Indian Institute of Technology Hyderabad

Computer Science and Engineering Department

CS6890 - Fraud Analytics

Assignment: Identify sets of dealers doing circular trading using Node2Vec

Spring 2023

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Sankalp Mittal CS21MTECH12010
Shikhar Jain CS22MTECH02002
Suranjan Daw CS21MTECH12008
Sanyog Sharma AI21MTECH12003
Akshad Shyam Purushottamdas AI22MTECH02006

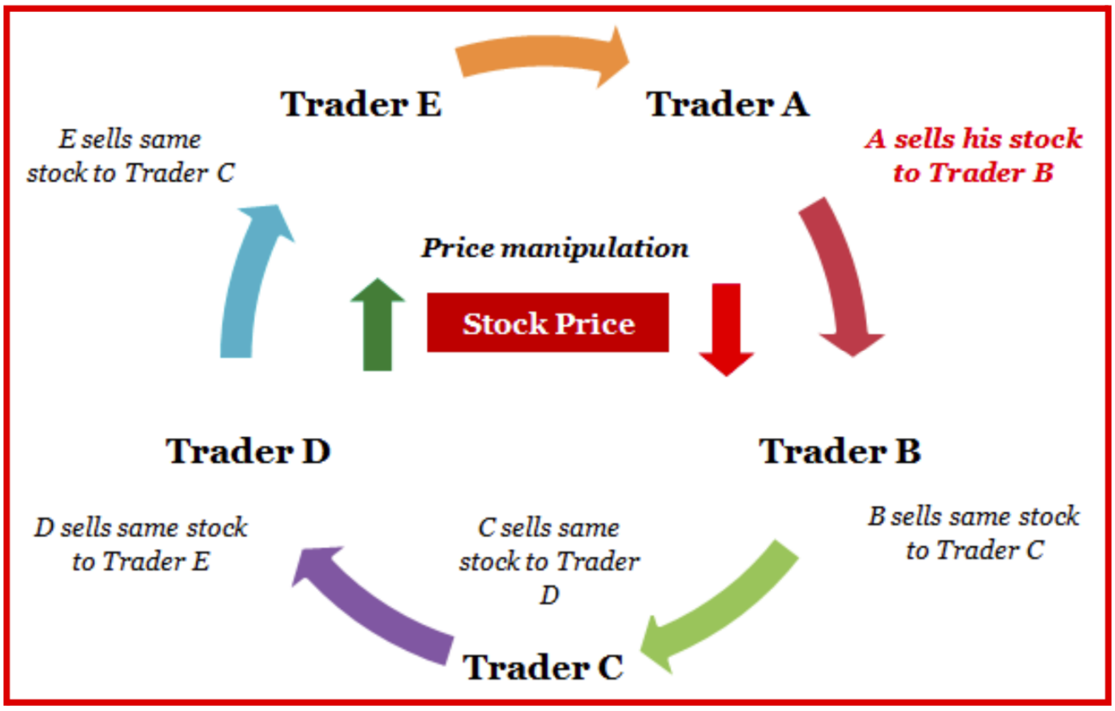

Trader E
Trader A
E sells same stock to Trader C
A sells his stock to Trader B
Price manipulation
Stock Price
Trader D
Trader B
D sells same stock to Trader E
C sells same stock to Trader D
B sells same stock to Trader C
Trader C

Figure 1: Circular Trading

Abstract

This document holds the assignment submission for implementing the Node2Vec algorithm in the context of identifying various *circular traders* for the dataset (where each data-point represents an invoice) provided as part of the problem document. In other words, our work illustrates the methodology of finding the circular traders for the provided dataset.

