Graph Model Explainer Tool

Yudi Zhang, Naveed Janvekar, Phanindra Reddy Madduru, Nitika Bhaskar
yudzhang,njjanvek,maddurup,nitikb@amazon.com
Amazon Inc., USA

Abstract

Graph Neural Networks (GNNs) have gained popularity in various fields, such as recommendation systems, social network analysis and fraud detection. However, despite their effectiveness, the topological nature of GNNs makes it challenging for users to understand the model predictions. To address this challenge, we built a user-friendly UI to visualize the most important relationships for both homogeneous and heterogeneous static graphs models, which a post-hoc explanation technique called GNNExplainer is implemented. This UI can be applied to a wide range of applications that use graph models. It offers an intuitive and interpretable way for users to understand the complex relationships within a graph and how they influence the model's predictions.

Keywords: Graph Neural Networks, GNNExplainer, Visualization

ACM Reference Format:
Yudi Zhang, Naveed Janvekar, Phanindra Reddy Madduru, Nitika Bhaskar. 2023. Graph Model Explainer Tool. In *Proceedings of 19th International Workshop on Mining and Learning with Graphs (MLG'23).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Graph Neural Networks (GNNs) are achieving ever-increasing performance on many artificial intelligence tasks. It drives several real-world applications including drug-discovery, recommendation systems, social networks, and fraud detection etc. Unfortunately, a major limitation of graph model is that they are not amenable to interpretability since usually graphs contain topological information represented as feature and adjacency matrices, and are less intuitive than images and texts. Understanding the graph model often requires domain knowledge. However, without reasoning the underlying mechanisms behind the predictions, GNNs cannot be fully trusted. We need to provide convincing evidence to support the predictions. In order to make GNNs fully effective, it is desired to provide both accurate predictions and human-intelligible explanations.

In the past few years, tree-based model interpretation methods [4, 2] are popular and used by many researchers. For example, TreeSHAP [2] is designed for tree-based model interpretation. It computes Shapley [6] values for tree-based machine learning models, which provide a way to fairly allocate the contribution of each feature to the prediction made by the model. It helps interpret the model's predictions and gain insights into the importance of different features in the decision-making process. Recently, explainability of GNNs has achieved significant progress. Many post hoc techniques [11, 8, 13] have been developed to explain predictions, giving rise to the area of GNN explainability. PGExplainer [3] trains a mask predictor to generate a discrete masks for learning the importance. SubgraphX [14] uses the Shapley value and onbtain the most important subgraphs with Monte Carlo Tree Search (MCTS). These methods focus on different aspects of the graph models and provide different views to understand these models. They typically address a set of queries, such as determining the relative importance of input edges, identifying critical input nodes, evaluating the significance of node features, and uncovering optimal graph patterns for maximizing the prediction of a particular class. [12] The explainability of a prediction model is important for making it reliable. In addition, it sheds light on potential flaws and generates insights on how to further refine a model.

To provide transparent and easy to understand explanations for the predictions of complex graph models, it is desired to have a user-friendly tool that allows users to unpack a model's black box through interactive exploration of graphs and visualizations. Many graph database visualization tool exists [5, 7], however, they do not provide explanations for the connections that contribute to predictions in graph models. To address this, we propose a new tool that utilizes GNNExplainer [12] as the backend explainer model and displays the most critical connections leading to model predictions. Our focus is on providing explanations for predictions, thereby improving the comprehensibility, this include both homogeneous and heterogeneous graph models.

Our main contribution in the paper is to build an UI, which provides a more interpretable way to understand complex graph models and identify the significant connections that contribute to predictions. We expect this UI provide deeper

MLG'23, Aug 2023, Long Beach, CA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

model understanding for model developers and improve manual investigation efficiency for non technical person.

