Panama Papers Analysis using Graph Analytics and Embeddings

Eva Chamilaki
Dept. of Computer Science, University of Crete
OramaVR
Heraklion, Crete
evacham7@gmail.com

Manos Chatzakis
Dept. of Computer Science, University of Crete
ICS-FORTH
Heraklion, Crete
chatzakis@ics.forth.gr

ABSTRACT

The recent leak regarding Panama Papers shown that developing methods to analyze and study Fraud Detection graphs can provide significant results to improve our understanding of the data and find patterns in the available information. This report presents ways to analyze a Panama Papers dataset, using data preprocessing, graph analytics and embeddings.
Keywords: Fraud Detection, Panama Papers, Graph Analytics, Embeddings.

1 INTRODUCTION

This report is part of the project for the course cs484 - Complex Networks Dynamics. It presents methods to analyze a published Panama Papers dataset using data preprocessing, graph analytics and embeddings. Our contribution is summarized in the following:

- We present some data preprocessing techniques, containing data cleaning and sampling in order to be able to load and analyze smaller parts of the dataset.
- A data analysis on the dataset based on graph analytics methods. We run a set of applicable algorithms over the graph in order to locate the most important nodes in the network.
- A data analysis on the dataset based on exploiting Knowledge Graph Embeddings for Fraud Detection graphs, using the most important nodes based on the analytics. This way, we can offer methods so as to find similar nodes based on similarity search.
- A combination of the aforementioned approaches, utilizing the results of graph analytics for a graph embeddings database. The idea is depicted in the figure 1.

The rest of the report is organized as follows: Section 1 is the introduction, while Section 2 contains some basic background information. Section 3 describes how the dataset is preprocessed. Section 4 presents the work performed regarding the graph analytics. Section 5 presents the results of the embeddings creation over the Fraud Detection graph. Section 6 is the conclusion.

2 PRELIMINARIES

In this section, basic terms are presented.

[Fraud Detection Graph] Fraud detection is a set of processes and analyses that allow us to identify and prevent unauthorized financial activity. To analyze these activities better, the data are represented as a network visualizing the relationship between the companies, people and the other type of entities.

[Panama Papers] Documents with detailed financial and attorney–client information for more than 214,488 offshore entities. The documents, some dating back to the 1970s, were created by, and taken from, former Panamanian offshore law firm and corporate service provider Mossack Fonseca [9].

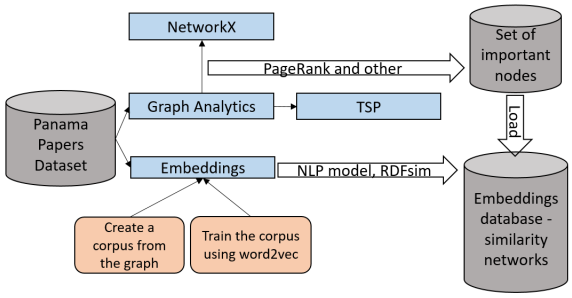

Figure 1: The work outline of this project

[Graph Analytics] Graph analytics is a set of processes that are executed over a network in order to harvest information and create statistics about the graph.

[Embeddings] Embeddings are vector representations of words in a multidimensional vector space used to aid the problem of similarity search by exploiting cosine similarity. There is a proliferation of approaches regarding the use of embeddings over the data of Knowledge Graphs, called Knowledge Graph Embeddings. They are a part of the broad field of Natural Language Processing.

3 DATASET PREPROCESSING

In this section we discuss general information about the dataset we used, how the preprocessing was done and what problems were encountered.

3.1 Dataset

The dataset we decided to use is a free access dataset containing information from offshore leaks starting back in 2013 [7]. It contains data about Panama, Pandora, Paradise papers etc. in a CSV format. For simplicity, we decided to use the data referring only to Panama papers. These data are a set of CSV files containing information about the nodes and a single CSV file that contains the edges between them. In total, there are 559600 nodes and 657488 edges.

3.2 Graph Type

The form of the graph can be summarized as a set of specific types of nodes and relations between them.

Regarding the nodes, we have:

- Entity: Shell company or offshore construct

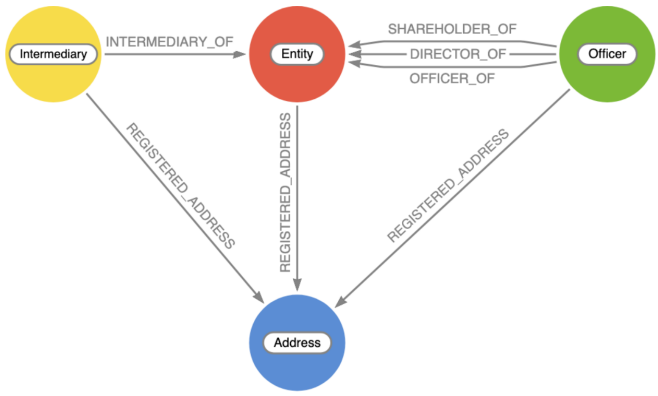

Figure 2: Fraud Detection Graph

- Officer: Proxy or real owners/shareholders/directors of a shell company
- Intermediary: Law firm or bank that helped create and manage the shell company
- Address: Registered addresses for the entities above

A summary of this schema can be seen in figure 2.

3.3 Preprocessing

The data, saved in a CSV format, contain many columns for the nodes that are not used for the graph analytics part. For the nodes, we decided to keep only the most basic information: ID, type of the node, country code and name. When the node represents an address, we decided to depict is using the name column. For the edges, we preprocessed the data to add a specific distinct link ID to every edge in the dataset, and we decided to keep: ID, SourceID, TargetID and information related to the type of the link between the nodes. We should note that any interaction and modification on the dataset was performed using CSVEditor, an open-source Java CSV editing API [1]. More information about these processes are presented later on.

3.4 Difficulties

We should note at this point that the size of the dataset is enormous. This wasn't a problem when implementing our own methods, but for analysis purposes we had to use graph analytics tools which could not maintain nor visualize such big datasets. To tackle this problem and to also explore how we can represent subsets of an entire dataset, we developed methods to take chunks of the dataset of desired size and load them to the tools we wanted. These methods are also presented later on.