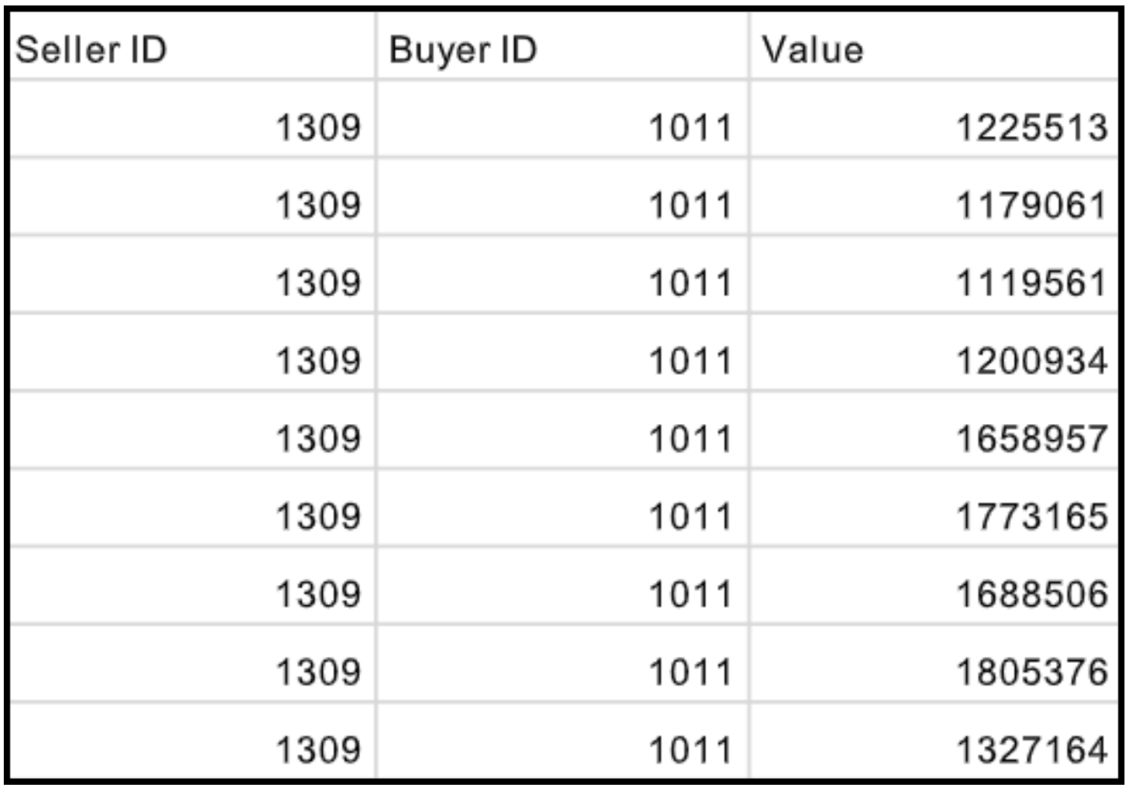
1 Problem Statement

Circular trading is a method of evading taxes under the Goods and Services Tax system. A group of deceitful traders attempts to conceal their illegal transactions by creating a series of fake transactions in a short period. These transactions have no real value or impact on traded goods or services. As the database of seller-buyer transactions is vast, it becomes difficult for data scientists to identify these circular traders and their fraudulent activities manually. To tackle this problem, this study proposes a framework that uses big data analytics and graph representation learning techniques to identify groups of circular traders and isolate their illegitimate transactions. The provided dataset `Iron_dealers_data.csv` is used to test this approach, and several communities of circular traders were successfully uncovered. See Fig. 1 to understand circular trading.

2 Description of Dataset

The dataset (see Fig. 2) contains three columns `Seller Id`: Unique ID for a seller, `Buyer Id`: Unique id of the buyer and `Value`: Amount in Rs. corresponding to the respective transaction. The dataset has 130535 rows with 703 sellers and 371 buyers, such that there are 799 unique individuals involved in the transactions captured by this dataset.

Below is the subgraph induced by a node whose ID is “290”. We can observe many parallel transactions, as shown in Fig. 3. We use Networkx Hagberg et al. 2008 library to preprocess the



Seller ID	Buyer ID	Value
1309	1011	1225513
1309	1011	1179061
1309	1011	1119561
1309	1011	1200934
1309	1011	1658957
1309	1011	1773165
1309	1011	1688506
1309	1011	1805376
1309	1011	1327164

Figure 2: Dataset Description

dataset as a graph. We use polynomial time algorithms like Johnson's algorithm to find the cycles in the graph and we uncover the following details as shown in Fig. 4.

This means that while uncovering circular traders, we need to consider all cycles of length two to five to capture the problem and identification better.

3 Algorithm

3.1 Node2Vec Algorithm

Node2vec is an algorithmic framework for learning continuous feature representations for network nodes. In node2vec, we learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. We also define a flexible notion of a node's network neighborhood and design a biased random walk procedure, which efficiently explores diverse neighborhoods. This algorithm generalizes prior work based on rigid notions of network neighborhoods, and we argue that the added flexibility in exploring neighborhoods is the key to learning richer representations. Node2vec is effective over existing state-of-the-art techniques on multi-label classification and link prediction in several real-world networks from diverse domains. For more details, refer Grover and Leskovec 2016.

3.2 DB-SCAN

DB-SCAN is a clustering algorithm that we will use to cluster the node embeddings. For more details, see Ester et al. 1996.