2 Method

2.1 GNNExplainer

GNNExplainer is one of the most popular methods in explaining graph model predictions as well as the very first methods to interpret GNN. It learns soft masks for edges and node features to explain the predictions via mask optimization. To obtain masks, it randomly initializes soft masks and treats them as trainable variables. Then GNNExplainer combines the masks with the original graph via element-wise multiplications. Next, the masks are optimized by maximizing the mutual information between the predictions of the original graph and the predictions of the newly obtained graph. Suppose we want to explain a prediction Y of a given node v_i and define the GNN model as f_θ. GNNExplainer acts to provide a local interpretation $G_S = (\boldsymbol{A}_S, \boldsymbol{X}_S)$, where $\boldsymbol{A}_S$ is the relevant adjacency matrix and $\boldsymbol{X}_S$ is the relevant subgraph structure. The method will find the optimal explanation G_S, which has the maximum Mutual Information (MI) with prediction Y:

$$MI\left(Y, (\boldsymbol{A}_S, \boldsymbol{X}_S)\right) := H(Y) - H\left(Y \mid \boldsymbol{A} = \boldsymbol{A}_S, \boldsymbol{X} = \boldsymbol{X}_S\right),$$

where H is the entropy. Since f_θ is already trained, the entropy $H(Y)$ is also fixed. That is to say, the explanation for v_i's prediction Y is a subgraph $\boldsymbol{A}_S$ and features $\boldsymbol{X}_S$ that minimize the uncertainty of pretrained GNN f_θ when the message-passing is limited to G_S:

$$\max_{(\boldsymbol{A}_S, \boldsymbol{X}_S)} MI\left(Y, (\boldsymbol{A}_S, \boldsymbol{X}_S)\right) \approx$$

$$\min_{(\boldsymbol{A}_S, \boldsymbol{X}_S)} -\sum_{c=1}^{C} I\left[\widehat{y}_l = c\right] \ln f_\theta\left(\boldsymbol{A}_S, \boldsymbol{X}_S\right)_{v_i}^c$$

where c is the number of class. In practice, the objective function of GNNExplainer can be optimized to learn adjacency mask matrix $\boldsymbol{M}_A$ and feature selection mask matrix $\boldsymbol{M}_F$ in the following equation:

$$\min_{(\boldsymbol{M}_A, \boldsymbol{M}_F)} \mathcal{L}\left(f_\theta, \boldsymbol{A}, \boldsymbol{M}_A, \boldsymbol{X}, \boldsymbol{M}_F, v_i, \widehat{y}_l\right) :=$$

$$-\sum_{c=1}^{C} I\left[\widehat{y}_l = c\right] \ln f_\theta\left(\boldsymbol{A} \odot \sigma\left(\boldsymbol{M}_A\right), \boldsymbol{X} \odot \sigma\left(\boldsymbol{M}_F\right)\right)_{v_i}^c$$

where $\odot$ denotes element-wise multiplication, and σ is the sigmoid function that maps the mask to $[0,1]^{n \times n}$. After the optimal mask is obtained $\boldsymbol{A}_S = \boldsymbol{A} \odot \sigma\left(\boldsymbol{M}_A\right)$ use a threshold to remove low values, similarly $\boldsymbol{X}_S = \boldsymbol{X} \odot \sigma\left(\boldsymbol{M}_F\right)$.

2.2 Heterogeneous Graph Neural Network

Suppose we have a heterogeneous graph denoted as $G = \{V, E, R\}$, where $v_i \in V$ is a node, $e_i = \left(v_i, r, v_j\right) \in E$ is a labeled relation in which the label is defined as the label of source node v_i and $r \in R$ is a relation type. A single layer of heterogeneous GNN aggregates information for every node from their neighbors and updates the node's hidden state with a linear transformation followed by a nonlinear elementwise activation function as in:

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} \boldsymbol{W_r}^{(l)} h_j^{(l)} + \boldsymbol{W_0}^{(l)} h_i^{(l)}\right)$$

where $h_i^{(l)} \in R^{d_l}$ denotes the hidden state of v_i in the l-th layer of heterogeneous GNN with d_l being the hidden state dimensions in layer l. For $l = 0, h_i^{(0)} = 0$ if v_i is an attribute node; if v_i is a node, $h_i^{(0)} = X_i$ is the node feature of i. A relation-specific scaler $c_{i,r}$ is applied at v_i, which is typically set as the r-degree of v_i, i.e., $c_{i,r} = |N_i^r|$ with N_i^r being the set of neighbors of v_i under the relation r. $\boldsymbol{W_r}^{(l)}, \boldsymbol{W_0}^{(l)} \in R^{d_l \times d_{l+1}}$ are learnable parameters in the l-th layer for interaction with relations of type r as well as within themselves, respectively. This formula is an extension of the classical GCN [1].

2.3 HGNNExplainer

To apply the GNNExplainer to a heterogeneous graphs, where each relation weight matrix $\boldsymbol{W_r}$ represents the nodes' interaction with relations of type r, the approach taken here is to use masks $\boldsymbol{M}_{\boldsymbol{w_r}}$ on $\boldsymbol{W_r}$ to learn the importance (masks). When we developed this tool, there was no pre-existing code available for this particular extension. As a result, we have provided the following pseudo code to outline our approach:

Algorithm 1: Implementation of (Heterogeneous GNN) HGNNExplainer.

```
Class HGCNLayer(nn.Module):
    def __init__(self, in_size, out_size, etypes,
     dropout=0):
        super().__init__()
        self.weight = nn.ModuleDict(name:
          nn.Linear(in_size, out_size) for name in etypes)
    def forward(self, graph, feat, eweight=None):
        func = {}
        for srctype, relation, dsttype in
         graph.canonical_etypes:
            wh = self.weight[relation](feat[srctype])
            graph.nodes[srctype].data[f'wh_relation'] =
              wh
            if eweight is None
                func[relation] = (fn.copy_u(f'wh_relation',
                  'm'), fn.mean('m', 'h'))
            else
                graph.edges[relation].data['w'] =
                  eweight[relation]
                func[relation] =
                  (fn.u_mul_e(f'wh_relation', 'w', 'm'),
                  fn.mean('m', 'h'))
            graph.multi_update_all(func, 'sum')
            return {ntype: graph.nodes[ntype].data['h']
              for ntype in graph.ntypes if 'h' in
              graph.nodes[ntype].data}
```

3 Visualization

Visualization is crucial for understanding and interpreting graph models. Graph models are complex and can capture intricate relationships among nodes and edges in a graph, making it difficult to comprehend their inner workings without visualization. Visualization provides a way to visually inspect the learned representations

from the graph and infer the model's behavior. It also facilitates error analysis and model debugging, which are essential for improving model performance. Several literature reviews have highlighted the importance of visualization in graph models. For example, a review of GNN applications in biology [15] emphasized the need for visualization to interpret and communicate the results of GNN models. Similarly, a survey of GNN applications in social network analysis [9] identified visualization as an essential component of GNN model interpretation.

The backend of the UI utilizes AWS services including Amazon S3 (Simple Storage Service), AWS Lambda, AWS Sagemaker to store and process the explanations. And we use Python package **pyvis** (>= 0.3.2) for graph visualizations. To provide users some interactive functionalities, we use **flask** (>= 2.2.3), which is a popular web framework written in Python to build this visualization tool and APIs. Specifically, we build a backend pipeline for data processing, graph model explanation and visualization, and then use flask's template engine to create a frontend user interface for users to interact with the data. Figure 1 shows the UI pipeline, the detailed workflow follows

1. Users bring their own trained model, including model class in a python file and graph. We currently support graph and model deployed with **DGL** >= 0.9.1.
2. Lambda function is triggered and invoke a docker image deployed on Sagemaker. Pre-implemented explainer model on our backend is implemented for graph nodes, i.e. finding the most important edges that contributes to the node prediction. Subgraph visualizations and some other customized meta data will be produced internally.
3. The explainer model dumps the output produced in the previous step to a S3 bucket.
4. Users put a node id to the UI, and the corresponding graph visualization will be displayed. Other interactive functionalities such as filter nodes and edges are enabled as well.