4 GRAPH ANALYTICS AND VISUALIZATIONS APPROACH

This section presents methods used to analyse the data. These methods belong to three discrete categories: Base Dataset Analysis, Graph Analytics using Tom Sawyer Perspectives and Graph Analytics using NetworkX.

4.1 Base Dataset Analysis

The goal of this analysis is to find methods to get a subset of a big dataset without losing information about the form of the data. The results are used by the next analysis method for the visualization tool. The methods that will be presented have applications when we want to analyze chunks of a big data collection. They can be summarized in three approaches: The first approach just takes a sequential set of edges from the file and finds the corresponding nodes. The second follows the same logic using sampling. The third approach tries to reconstruct parts of the graph.

4.1.1 Simple. This method is the most simple one. To get a subset of the dataset, we just retrieve N sequential edges from the CSV edge file, and then we go through the corresponding node data from the collection of the CSV node files. The whole process is explained in the algorithm 1 in an abstract way. Although this method is fast and simplistic, it is obvious that the result can lead us to false conclusion about the form of the graph. This case could occur when the sample is too small.

4.1.2 Sampling. In order to be more accurate on the representation of the (sub)graph, we decided to use a variation of the previous algorithm, which randomly selects N edges from the file and proceeds to find the corresponding nodes. The whole process is abstractly illustrated in algorithm 2. This method actually gave good results, but we noticed that they did not create a large connected component of the graph (as we would expect) but instead small discrete networks of nodes. For this reason, we decided to investigate the matter even further, with a reconstruction method.

4.1.3 Reconstruction. To reconstruct paths of graph we created an algorithm that randomly selects a starting node and proceeds by creating its own network in a leveled BFS manner. The method is depicted in algorithm 3. Using this algorithm for large sets of nodes, we noticed the same behavior as in sampling. This means that small discrete networks of nodes is a case that occurs in the dataset many times. Although we used only one from the aforementioned approaches, the development of these algorithms helped us understand better how the data are structured. After some basic testing, we noticed that the best method to take subsets of the dataset is Sampling, thus we proceeded with this for the next step.

4.2 Analysis and Visualization using Perspectives

4.2.1 Software. Tom Sawyer Perspectives is a graph analytics and visualization tool.

4.2.2 Schema. To import the data, we needed to define a schema for our model. This schema should represent the type of the data and the interaction between them. We decided to define a simple schema. The schema contains the nodes with their attributes, and the corresponding edges, as seen in the figure 3.

4.2.3 Graph. By loading the dataset of the graph based on sampling, we managed to visualize a subgraph of the data. The subgraph consists of small (and bigger) networks of nodes. The graph is presented in figures 4 and 5, while the graph in full scale can be seen in figure 6.

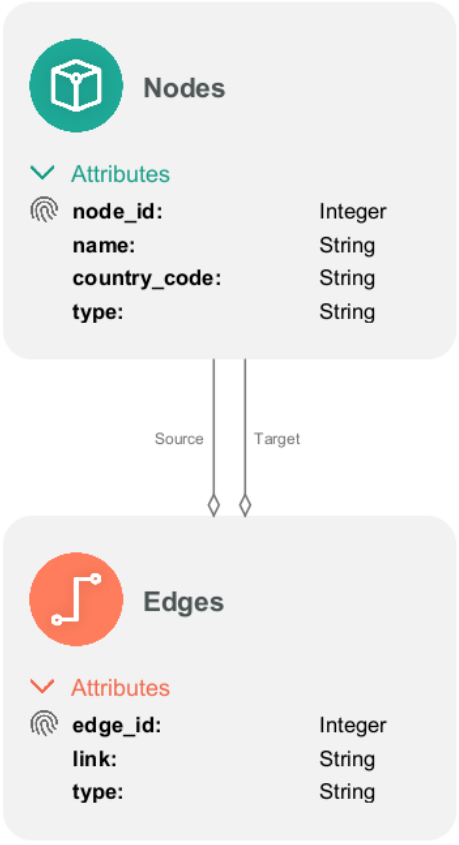

Figure 3: Model Schema

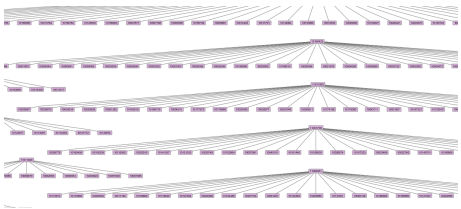

Figure 4: The subgraph using the node IDs as labels, as loaded from the Perspectives previewer.

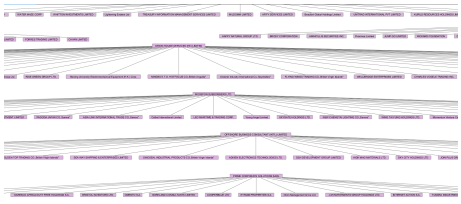

Figure 5: The subgraph using the node names as labels, as loaded from the Perspectives previewer.

Algorithm 1 Simple Algorithm: It takes a sequential number of edges and reconstructs a subset of the graph

Require: $startRow \geq 0$ To be the number of the row we will start taking edges
Require: $count \geq 1$ To be the number of edges to take in total
Require: $NodeOutputFile \neq NULL$ To be the result file path of the Nodes
Require: $EdgeOutputFile \neq NULL$ To be the result file path of the Edges

```
edges ← CSV(edgesPath)
rowID ← 0
selectedEdges ← edges.selectRows(startRow, startRow + count)
nodeIDs ← hashSet()

/*Phase 1: Keep the selected edges with a rowID in a separate CSV file and save the node IDs in a set, to find them later.*/
for all edges e in selectedEdges CSV file do
    currLine ← e.getLine()
    fromID ← currLine.getFromID()
    toID ← currLine.getToID()
    nodeIDs.add(fromID, toID)
    EdgeOutputFile.writeLine(rowID+ , +currLine.toString())
in the selected output file
    rowID ← rowID + 1
end for

/*Phase 2: Traverse all CSV file that contain the nodes and save their data in a separate node CSV file.*/
for all CSV Files with nodes nodeCSV do
    CSVLines ← nodeCSV.getDesiredColumns().getLines()
    for all CSVLines line do
        currNodeID ← line.getNodeID()
        if nodeIDs.contains(currNodeID) then
            NodeOutputFile.writeLine(line.toString())
        else
            Ignore the current line and proceed
        end if
    end for
end for
```