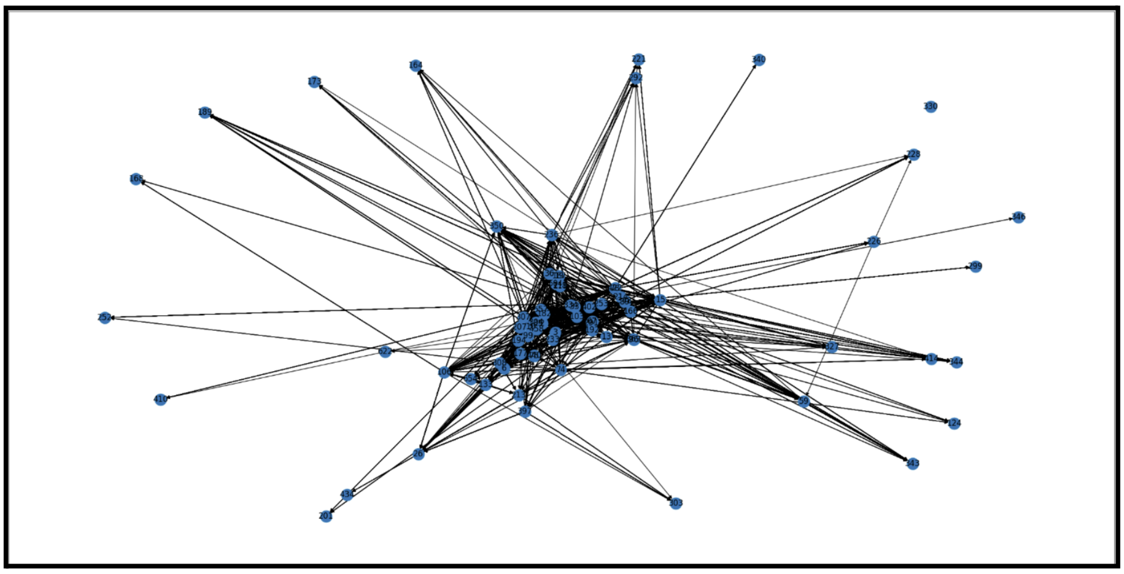


Figure 3: Presence of parallel edges in dataset

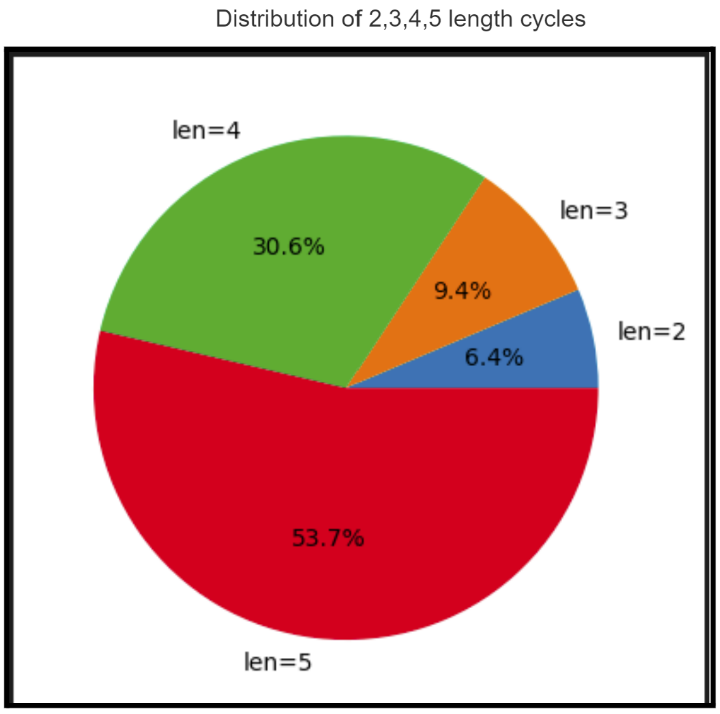


Figure 4: Cycles in input graph

3.3 Our Approach

We build our approach on top of several works, such as Mehta et al. 2022. See Table 1 for notations. We first find all the cycles listed in Table 1. We are given a directed weighted graph G. We need to convert our graph to a weighted, undirected graph G'. The work Klymko et al. 2014 gives an algorithm to convert an unweighted directed graph to a weighted undirected graph by counting the number of two-cycles (back-edges) in every three-cycle and giving weights to each edge of 3-cycle proportionately. Building upon this, we design a novel weighting scheme used to design G' from

G as follows,

$$\forall e \in \bigcup_{i=2}^{5} C_i \;\; w'(e) = \sum_{i=2}^{5} \left(\left(\alpha_{iSW} \times e\left(C_{iSW}\right) \right) + \left(\alpha_{iNSW} \times e\left(C_{iNSW}\right) \right) \right) \times max\left(\log_{10}(w(e_{max,iSW})), 1 \right) \tag{3.1}$$

Also, $w'(e_r) = max(w'(e_r),\ w'(e))$ and $w'(e) = max(w'(e_r),\ w'(e))$, where e_r denote reverse edge of e. Note that from the above equation, we are biasing G' only to contain edges from cycles. In this way, graph will not be very sparse. We also design our approach such that $\alpha_{kSW} >> \alpha_{kNSW}$. We detail our approach in Alg. 1.

Notation	Meaning
G	Input directed weighted graph
G'	Processed Undirected weighted graph
C_k	k-cycle
C_{kSW}	k-cycle having same edge weights (under tolerance)
C_{kNSW}	k-cycle having different edge weights
$e_{max,kSW}$	Maximum weight edge of C_{kSW}
α_{kSW}	Multiplicative factor of C_{kSW}
α_{kNSW}	Multiplicative factor of C_{kNSW}
$w(e)$	Weight (transaction value) of edge $e \in G$
$w'(e)$	Weight (community factor) of edge $e \in G'$
$e(C)$	Count of cycle C containing edge e

Table 1: Symbols frequently used in this paper.

Algorithm 1 Identify circular trading

```
1: Input: Directed, weighted multi-graph $G$
2: Output: Communities harboring circular traders
3: Convert $G$ to a directed weighted simple graph by summing all parallel edges in $G$
4: Find all cycles $\bigcup_{i=2}^{5} C_i$ in $G$ in $O(V^2 log(V) + VE)$
5: Construct undirected weighted graph $G'$ using Eq. 3.1
6: Apply Node2Vec algorithm on $G'$ to get node embeddings $EM_{G'}$
7: Cluster node embeddings $EM_{G'}$ using DB-SCAN to get cluster assignments
8: To find, min_pts and eps of DB-SCAN, we tried using the elbow method. However, we
   realized the presence of many elbows (as shown in Fig. 5), making this method unsuitable. So
   we instead manually fine-tuned these hyperparameters and converged as shown in Table 2
9: Visualize the clustered embeddings $EM_{G'}$ using Principal Component Analysis (PCA) – done
   directly on $EM_{G'}$ – or T-SNE in 2-D space
10: Verify that for each cluster, we get tightly knit communities in $G$ containing cycles, especially
    the ones having the same weight edges under some tolerance given by $\frac{std(w(e),\, e \in C)}{mean(w(e),\, e \in C)} < tol$
```

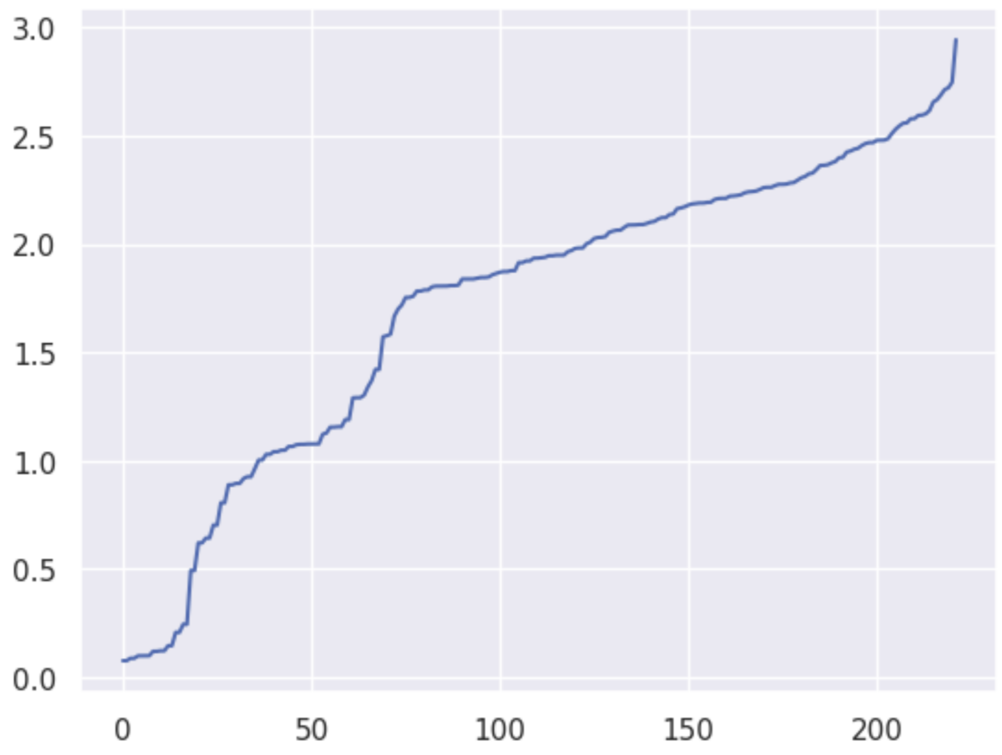
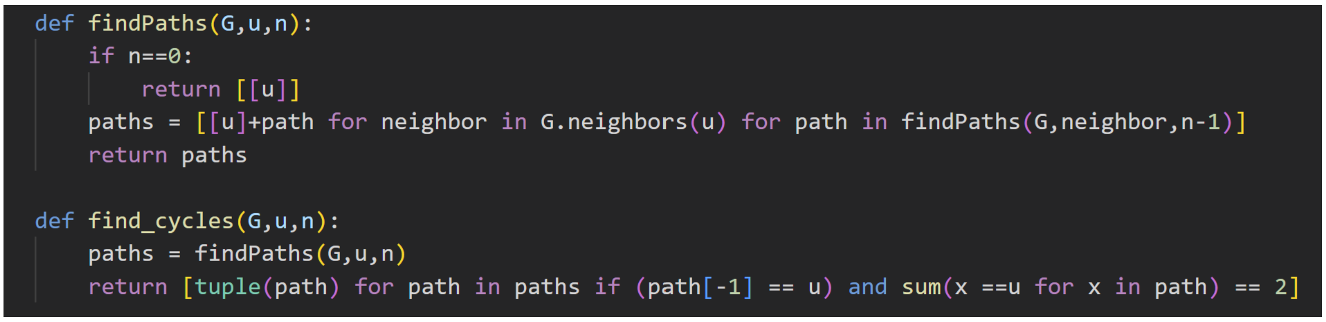