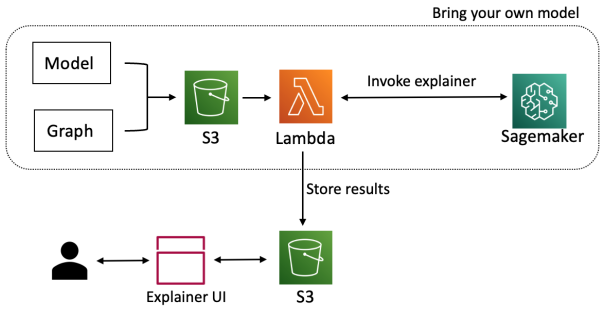


Figure 1. UI pipeline overview.

Noticeable, step 2 and 3 can be fully integrated to the backend of the UI (refers to the code part that uses **flask**), where the explainer model will be trained directly once user request to visualize some nodes. This will resolve a scalability issue that when the graph is too big, pre-computing all the explanations would be time consuming.

4 Applications

4.1 Fraud Detection with RGCN Model

Many e-commerce companies work on developing models for finding fraud entities like sellers or buyers based on their relationships. They use advanced graph models like Relational Graph Convolution Network (RGCN) and temporal graph neural network (TGN) to improve the predictions. We onboard one e-commerce fraud detection RGCN model to our UI and help the investigators with several goals:

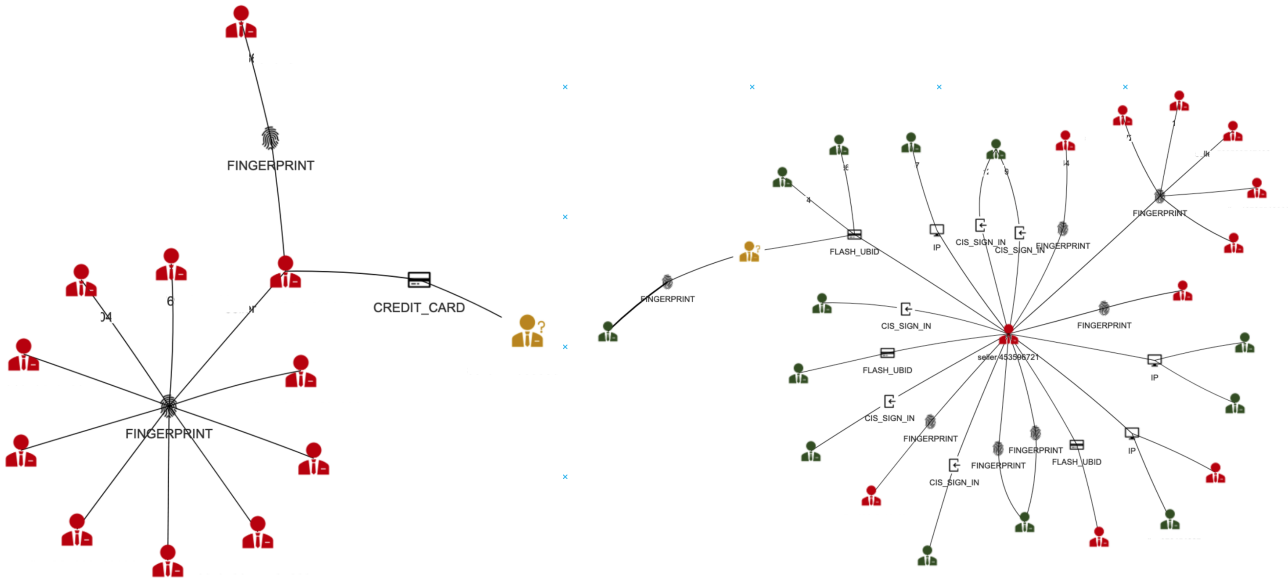
1. Help with manual investigations: the tool has a wide range of applications for the business team, such as handling escalations and audits for graph models, analyzing a graph network's spread, detecting problematic edges or relations, identifying new patterns of suspicious behavior, and examining aggregated data for person that necessitate manual investigations. By providing insights into an entity's graph structure, this tool can improve overall investigation processes.
2. Improve bad actor recall: although RGCN performs well in general, some fraudsters are still missed. Providing important neighbors of each person in the graph enables investigators to identify underlying relationships between predicted bad and good person. If predicted good person is strongly related to bad person, such suspicious person can be sent for manual investigations.