4.2.4 Algorithms Simulated. Perspectives has pre-implemented several algorithms applicable for Fraud Detection graphs. In this part of the report we present which algorithms we designed to run and provide figures about the results. As we stated before, because of the big size of the dataset, the tool could only load small subsets of it. Thus, we used the tool only for the visualizations (final results for the whole graph are presented in the next section), for a graph having around 4000 edges (using the sampling method). Profoundly, we chose to run graph algorithms that rank the nodes depending on how important they are, based in the incoming and outgoing links and the network that each node belongs. These algorithms are Eigenvector Centrality, Degree Centrality and Clustering. To

Algorithm 2 Sampling Algorithm: It takes a number of samples from the edges and constructs a subset of the graph

Require: $count \geq 1$ To be the number of samples to take from the dataset
Require: $NodeOutputFile \neq NULL$ To be the result file path of the Nodes
Require: $EdgeOutputFile \neq NULL$ To be the result file path of the Edges

```
edges ← CSV(edgesPath)
min ← 0
max ← edges.getLines().size()
rowID ← 0
randomRows ← generateRandomNums(min, max, count)
selectedEdges ← edges.selectRows(randomRows)
nodeIDs ← hashSet()

/*Phase 1: Keep the selected edges with a rowID in a
separate CSV file and save the node IDs in a set, to
find them later.*/
for all edges e in selectedEdges CSV file do
    currLine ← e.getLine()
    fromID ← currLine.getFromID()
    toID ← currLine.getToID()
    nodeIDs.add(fromID, toID)
    EdgeOutputFile.writeLine(rowID + , +currLine.toString())
in the selected output file
    rowID ← rowID + 1
end for

/*Phase 2: Traverse all CSV file that contain the
nodes and save their data in a separate node CSV
file.*/
for all CSV Files with nodes nodeCSV do
    CSVLines ← nodeCSV.getDesiredColumns().getLines()
    for all CSVLines line do
        currNodeID ← line.getNodeID()
        if nodeIDs.contains(currNodeID) then
            NodeOutputFile.writeLine(line.toString())
        else
            Ignore the current line and proceed
        end if
    end for
end for
```

run these algorithms we used the built-in analyzers of Tom Sawyer Perspectives tool as seen in figure 7.

[Eigenvector Centrality] Eigenvector Centrality is a measure of the influence of a node in a network. Relative scores are assigned to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. A high eigenvector score means that a node is connected to many nodes who themselves have high scores. After running this algorithm we noticed that nodes with high score are ORION HOUSE SERVICES

Algorithm 3 Reconstruction Algorithm: Begins from a source node and recreates it's network using a leveled BFS approach. This algorithm needs to be called from several source nodes.

Require: $depth \geq 1$ The depth of the paths. Usually this is set to 3, but it can be any number above 0.
Require: $NodeOutputFile \neq NULL$ To be the result file path of the Nodes.
Require: $EdgeOutputFile \neq NULL$ To be the result file path of the Edges.
Require: $startingNodeID \geq 0$ The ID of the source node.

```
edges ← CSV(edgesPath)
nodeIDs ← hashSet()
levelNodes ← list() of list()
firstLevelNode ← list()
currentLevelNodes ← list()
firstLevelNode.add(startingNodeID)
levelNodes.add(firstLevelNode)
levelBFS ← 0

/*Phase 1: Build the direct network of the given node
using BFS-like approach.*/
while levelBFS ≤ depth and not (currentLevelNodes ←
levelNodes.get(levelBFS)).isEmpty() do
    nextLevelNodes ← list()
    for currentNode in currentLevelNodes do
        nodeIDs.add(currentNode)
        neighbours ← CSV.groupBy(edges, currentNode).getIDs()
        for toID in neighbours do
            if not nodeIDs.contains(toID) then
                nextLevelNodes.add(toID)
            end if
            EdgeOutputFile.writeLine(currentNode, toID)
        end for
    end for
    levelNodes.add(nextLevelNodes)
    levelBFS ← levelBFS + 1
end while

/*Phase 2: Traverse all CSV file that contain the
nodes and save their data in a separate node CSV
file.*/
for all CSV Files with nodes nodeCSV do
    CSVLines ← nodeCSV.getDesiredColumns().getLines()
    for all CSVLines line do
        currNodeID ← line.getNodeID()
        if nodeIDs.contains(currNodeID) then
            NodeOutputFile.writeLine(line.toString())
        else
            Ignore the current line and proceed
        end if
    end for
end for
```

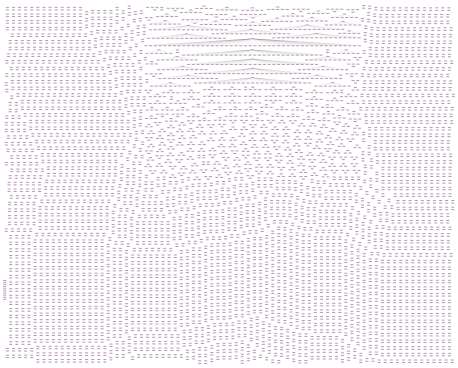

Figure 6: The complete visualisation of the subgraph, as loaded from the Perspectives previewer.

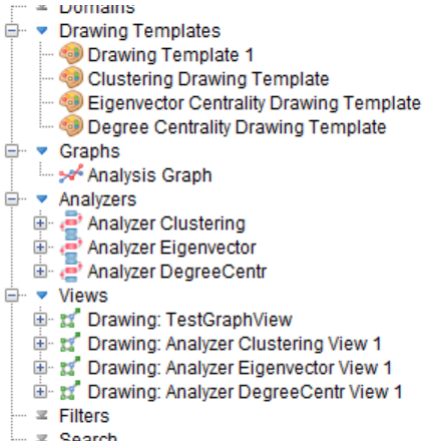

Figure 7: The analyzers used from the Perspectives, to apply the algorithms.