Figure 5: Elbow Method, we obtained upon following sir's advice and Alg. for calculating eps and min_pts. This plot is one of the outcomes of the Alg. which uses nearest neighbors.

3.4 Code Walkthrough

We give the code walkthrough of Alg. 1 as follows. The code to find cycles is shown in Fig. 6. Then the code for counting 3-cycles $\forall e \in G$ is shown in Fig. 7. Finally, we show the implementation of Eq. 3.1 in Fig. 8. Note that *after taking 5% tolerance among weights of edges of any cycle, only these above edges are coming in a cycle having "similar" weights on its edges (odd indices in Fig. 9), and all these edges are coming in a two-cycle only.* This is illustrated in Fig. 9.



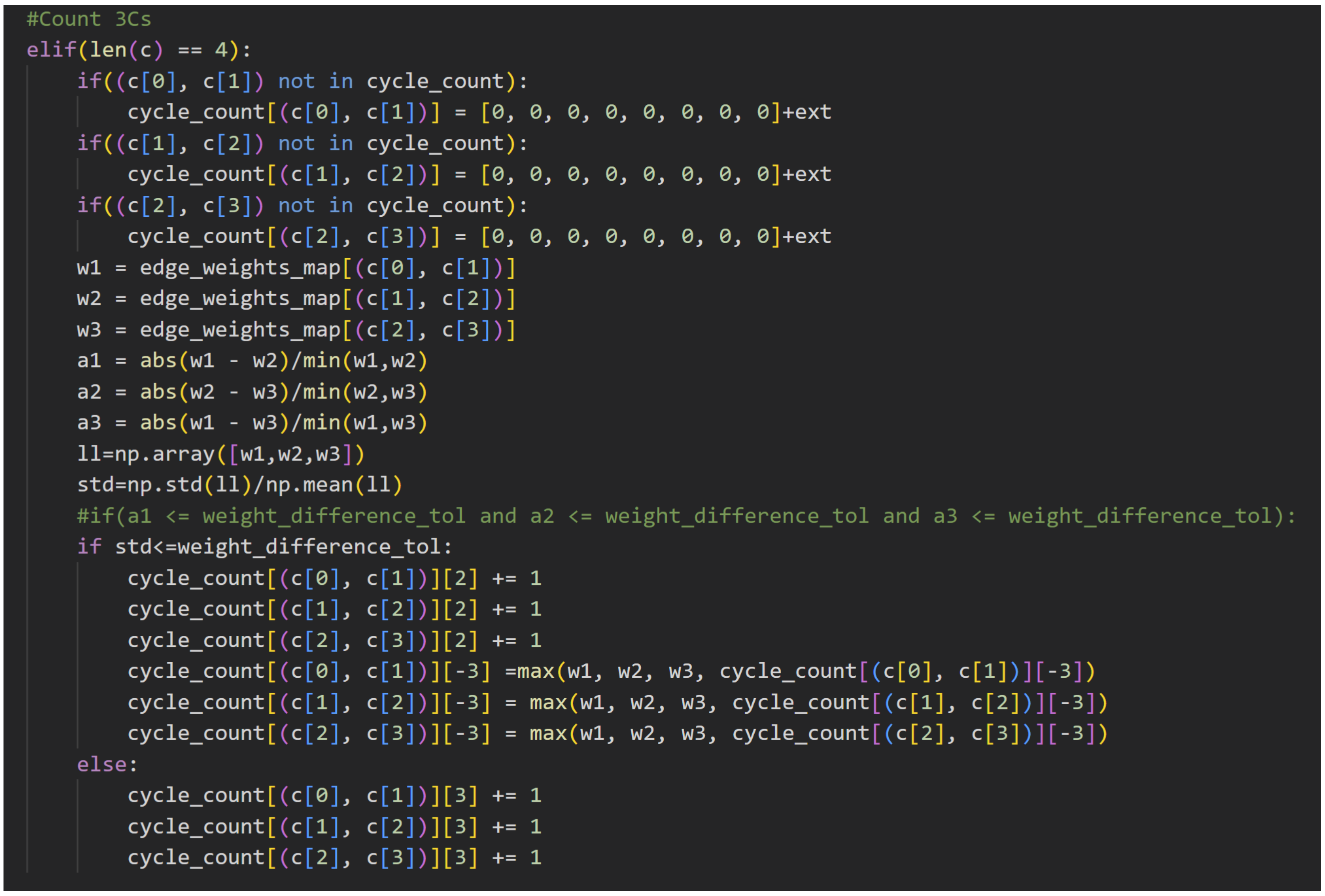
```
def findPaths(G,u,n):
    if n==0:
        return [[u]]
    paths = [[u]+path for neighbor in G.neighbors(u) for path in findPaths(G,neighbor,n-1)]
    return paths

def find_cycles(G,u,n):
    paths = findPaths(G,u,n)
    return [tuple(path) for path in paths if (path[-1] == u) and sum(x ==u for x in path) == 2]
```