The graph model ingests twelve edge types, including anonymized credit card, and anonymized bank account etc. The main goal of the explainer model is to keep the top few important edges (based on their contribution to final model prediction), which reveals the critical relations among person. Intuitively, importance means if we remove an edge, the more important an edge, the less accurate of the model's predictions. Therefore, these edges should be kept when making the explanations.

Follow the method section, in this paper we illustrate the tool with 3-hop connections for a person (here the connection is person-person interaction, back to RGCN model, we would have a 6-hop connection heterogeneous graph), then the learned person-relation edge mask $\boldsymbol{M}_{w_r}$ can be obtained. Given a threshold t (default 0.7), normalized edge masks with values bigger than that threshold will be considered as important and kept. Then **pyvis** is used to visualize the graph, an example is shown in Figure 2. We provide this interactive graph where the interested node will be highlighted in yellow and the other important edges identified by RGCNExplainer will be shown as well. In these two visualizations, both interested person are predicted as high risk with probability greater than 0.95, however, the explanations for Figure 2b indicates further investigations need to be carried as most of the connections are linked to good (green) person. Apart from graph, we show node level metadata such as number of relations to better demonstrate the explanations as well as some filter and selection options to filter the graphs (see Appendix 3 for examples). More functionalities and customization can be added upon requested.

4.2 Entity Cluster Visualization

Visualizing clusters and full entity relations are also valuable because it allows people to easily see the patterns and relationships between data points within a cluster. This can make it easier to



(a) (b)

Figure 2. Interactive graphs showing the important connections identified by the explainer (importance is determined by a threshold). Red means bad, green means good, and the yellow one with a question mark is the interested person. Additional information such as predicted probability, and label can be found by clicking the icons.

interpret the results of a clustering algorithm and gain insights that may not have been immediately apparent from the raw data. For example, by visualizing the clusters, one could see that a particular segment of customers is highly correlated with certain products or services, which could inform product development or sales strategies. With the architecture already developed for GNNExplainer, we further generalize this tool for visualizing entity clusters (see Appendix 4 for an example) as well as holistic views of entities. Users can bring their own clusters of data in a csv format and use the tool to make more informed decisions by uncovering hidden patterns and relationships.

5 Discussion

In this paper, we have created a novel visualization tool that utilizes GNNExplainer [11] to uncover the inner workings of Graph Neural Networks and explain their predictions. This tool is the first of its kind, providing a comprehensive way to visualize and understand the predictions made by GNN models. As e-commerce companies, research institutions deploy multiple GNN models, our tool has the potential to be widely used and shared, offering benefits for a variety of people.

Our tool is advantageous for several reasons. For one, it can improve the performance of GNN models by identifying ambiguous or mislabeled nodes. By labeling these nodes correctly, the model can make better predictions through active learning, improving its training overall. Furthermore, our tool can detect potential biases in the model, such as over-reliance on certain edge information or vulnerability to simple connections. However, there are still some other features can be further added to the tool. It would be more general to include options for explaining various graph models. We can add explanation methods for graph attention models, label propagation, and temporal graph models (on recent paper has developed a temporal graph explainer [10]), etc. As graph models are widely adopted across multiple teams, adding this feature enables more people benefit from it. Another critical feature is to make the UI more user friendly and interactive, such as adding highlight options etc. As discussed in the method section, the pipeline's scalability is limited when it comes to handling vast graph datasets. However, this can be easily fixed once **DGL** is available on CodeBrowser. Meanwhile, to address this challenge, one can leverage Neptune database, which enables almost real-time inference. Combining this tool with Neptune database can enhance the processing speed and improve the overall performance of the pipeline.

