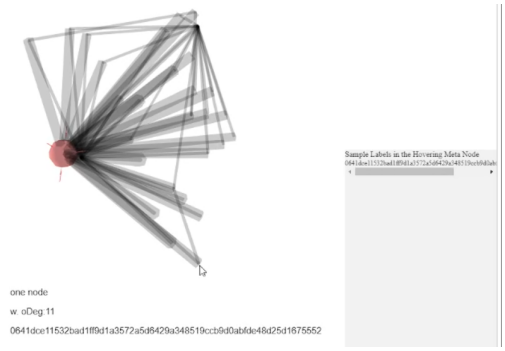

Fig. 12. Node Exploration

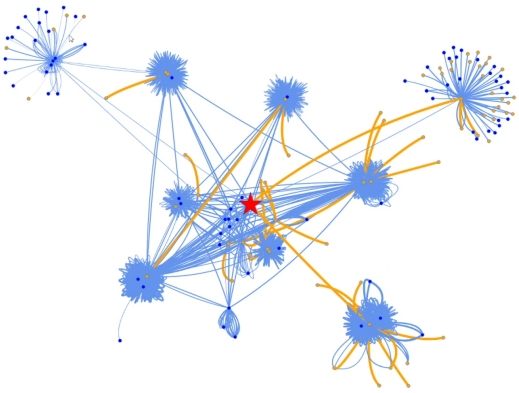

Fig. 13. Connected Node Exploration

- Find laundering transactions and follow laundering chains

VI. Deep Learning

- The project was extended for the Deep Learning aspect individually by Nish Patel. In this second phase of the project, the fundamental question asked was "Can deep learning be utilized to predict the accounts that are involved in a money laundering transaction?"
- With the findings in Phase 1 being that graphs are the most suitable data structure to model this problem, a literature review was conducted on graph convolutional neural networks. With the recent advances in graph convolutional neural networks, this was the most prevalant method of leveraging deep learning in the literature review. Essentially the problem boils down to a very complex binary classification problem with the goal being that target end users (fraud analysts) can be alerted to

potential fraudulent accounts for manual investigation by the neural network.

- Once graph convolutional neural networks were decided as the methodology, a python script was created to generate the necessary input files to create a pytorch graph object using the torch_geometric extension. This extension allows the graph to be directly input as the training/test data into graph convolutional neural networks.
- When first training the model, results were very poor (balanced_accuracy ˜0.5). Given there were only 3 node features (account_id, bank which are included in the dataset and is_laundering which was created in phase 1), a feature engineering task was conducted to create more node features to help the model identify patterns for the predictions.
- 8 features were created:
  - transactions_from, transactions_to: the total number of payments made and received by the account
  - total_usd_from, total_usd_to: the total amount of money (in USD) sent and received by the account
  - avg_usd_from, avg_usd_to: the average transaction amount (in USD) sent and received by the account
  - total_currencies_from, total_currencies_to: count of the number of unique currencies used in transactions by the account
- These features were created to give the model information on the account's behaviors to try to identify outlier patterns. For example, if an account has a low avg_usd_from/to and there is suddenly a large transaction, that could be something that could be an identifying factor for the model to predict if the account is involved in laundering transactions or not.
- The additional features did not help model performance, however, as the balanced_accuracy stayed steady at approximately 0.5
- The next approach was to introduce a Graph Attention Network (GAT) to capture the differing weightage that should be applied to neighboring nodes. In a GCN, the weights assigned to neighboring nodes is the same for all neighbors. By adding attention, the weights for neighboring nodes can be individually assigned and the network can selectively attend to the neighboring nodes that are more important. This is particularly important because of the class imbalance that is present in the data (only ˜0.1% of the transactions and ˜2% of accounts are flagged as money laundering). Attention provides the ability to give weightage to neighboring nodes that are involved in money laundering activity even if they are a much smaller portion of the total nodes
- The results with GAT were still staying steady with balanced_accuracy at approximately 0.5.
- The next item to tweak was the loss function. pytorch offers an option to provide weighting to account for the class imbalance in the loss function if you use BCEWithLogitsLoss so the models were updated to use this loss function. The weighting was calculated based on the counts. Weighting for laundering accounts was given as the total number of accounts/number of laundering accounts to account for class imbalance.
- This had a marginal improvement in the GAT model with balanced_accuracy ticking up to approximately 0.52.
- With the given time constraints, other models were not able to be tried but the GAT with BCEWithLogitsLoss was used as the base model. Hyperparameters were tuned for this model:
  - Learning rate – a final value of 0.0001 is used
  - Number of epochs – a final value of 15 epochs is used
  - Number of convolutional layers – 2 convolutional layers are used in both GAT and GCN
  - Number of dense layers – 2 dense layers are used in GCN and 1 dense layer is used in GAT
  - Dropout – a value of 0.3 is used in both GAT and GCN
- The above iterative approach was all done on a smaller dataset (approx 5.5M edges and approx 500k nodes) as the large dataset had onerus training time (˜6 hours per model run). Once the hyperparameters were tuned on the GAT with BCEWithLogitsLoss model, a conda environment was created in ilab and the model was set to run via slurm on the large dataset with the gpu's available on ilab.
- The final results of the large model were similar to the results with the small model with a balanced_accuracy of approximately 0.52. Results for each iteration and the hyperparameters used are included in the below table.