Figure 6: Cycle Detection

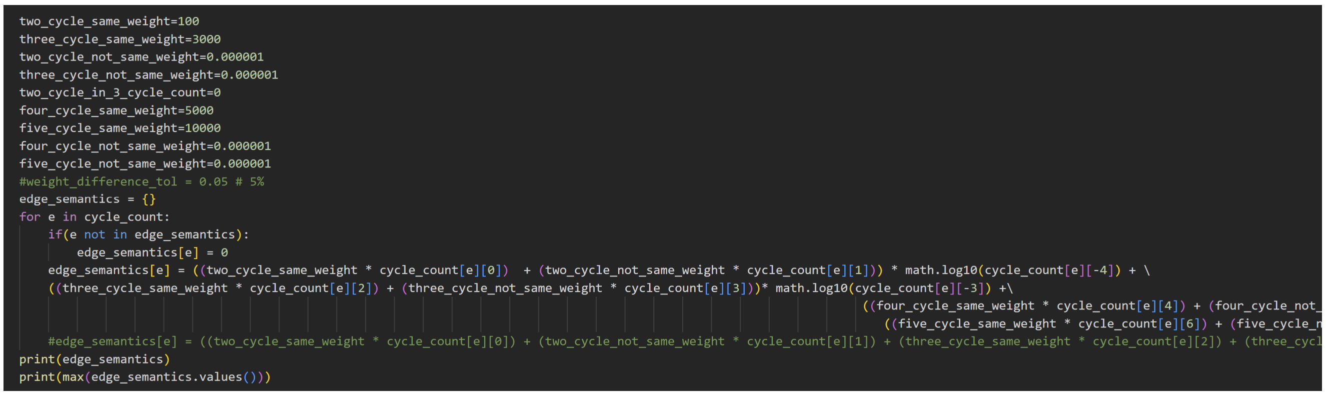
4 Results

We show the results for the following hyperparameters as shown in Table 2. Further, for each cluster in Fig. 12, we trace those nodes back to G to visualize the communities. For instance, Fig.



```
#Count 3Cs
elif(len(c) == 4):
    if((c[0], c[1]) not in cycle_count):
        cycle_count[(c[0], c[1])] = [0, 0, 0, 0, 0, 0, 0, 0]+ext
    if((c[1], c[2]) not in cycle_count):
        cycle_count[(c[1], c[2])] = [0, 0, 0, 0, 0, 0, 0, 0]+ext
    if((c[2], c[3]) not in cycle_count):
        cycle_count[(c[2], c[3])] = [0, 0, 0, 0, 0, 0, 0, 0]+ext
    w1 = edge_weights_map[(c[0], c[1])]
    w2 = edge_weights_map[(c[1], c[2])]
    w3 = edge_weights_map[(c[2], c[3])]
    a1 = abs(w1 - w2)/min(w1,w2)
    a2 = abs(w2 - w3)/min(w2,w3)
    a3 = abs(w1 - w3)/min(w1,w3)
    ll=np.array([w1,w2,w3])
    std=np.std(ll)/np.mean(ll)
    #if(a1 <= weight_difference_tol and a2 <= weight_difference_tol and a3 <= weight_difference_tol):
    if std<=weight_difference_tol:
        cycle_count[(c[0], c[1])][2] += 1
        cycle_count[(c[1], c[2])][2] += 1
        cycle_count[(c[2], c[3])][2] += 1
        cycle_count[(c[0], c[1])][-3] =max(w1, w2, w3, cycle_count[(c[0], c[1])][-3])
        cycle_count[(c[1], c[2])][-3] = max(w1, w2, w3, cycle_count[(c[1], c[2])][-3])
        cycle_count[(c[2], c[3])][-3] = max(w1, w2, w3, cycle_count[(c[2], c[3])][-3])
    else:
        cycle_count[(c[0], c[1])][3] += 1
        cycle_count[(c[1], c[2])][3] += 1
        cycle_count[(c[2], c[3])][3] += 1
```