To summary, the importance of explainable artificial intelligence (XAI) has become increasingly apparent as real-world problems grow more complex. Our tool offers a promising solution to this problem, providing researchers a way to identify potential issues and improve the accuracy and effectiveness of GNN models. In addition, our tool has the potential to promote the wider adoption of GNNs in practical applications and make data more accessible to non-technical individuals.

References

[1] T. N. Kipf and M. Welling. *Semi-supervised classification with graph convolutional networks*. 2020.

[2] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. *Consistent Individualized Feature Attribution for Tree Ensembles*. 2019. arXiv: 1802.03888 [cs.LG].

[3] Dongsheng Luo et al. "Parameterized Explainer for Graph Neural Network". In: (2020). arXiv: 2011.04573 [cs.LG].

[4] Christoph Molnar. "A unified approach to interpreting model predictions". In: *Advances in Neural Information Processing Systems* 30 (2019), pp. 4765–4774.
[5] *Neo4j*. Graph database software. URL: https://neo4j.com/.
[6] Lloyd S. Shapley. "A Value for n-person Games". In: *Contributions to the Theory of Games* II (1953). Ed. by Harold W. Kuhn and Albert W. Tucker, pp. 307–317.
[7] *TigerGraph*. Distributed graph database system. URL: https://www.tigergraph.com/.
[8] M. Vu and M. T. Thai. *Pgm-explainer: Probabilistic graphical model explanations for graph neural networks*. 2020.
[9] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2020), pp. 4–24.
[10] Wenwen Xia et al. "Explaining Temporal Graph Models through an Explorer-Navigator Framework". In: *The Eleventh International Conference on Learning Representations*. 2023.
[11] Z. Ying et al. *Gnnexplainer: Generating explanations for graph neural networks*. 2019.
[12] H. Yuan et al. *Explainability in Graph Neural Networks: A Taxonomic Survey*. 2022.
[13] H. Yuan et al. *XGNN: Towards model-level explanations of graph neural networks*. 2020.
[14] Hao Yuan et al. *On Explainability of Graph Neural Networks via Subgraph Explorations*. 2021. arXiv: 2102.05152 [cs.LG].
[15] Meng Zhang, Yunfei Chen, and Ning Guan. "Graph neural networks in biology: a survey". In: *Briefings in Bioinformatics* 22.4 (2021), pp. 1938–1955.

6 Appendix

Figures 3 and 4 shows additional information that will be displayed on the UI.

relation	importance
FINGERPRINT	0.145958
FINGERPRINT	0.142003
FINGERPRINT	0.141650
FINGERPRINT	0.138906
FINGERPRINT	0.136782
IP	0.160384
IP	0.134317

a

Connection Summary

	relation	counts	connections	blocked
0	BANK_ACCOUNT	0	0	0
1	CIS_SIGN_IN	0	0	0
2	CREDIT_CARD	28	1	1
3	EMAIL	0	0	0
4	FINGERPRINT	2	0	0

b

Figure 3. Extra information that will be displayed on the UI. Table a shows the most important 1 hop connections, where the normalized importance scores are shown. Table b shows all the 1 hop edges of the interested seller, the number of connected person (and how many of them are fraudster(blocked)) for each edge type are shown as well.