LIMITED, MOSSACK FONSECA CO. and OFFSHORE BUSINESS CONSULTANT (INT'L) LIMITED.

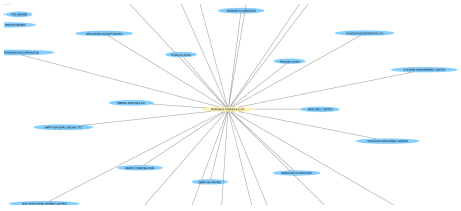

Figure 8: A section of the eigenvector centrality graph, loaded by the Perspectives previewer.

[Degree Centrality] Degree Centrality of a node is simply its degree—the number of edges it has. The higher the degree, the more central the node is. This can be an effective measure, since many nodes with high degrees also have high centrality by other measures and this is why both Eigenvector and Degree centrality gave the same results for the subset we loaded. Again, two "popular" nodes are ORION HOUSE SERVICES LIMITED and MOSSFON SUBSCRIBERS LTD.

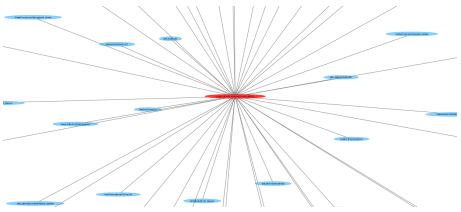

Figure 9: A section of the degree centrality graph, loaded by the Perspectives previewer.

[Clustering] Clustering is the task of grouping a set of objects in such way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). After the clustering, we noticed that important nodes are ORION HOUSE SERVICES LIMITED and MOSSACK FONSECA CO. A

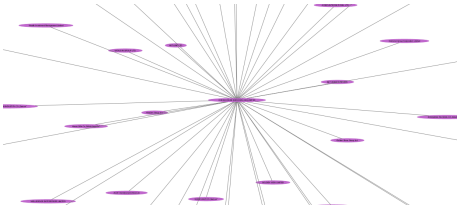

Figure 10: A section of the clustering graph, loaded by the Perspectives previewer.

more abstract representation of the visualization created by the TSP analyzers can be seen in figure 11.

4.3 Analysis using NetworkX

In order to offer a complete analysis over all nodes of the network, we decided to implement some algorithms using NetworkX Python API [6].

4.3.1 API Description. NetworkX is an open source Python API which has implemented several graph algorithms for many kinds of graphs, while it also supports data visualization. We decided to use it as it could load the data completely.

4.3.2 Loading the data. As mentioned above, the data for the nodes are splitted into some csv files, while the edges are kept in a separate csv file. Thus, we implemented an initialization procedure which loads the data to NetworkX (nodes and edges) and creates a dictionary for the node IDs and their names. The process is shown in algorithm 4.

4.3.3 PageRank. PageRank computes a ranking of the nodes in the graph based on the structure of the incoming and out-coming links. Although is was originally designed as a ranking algorithm for the web-pages of Google, newer versions of PageRank can aid problems from a wide domain of fields. Using PageRank to rank the nodes of our network can provide interesting results by showing the important nodes of the graphs regarding the link structure. Using NetworkX we managed to get the results for the most important

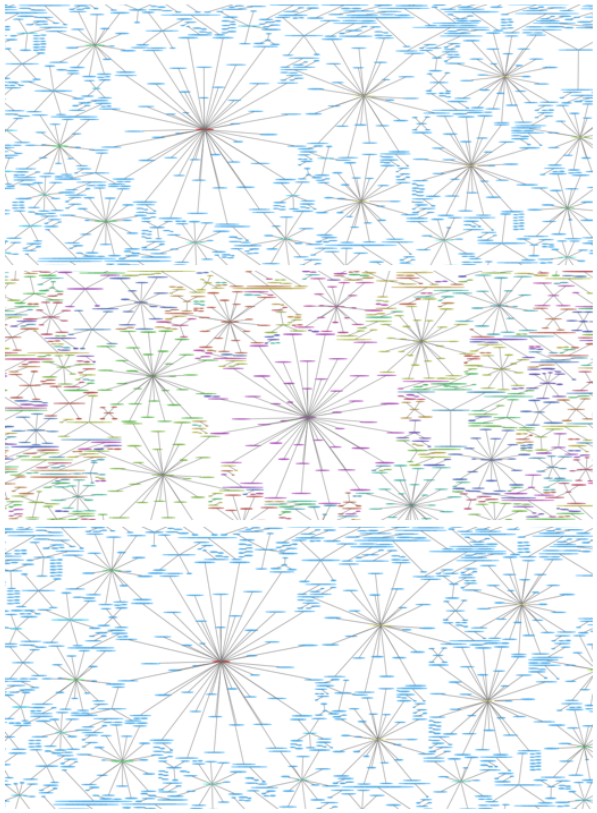

Figure 11: The eigenvector, clustering and degree centrality of the subgraph, loaded by the Perspectives previewer.

Algorithm 4 Initialization algorithm: It loads the data from the csv and created the graph and a dictionary.

```
IDs2Names ← dictionary()
Graph ← nx.directedGraph()
NodeFiles ← [officer.csv, entity.csv, intermediary.csv, address.csv]
EdgeFile ← edge.csv

/*Add all the nodes*/
for file in NodeFiles do
    for row in file.rows() do
        rowID ← row.getID()
        rowName ← row.getName()
        Graph.addNode(rowID)
        IDs2Names.addMapping(rowID, rowName)
    end for
end for

/*Add all the edges*/
for row in EdgeFile.rows() do
    fromID ← row.getFromNodeID()
    toID ← row.getToNodeID()
    Graph.addEdge(fromID, toID)
end for
```

nodes of the network as shown in algorithm 5. It needs to be stated that we decided to get only the nodes that are actual entities of the graph (i.e. we excluded the address nodes from the final results). For our computations, we simulated the process for 100 iterations and damping parameter of 0.85. Our final results are listed in the table below.