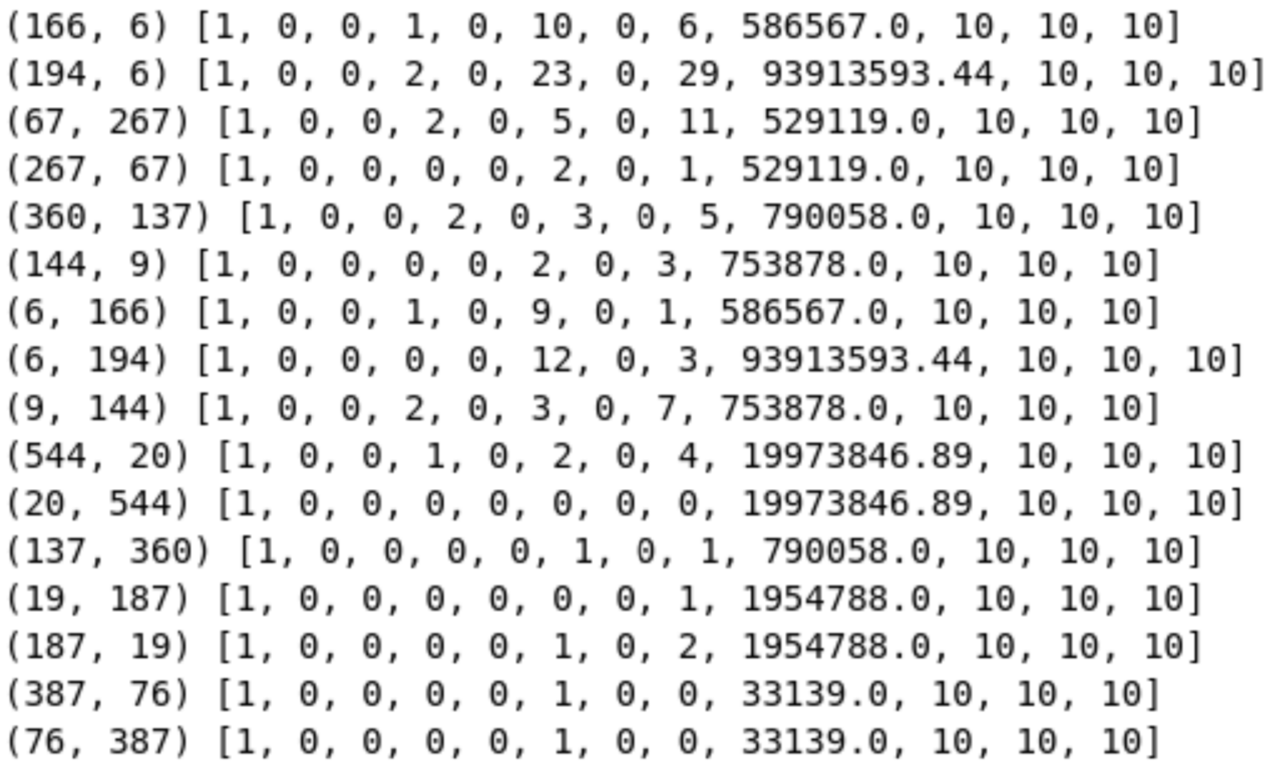
Figure 7: Cycle Count



```
two_cycle_same_weight=100
three_cycle_same_weight=3000
two_cycle_not_same_weight=0.000001
three_cycle_not_same_weight=0.000001
two_cycle_in_3_cycle_count=0
four_cycle_same_weight=5000
five_cycle_same_weight=10000
four_cycle_not_same_weight=0.000001
five_cycle_not_same_weight=0.000001
#weight_difference_tol = 0.05 # 5%
edge_semantics = {}
for e in cycle_count:
    if(e not in edge_semantics):
        edge_semantics[e] = 0
    edge_semantics[e] = ((two_cycle_same_weight * cycle_count[e][0])  + (two_cycle_not_same_weight * cycle_count[e][1])) * math.log10(cycle_count[e][-4]) + \
    ((three_cycle_same_weight * cycle_count[e][2]) + (three_cycle_not_same_weight * cycle_count[e][3]))* math.log10(cycle_count[e][-3]) +\
                                                                                                                     ((four_cycle_same_weight * cycle_count[e][4]) + (four_cycle_not_
                                                                                                                        ((five_cycle_same_weight * cycle_count[e][6]) + (five_cycle_n
    #edge_semantics[e] = ((two_cycle_same_weight * cycle_count[e][0]) + (two_cycle_not_same_weight * cycle_count[e][1]) + (three_cycle_same_weight * cycle_count[e][2]) + (three_cycl
print(edge_semantics)
print(max(edge_semantics.values()))
```

Figure 8: Eq. 3.1 Weight mapping function implementation

13 denotes a community containing a single 2-cycle. Interestingly, these similar communities have almost the same edge weights in G as shown in Fig. 14. Similarly, a community having 2-cycles and a 3-cycle is shown in Fig. 20. A community containing 2-cycle, 3-cycle, and 4-cycle is shown in Fig. 15 and Fig. 19. A tightly knit community containing all the cycles is shown in Fig. 16. Finally, observe the community shown in Fig. 17 and notice how edge weights have pair-wise similarity as shown in Fig. 18.



```
(166, 6) [1, 0, 0, 1, 0, 10, 0, 6, 586567.0, 10, 10, 10]
(194, 6) [1, 0, 0, 2, 0, 23, 0, 29, 93913593.44, 10, 10, 10]
(67, 267) [1, 0, 0, 2, 0, 5, 0, 11, 529119.0, 10, 10, 10]
(267, 67) [1, 0, 0, 0, 0, 2, 0, 1, 529119.0, 10, 10, 10]
(360, 137) [1, 0, 0, 2, 0, 3, 0, 5, 790058.0, 10, 10, 10]
(144, 9) [1, 0, 0, 0, 0, 2, 0, 3, 753878.0, 10, 10, 10]
(6, 166) [1, 0, 0, 1, 0, 9, 0, 1, 586567.0, 10, 10, 10]
(6, 194) [1, 0, 0, 0, 0, 12, 0, 3, 93913593.44, 10, 10, 10]
(9, 144) [1, 0, 0, 2, 0, 3, 0, 7, 753878.0, 10, 10, 10]
(544, 20) [1, 0, 0, 1, 0, 2, 0, 4, 19973846.89, 10, 10, 10]
(20, 544) [1, 0, 0, 0, 0, 0, 0, 0, 19973846.89, 10, 10, 10]
(137, 360) [1, 0, 0, 0, 0, 1, 0, 1, 790058.0, 10, 10, 10]
(19, 187) [1, 0, 0, 0, 0, 0, 0, 1, 1954788.0, 10, 10, 10]
(187, 19) [1, 0, 0, 0, 0, 1, 0, 2, 1954788.0, 10, 10, 10]
(387, 76) [1, 0, 0, 0, 0, 1, 0, 0, 33139.0, 10, 10, 10]
(76, 387) [1, 0, 0, 0, 0, 1, 0, 0, 33139.0, 10, 10, 10]
```

Figure 9: Actual cycles present in dataset for 5% tolerance

Node2Vec							DB-SCAN		Visualizations		
Embedding Dimension	Walk Length	Number of Walks	p	q	Windows	Min Count	Eps	Min Points	DBSCAN	T-SNE	PCA
130	400	100	0.7	1.4	5	100	1.6	2	Fig. 10	Fig. 11	Fig. 12