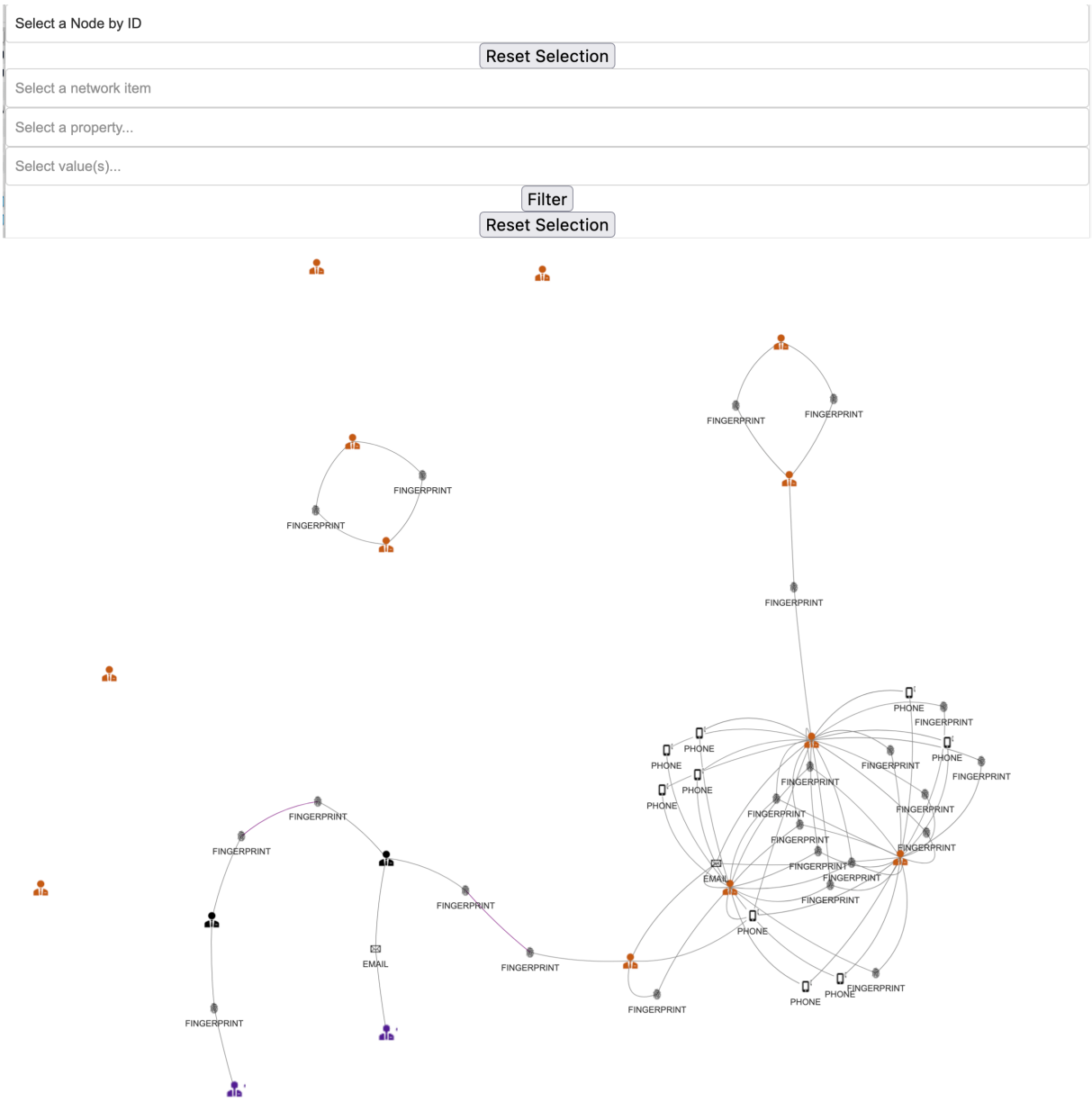


Figure 4. Cluster view of a set of interested entities. Users can select and filters using the top boxes. Users need to bring a csv file containing clusters of entities, where the format follows source entity, destination entity, and relation name.