Node	Pagerank
ACCELONIC LTD. (entity)	0.00077
VELA GAS INVESTMENTS LTD. (entity)	0.00060
Dale Capital Group Limited (entity)	0.00034
BOB AGENTS LIMITED (entity)	0.00026
GNG LTD. (entity)	0.00023
INGELSA LTD. (entity)	0.00020
KEINES INVESTMENTS LIMITED (entity)	0.00019
MAGN DEVELOPMENT LIMITED (entity)	0.00019
SITEK GROUP LIMITED (entity)	0.00018
3 DIP S.A. (entity)	0.00017

Algorithm 5 PageRank NetworkX: Running PageRank using NetworkX API.

```
dictionary ← nx.pagerank()
results ← dictionary.sort().getTopK()
printResults(results)
```

4.3.4 Eigenvector Centrality. Eigenvector Centrality is a measure of the influence of a node in a network. Relative scores are assigned to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. The result of this algorithm is the association of a score to each node. A high score value means that a node has connections to many other nodes that also have high scores. This way we can also locate important nodes of the graph. PageRank (implemented in the previous subsection) is a variation of this method. Using NetworkX we managed to get the results using algorithm 6. Our results are presented in the following table. As PageRank is a variation of this algorithm, it is normal to expect some results to be the same. Again for this approach, we decided to rule out the nodes that correspond to addresses.

Node	EV Centrality
ACCELONIC LTD. (entity)	0.05063
VELA GAS INVESTMENTS LTD. (entity)	0.02479
Dale Capital Group Limited (entity)	0.02253
MAGN DEVELOPMENT LIMITED (entity)	0.01247
DigiWin Systems Group Holding Limited (entity)	0.01056
GNG LTD. (entity)	0.01016
BOB AGENTS LIMITED (entity)	0.00865
INGELSA LTD. (entity)	0.00845
ROCKOVER RESOURCES LIMITED (entity)	0.00840
LUK FOOK (CONTROL) LIMITED (entity)	0.00784

4.3.5 Degree Centrality. Another method to locate the important nodes of the graph is to calculate the fraction of nodes that the source node is connected to, either by outgoing or incoming links.

Algorithm 6 EigenVector Centrality NetworkX: Running EigenVector Centrality using NetworkX API.

```
dictionary ← nx.pagerank()
results ← dictionary.sort().getTopK()
printResults(results)
```

Using NetworkX we managed to get results by the process depicted in algorithm 7. The results are depicted in the following table.

Node	Degree
ORION HOUSE SERVICES (HK) LIMITED (intermediary)	0.01254
MOSSACK FONSECA CO. (intermediary)	0.00780
PRIME CORPORATE SOLUTIONS SARL (intermediary)	0.00736
OFFSHORE BUSINESS CONSULTANT (INT'L) LIMITED (intermediary)	0.00732
MOSSACK FONSECA CO. (SINGAPORE) PTE LTD. (intermediary)	0.00695
MOSSFON SUBSCRIBERS LTD. (officer)	0.00694
CONSULCO INTERNATIONAL LIMITED (intermediary)	0.00566
MOSSACK FONSECA CO. (GENEVA) S.A. (intermediary)	0.00534
MOSSACK FONSECA CO. (U.K.) LIMITED (intermediary)	0.00454
MOSSACK FONSECA CO. (PERU) CORP. (intermediary)	0.00367

Algorithm 7 Degree Centrality NetworkX: Running PageRank using Degree Centrality API.

```
dictionary ← nx.degree()
results ← dictionary.sort().getTopK()
printResults(results)
```

Algorithm 8 Clustering NetworkX: Running PageRank using Clustering API.

```
dictionary ← nx.clustering()
results ← dictionary.sort().getTopK()
printResults(results)
```

4.3.6 Clustering. Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It is a main task of exploratory data analysis, and a common technique for statistical data analysis, used in many fields, including pattern recognition, image analysis, information retrieval etc. The given result is a group of clustering coefficients, which are a measure of the degree to which nodes in a graph tend to cluster together.

Node	Centrality
Aurelio Ledergerber (officer)	0.50000
FIDUCIAIRE PIRMIL SA (officer)	0.50000
Eiger Advisory Group Inc. (entity)	0.08333
WHELAN PROPERTIES LIMITED (entity)	0.02381

4.3.7 What do these results offer? The algorithms we run over our network (especially the rankings) can give us a better understanding on how the network works, which nodes are important etc. Our ranking results can show us nodes with high fraud activity over the fraud detection network, and can give us input for further investigation on the actions done by them. This idea is the base of our next section: How can we go one step further over the analysis of the graph, using different analysis methods.

5 EMBEDDINGS APPROACH

In this section we present our approach regarding the Embeddings process creation and exploitation.

[General Idea] We can use NLP (Natural Language Processing) libraries using custom corpus to produce vectors from the words than are inside the text. The pre-trained neural network of the NLP library will put the vectors of the words that are semantically close to be also close in the vector space. This way, we can aid similarity search problems. For example, we could iterate over the Fraud detection data and find the most similar off-shores based on their vectors.

[word2vec] To generate these vectors we exploit word2vec [8], a shallow two-layer neural network model that uses a single hidden layer for producing word embeddings. The input is a text (in our case triples of fraud detection data) while the output is a multidimensional vector (usually with over a hundred of dimensions) for each unique word (in our case nodes) that appears in the text. The vectors it produces are based on the co-occurrence of words (in our case node names and link types) in context windows of a given size, and it aims at grouping the vectors of similar words (based on their co-occurrence) closely in the vector space. The recommended value of the size of context window depends on the word2vec model, e.g., in the skip-gram model, that value is usually around 10. However, in our case we use a context window of size 3, since for an entity we produce one sentence for each of its triples (i.e., each sentence contains 3 words), which means that we capture similarities based on the co-occurrence of nodes and the links between them.