Run #	features?	dataset	model	loss	conv layers	dense layers	dropout	lr	accuracy
1	no features	small	GCN	BCE	2	2	0.3	0.0001	0.4941
2	features	small	GCN	BCE	2	2	0.5	0.0001	0.5013
3	features	small	GAT	BCE	2	1	0.3	0.0001	0.5019
4	features	small	GCN	BCELogit(weight)	2	2	0.5	0.0001	0.5052
5	features	small	GAT	BCELogit(weight)	2	1	0.3	0.0001	0.5232
6	features	small	GAT	BCELogit(weight)	3	1	0.3	0.0001	0.5083
7	features	small	GAT	BCELogit(weight)	2	2	0.3	0.0001	0.4988
8	features	small	GAT	BCELogit(weight)	2	2	0.5	0.0001	0.5062
9	features	small	GCN	BCELogit(weight)	2	2	0.3	0.0001	0.5077
10	features	small	GCN	BCELogit(weight)	3	3	0.3	0.001	0.5084
11	features	small	GAT	BCELogit(weight)	2	2	0.3	0.0005	0.52
12	features	large	GAT	BCELogit(weight)	2	1	0.3	0.0001	0.5204

Fig. 14. Deep Learning Results

- These results are disappointing as the balanced_accuracy did not substantially improve with the different iterations. One reason could be that there not enough distinguishing features to accurately predict accounts that are involved in money laundering. In a real-life scenario, other information about the owner of the account would be available and provide context of multiple accounts being owned by the same person. In this dataset, there is no information on individuals and their assets, which makes the task more difficult

VII. Architecture

The user interface was developed with HTML5, CSS, Javascript, and Node.js. The database was created using Neo4j and Neovis was used to visualize the data. Graph City interface was implemented using the GitHub repository[10] Additional libraries and dependencies include:

- Python
- PySpark
- Jupyter Notebook
- Pandas
- Matplotlib
- Numpy

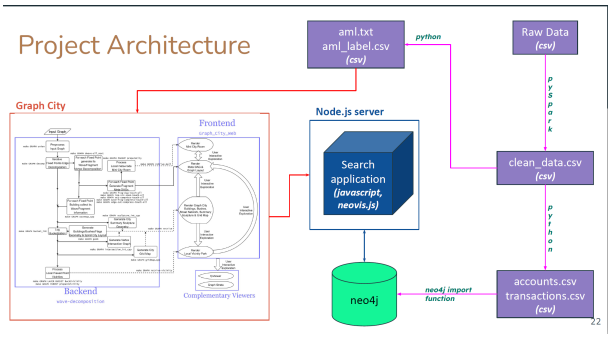

Fig. 15. Project Architecture

For deep learning, the pytorch was used for model training and the below project architecture was used.

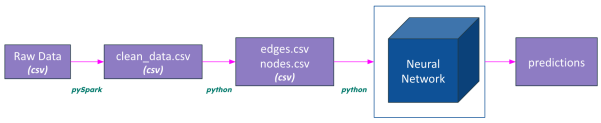

Fig. 16. Deep Learning Project Architecture

VIII. Project Timeline

Fig. 17. Project Timeline and Gantt Chart

The deep learning timeline and gantt chart is included below.

Fig. 18. Deep Learning Project Timeline and Gantt Chart

Division of Work:

- Nish Patel: Data Processing, Background Processing, UI, Visualizations, Testing, Deep Learning
- Debopriyo Ghosh: UI, Graph-City Implementation, Integration, Testing, Reporting
- Jahnavi Manchala: Data Visualizations, Graph-City Exploration, Testing

IX. Future work.

- evolveGCN was suggested in literature review as an alternative model. It includes time-series capababilities to model the temporal aspect of the problem. This model could be used to take into account the evolving nature of the graph.
- Integrate the predictions into the visualization application from Phase 1
- Incorporate comments provided on Phase 1 project

References

[1] J. Abello, H. Zhang, D. Nakhimovich, C. Han, and M. Aanjaneya, "Giga graph cities: Their buckets, buildings, waves, and fragments," *IEEE Computer Graphics and Applications*, vol. 42, pp. 53–64, 2022.

[2] Ibm money laundering. [Online]. Available: https://ibm.ent.box.com/v/AML-Anti-Money-Laundering-Data/file/780515045707

[3] Aml dataset. [Online]. Available: https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-moneylaundering-aml

[4] Graph analytics. [Online]. Available: https://datawalk.com/whitepaper-graph-analytics-the-new-game-changer-for-aml/

[5] Neo4j. [Online]. Available: https://neo4j.com/developer-blog/graph-visualization-with-neo4j-using-neovis-js/

[6] J. Abello, D. Nakhimovich, C. Han, and M. Aanjaneya, "Graph cities: Their buildings, waves, and fragments," in *EDBT/ICDT Workshops*, 2021.

[7] J. Abello and D. Nakhimovich, "Graph waves," *Big Data Res.*, vol. 29, p. 100327, 2020.

[8] J. Abello and F. Queyroi, "Fixed points of graph peeling," *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pp. 256–263, 2013.

[9] Neovis. [Online]. Available: https://github.com/neo4j-contrib/neovis.js

[10] Graph cities. [Online]. Available: https://github.com/endlesstory0428/Graph-Cities