Table 2: Hyperparameter Tuning & Experimental Results

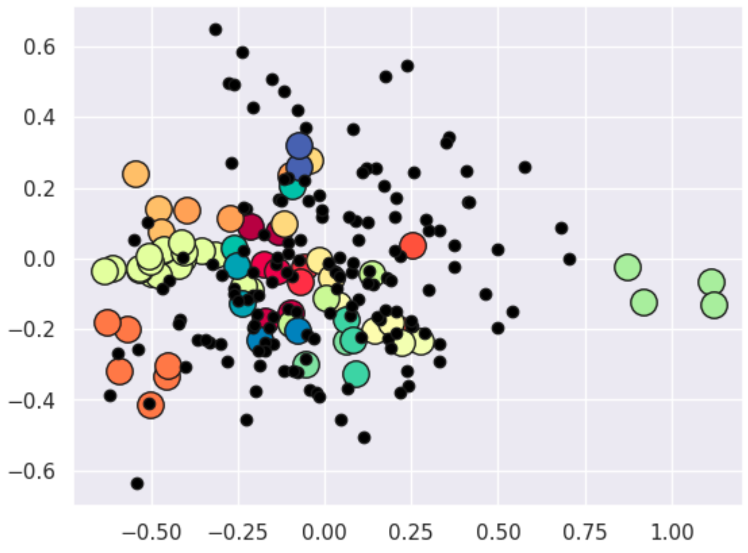


Figure 10: DB-SCAN Visualization - Back dots are noise points

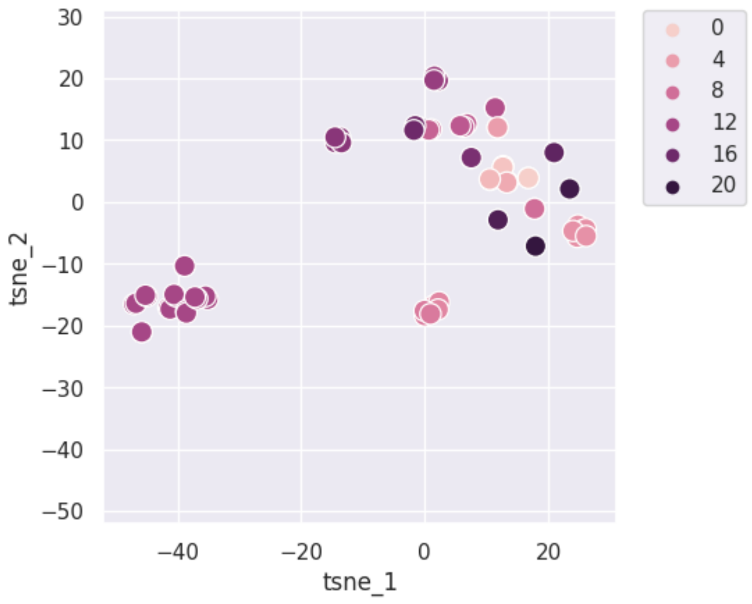


Figure 11: T-SNE Visualization over DBSCAN generated Labels

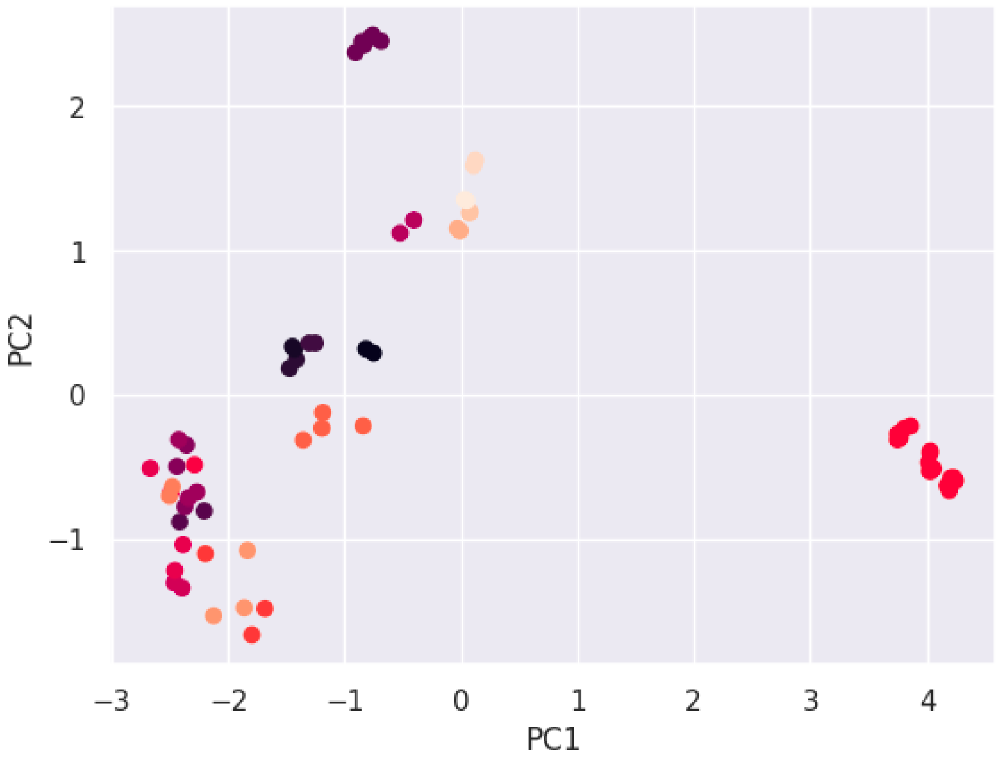


Figure 12: PCA Visualization of Node Embeddings. Commnunity Clusters can be observed.