5.1 Graph Embeddings

The basic challenge when using embeddings for graphs is to transform the graph into a text corpus, in order to load it to the corresponding library. The most used method is to represent the graph as a set of triples, where each triple is in the form of NODENAME LINKTYPE NODENAME. Such files can be produced easily and be used as a corpus. In the following sections, we describe how we transformed the Fraud Detection graph into corpus and how we trained the model.

5.2 Vocabulary Creation

For the vocabulary, we developed an algorithm that transforms the graph into sentences of triples representing the connections between the nodes. The challenge was to gather all the node files and replace the nodeIDs with their actual names. A summarization of the process is shown in algorithm 9.

5.3 Embeddings Training

5.3.1 Skip-gram model. For computing the embeddings, we use the skip-gram model of word2vec. We decided to use this model instead of the Continuous-Bag-of-Word (CBOW) model of word2vec, since in most cases Skip-gram outperforms CBOW for large datasets. Skip-gram model uses unsupervised learning techniques for finding the most related words (in our case Fraud Detection entities) for a given word, that is, it uses a target word for predicting the context

Algorithm 9 Vocabulary Creation Algorithm: It produces a text corpus from the edges of the graph

Require: $corpusPath \neq NULL$ To be the output file for the corpus

```
/*Phase 1: Create a mapping for the nodeIDs with their names*/
IDs2names ← HashMap < Int, String > ()
for currCSV in CSVfiles do
    lines ← currCSV.getDesiredColumns().getLines()
    for line in lines do
        lineID ← line.getID()
        lineName ← line.getName()
        IDs2Names.put(lineID, lineName)
    end for
end for

/*Phase 2: Create the corpus using the edges and the map*/
edges ← CSV(edgesPath)
lines ← edges.getLines()
for line in lines do
    fromID ← line.getFromID()
    toID ← line.getToID()
    link ← line.getLink()
    fromIDName ← IDs2Names.get(fromID)
    toIDName ← IDs2Names.get(toID)
    corpusPath.writeLine(fromIDName + link + toIDName)
end for
```

words. In particular, given a sequence of training words, say $word_1$, $word_2$, ... , $word_T$, and a context window of size c, the skip-gram model tries to maximize the average log probability, which follows:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(word_{t+j}|word_t). \tag{1}$$

The probability $p(word_{t+j}|word_t)$ is computed using the softmax function:

$$p(word_{t+c}|word_t) = \frac{exp(v'^{T}_{word_{t+c}} v_{word_t})}{\sum_{m=1}^{|U|} exp(v'^{T}_{words_m} v_{words_t})}. \tag{2}$$

In the above formula, v' and v are the input and the output vectors of a word $word$, whereas $|U|$ is the number of unique words in our vocabulary (text corpus).

5.3.2 Implementation. Our implementation exploits the SkipGram model of word2vec to create the embeddings, as shown in the algorithm 10.

5.4 Similarity Search over Fraud Detection Data

5.4.1 Similarity Search. One of the most important data mining tasks is similarity search i.e. "Given an embedding of a word in a multidimensional space, give me the most similar words of it using the vector space, Euclidean distance and cosine similarity". The

Algorithm 10 Embeddings Training Algorithm

Require: $corpusPath \neq NULL$ To be the input file for the corpus
Require: $datasetPath \neq NULL$ To be the output file for the embeddings

```
vectors ← model.trainWithDefaults(corpusPath)
vectors.removeLinks()
vectors.removeAddresses()
RDFsim.createDataset(vectors)
```

aforementioned term is contextualized for embeddings similarity search.

5.4.2 Similarity Method for computing the K most similar nodes. For finding the K most similar entities to a given entity, we employ the cosine similarity score of their vectors. The cosine similarity between two vectors v'_{vec_1} and $v'(vec_2)$ (each vector corresponds to a single word) is defined as:

$$\cos(\theta)_{vec_1,vec_2} = \frac{v'_{vec_1}\cdot v'_{vec_2}}{\|v'_{vec_1}\|\|v'_{vec_2}\|}. \tag{3}$$

The score ranges from -1 (i.e., two vectors are exactly opposite) to 1 (i.e., two vectors are exactly the same). To compute the K most similar entities to a given word $word$, we rank the rest words according to their $\cos(\theta)_{vec,vec'}$ score in descending order, and we keep the words placed in the first K positions in the ranking (i.e., the K most similar words to $word$).

5.5 RDFsim Engine Exploitation

5.5.1 RDFsim. RDFsim [5] is a similarity search engine over semantic Knowledge Graph data (mainly from DBpedia) developed and published by Manos Chatzakis, Michalis Mountantonakis and Yannis Tzitzikas in Summer 2021. Although it is designed and tuned for semantic data, we managed to port custom data using some of the RDFsim+ feautures. These features can load any corpus (even non-semantic data) and their embeddings, and manage to create similarity networks. RDFsim exploits the embeddings by implementing a BFS-like algorithm in order to build and visualize similarity graphs using a JavaScript framework. RDFsim is an open-source, under development tool [2, 3]. The tool manages to create the networks with it's own BSF-like method, shown in algorithm 11. An outline of the embedding creation and RDFsim exploitation is presented in figure 12.

5.5.2 How can RDFsim be used for our data. After producing the embeddings and the RDFsim-supported dataset, we can load our data directly into the search engine. Then, we can start doing queries on the data. Our main idea was to provide another method for finding relationships between companies and people that are inside the dataset. Using RDFsim, we can create large similarity networks of companies and people, and if the parameter tuning is good enough, we can find relations between companies and people that are not easily discoverable through classic graph analytics method. For example, if two companies have relationships with people that have some kind of relationships (through other nodes), word2vec understands that and the vectors of the companies will be closer into

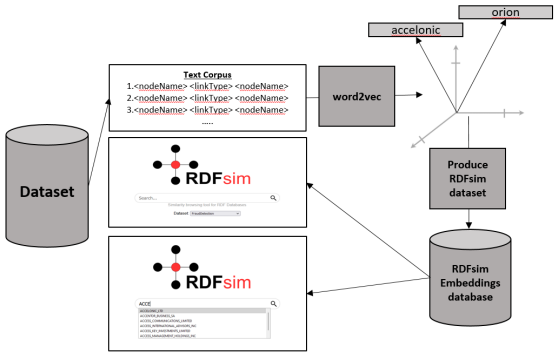

Figure 12: The process of producing the graph embeddings and the RDFsim exploitation

Algorithm 11 Creating the similarity graph G for the selected entity e_{sel}.

Require: Entity e_{sel}, depth L, number of similars K, Entity Index ei

```
The similarity graph starting G from entity e by using specific L and K
Initialize Graph
Graph G{
 V ← ∅,
 E ← ∅
}
level ← 1
V(level) ← ∅
Add the selected entity as the first node
G.V ← G.V ∪ {e_sel}
V(level) ← V(level) ∪ {e_sel}
Follow a BFS-like approach
while level ≤ L and V(level) ≠ ∅ do V(level + 1) ← ∅
    Traverse each node e of the current level
    for all e ∈ V(level) do
        similars(e, K) ← ei.getSimilarEntities(e, K)
        Traverse the top-K to similar entities of v and create the corresponding nodes/edges (if they do not exist)
        for all e' ∈ similars(e, K) do
            if e' ∉ G.V then
                G.V ← V ∪ {e'}
                G.E ← G.E ∪ {e, e'}
                V(level + 1) ← V(level + 1) ∪ {e'}
      {e, e'} ∉ G.E
                G.E ← G.E ∪ {e, e'}
            end if
        end for
    end for
end while
level ← level + 1
return G
```

the vector space. Also, using word2vec and RDFsim algorithms, we managed to load the whole dataset and produce embeddings for all the companies that have a significant number of relationships. We decided to do that because it has really efficient results, and because we decided that it is better to give results for the most important nodes of the dataset (i.e. nodes with many connections). Another good point for using RDFsim is that the parameters of the training can be easily tuned, while the model is scalable for better and more powerful systems. Some use cases about the idea of implementing knowledge graph embeddings algorithms to fraud detection graphs are presented in the next section.

5.5.3 Examples and Use Cases. Our port of the data and the local build of RDFsim is easily accessible though it's official development GitHub repository. In this section, we present some basic examples on how we can interact with the Fraud Detection data. For the use cases, we decided to examine the similarity networks of the nodes that had high scores in the algorithms implemented from NetworkX. Thus, we will explore the similarity graphs of the following nodes:

- ACCELONIC LTD. (entity node)
- VELA GAS INVESTMENTS LTD. (entity node)
- ORION HOUSE SERVICES (HK) LIMITED (intermediary node)
- (at this point, we can search and create the similarity network of any node we want)

[ACCELONIC LTD.] For this entity, the direct network leads us to the corresponding star-graph, consisting of nodes that are probably similar to our given entity. Indeed, RDFsim can also create

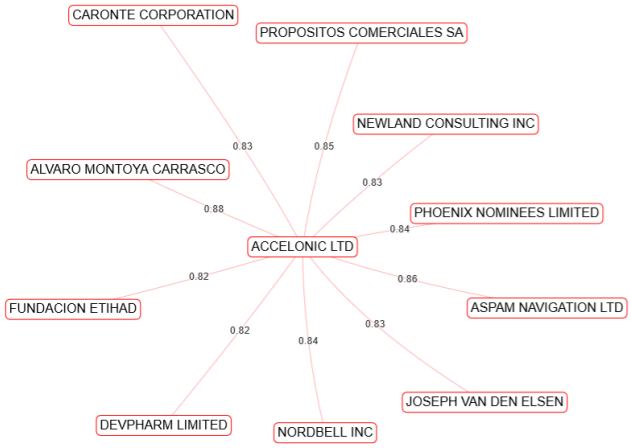

Figure 13: Direct similar nodes of the entity ACCELONIC LTD.

bigger networks, called similarity networks, for a given entity. In the corresponging figure, we have created the similarity graph for ACCELONIC LTD. We can use this graph to discover connections between nodes that are not necessarily connected in the original dataset.

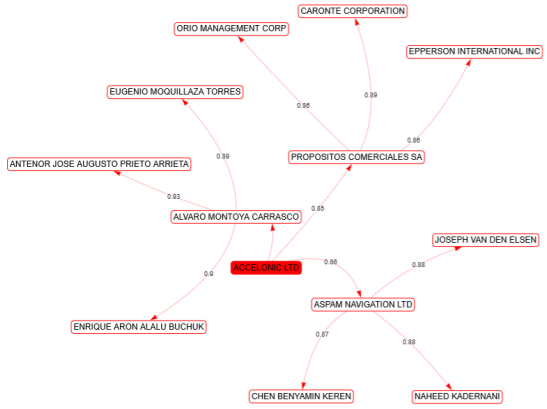

Figure 14: The similarity network of ACCELONIC LTD. The direction of the edges is the flow of similarity, while the labels of the edges are the cosine similarity between the nodes

[VELA GAS INVESTMENTS LTD.] For this entity, we repeat the same steps as previously.

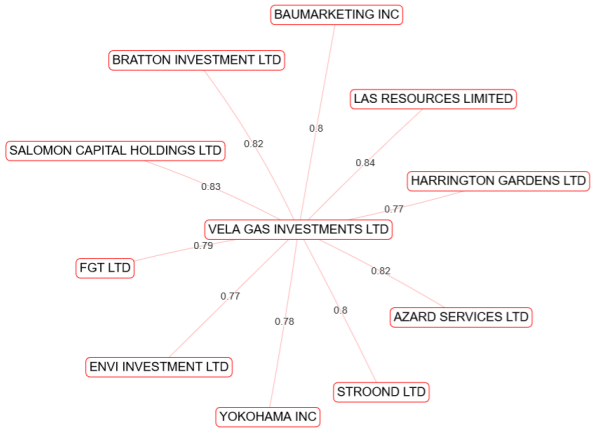

Figure 15: Direct similar nodes of the entity VELA GAS INVESTMENTS LTD.

[ORION HOUSE SERVICES (HK) LIMITED] For this entity, we repeat the same steps as previously.

5.5.4 Understanding the results. Lets look at the similarity network of ACCELONIC LTD. Using this network, we can discover connections between offshores and other entities that were not clear before. For example, we see that our current node has high similarity score with ASPAM NAVIGATION LTD. This result could mean that these two nodes may have connections to similar entities, officers, companies and more. Given that our base data are dense and might have many connections between nodes, it could be difficult to discover such connections, or even worse, such connections

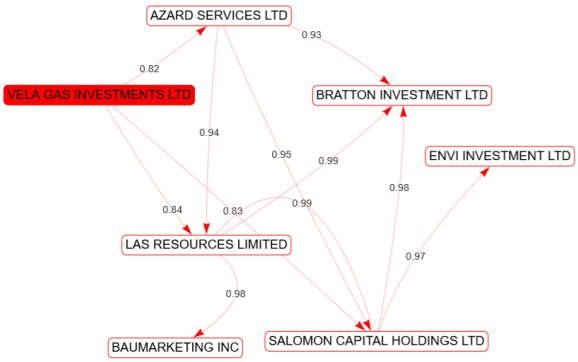

Figure 16: The similarity network of VELA GAS INVESTMENTS LTD. The direction of the edges is the flow of similarity, while the labels of the edges are the cosine similarity between the nodes

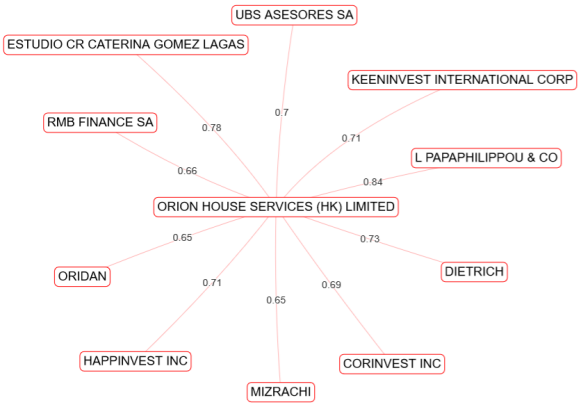

Figure 17: Direct similar nodes of the entity ORION HOUSE SERVICES (HK) LIMITED

might not even exist in the starting dataset. Exploiting graph embeddings for such graphs could actually offer a new way to discover relationships between the nodes of the graph, group offshores with same fraudulent activity and more, as shown in figure 19.

6 CONCLUSION

This report presented multiple methods to analyze the data harvested from the fraud detection graph of the Panama papers leak. We tried to develop methods to efficiently study subsets of big datasets, while we managed to get results about important entities of the graph using graph analysis algorithms and visualizations. Last but not least, we proposed an alternative method to interact with the data and find patterns between nodes of the graph through embeddings and similarity search, while we proposed a port of our data in the RDFsim search engine. All the work done and resources used are available through the repository of the report [4].

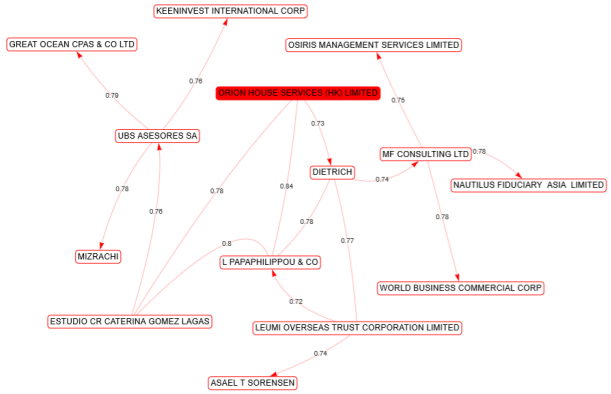

Figure 18: The similarity network of ORION HOUSE SERVICES (HK) LIMITED. The direction of the edges is the flow of similarity, while the labels of the edges are the cosine similarity between the nodes

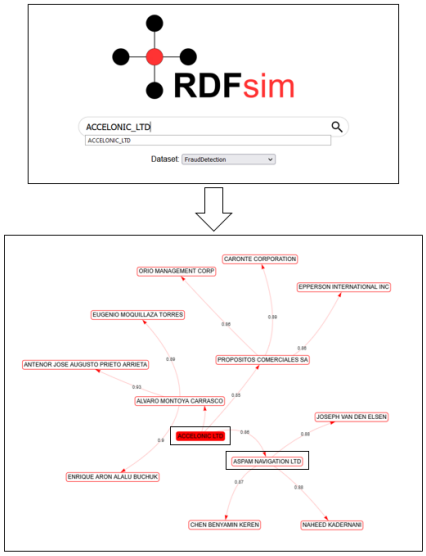

Figure 19: Understanding the results for the similarity network of ACCELONIC LTD.

REFERENCES

[1] Manos Chatzakis. *CSVEditor*. https://github.com/MChatzakis/CSVEditor. 2020.

[2] Manos Chatzakis. *RDFsim*. https://github.com/MChatzakis/RDFsim. 2021.

[3] Manos Chatzakis. *RDFsim-server*. https://github.com/MChatzakis/RDFsim-PublicVersion. 2021.

[4] Manos Chatzakis and Eva Chamilaki. *PanamaPapersAnalysis*. https://github.com/MChatzakis/Panama-papers-analysis. 2021.

[5] Manos Chatzakis, Michalis Mountantonakis, and Yannis Tzitzikas. "RDFsim: Similarity-Based Browsing over DBpedia Using Embeddings". In: *Information* 12.11 (2021). ISSN: 2078-2489. DOI: 10.3390/info12110440. URL: https://www.mdpi.com/2078-2489/12/11/440.

[6] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[7] ICIJ. *offshoreleaks-data-packages*. https://github.com/ICIJ/offshoreleaks-data-packages.

[8] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[9] Frederik Obermaier and Bastian Obermayer. *The Panama Papers: Breaking the story of how the rich and powerful hide their money*. Simon and Schuster, 2017.