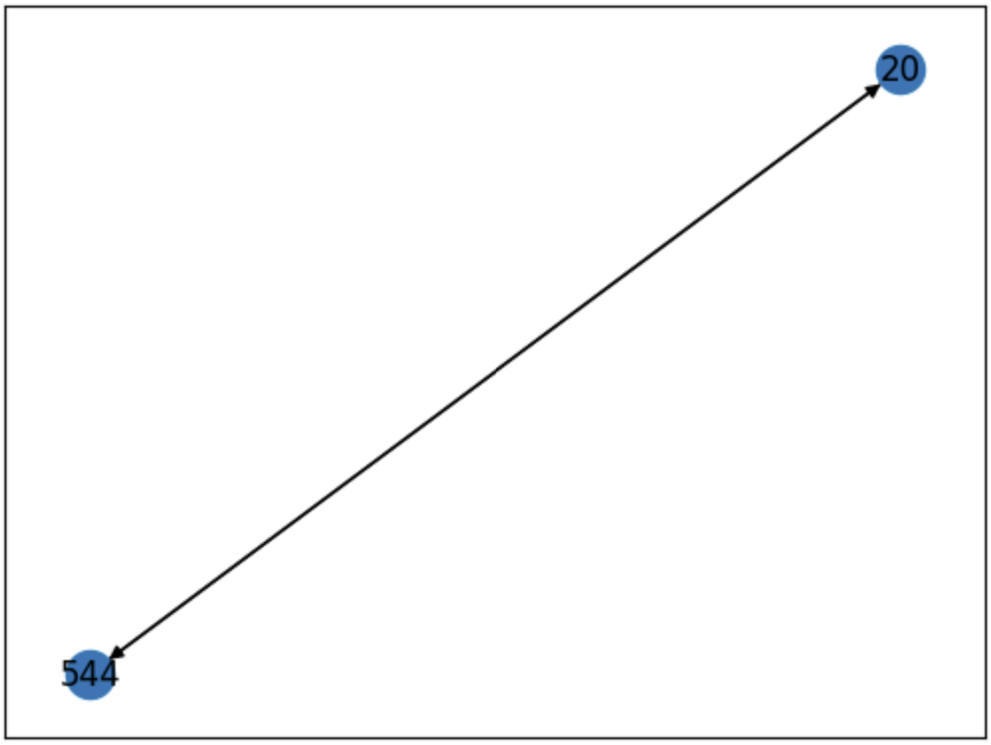
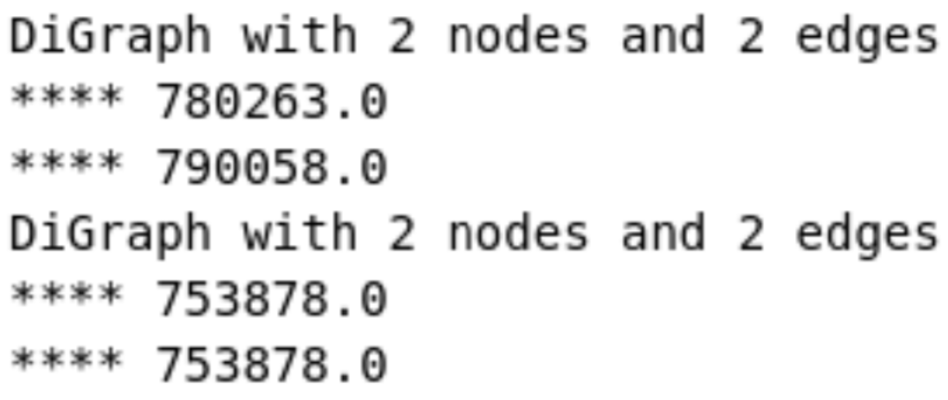


Figure 13: 2-Cycle Community. Community detected by Node2vec Embd.



```
DiGraph with 2 nodes and 2 edges
**** 780263.0
**** 790058.0
DiGraph with 2 nodes and 2 edges
**** 753878.0
**** 753878.0
```

Figure 14: Weights of 2-cycle communities. Community detected by Node2vec Embd.

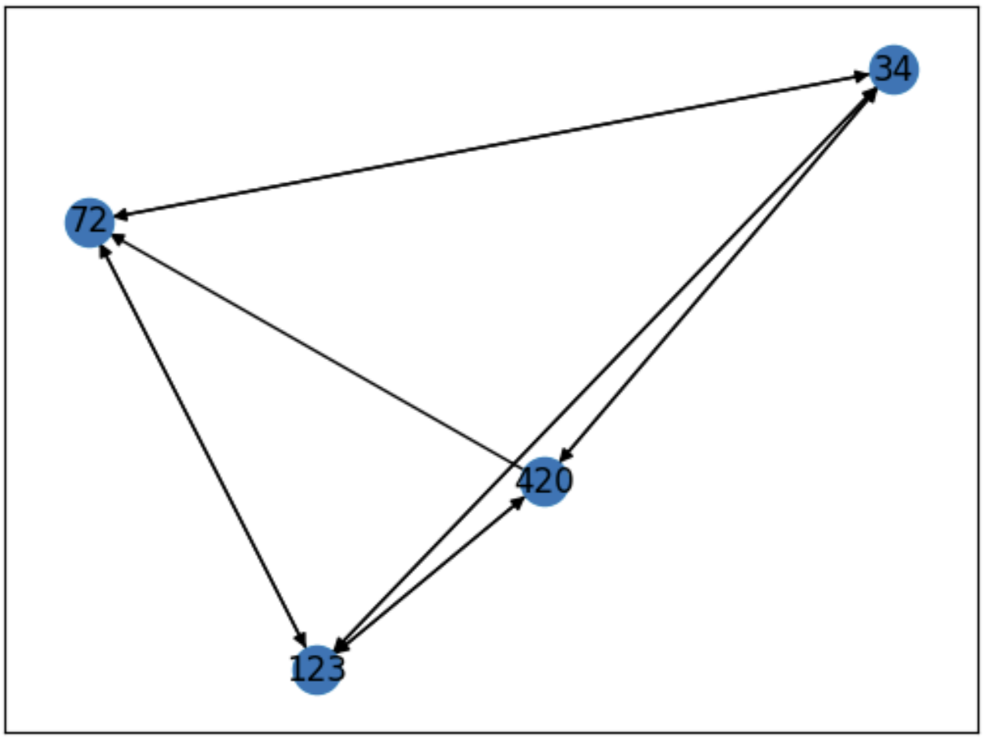


Figure 15: 2-Cycle, 3-cycle and 4-cycle Community. Community detected by Node2vec Embd.

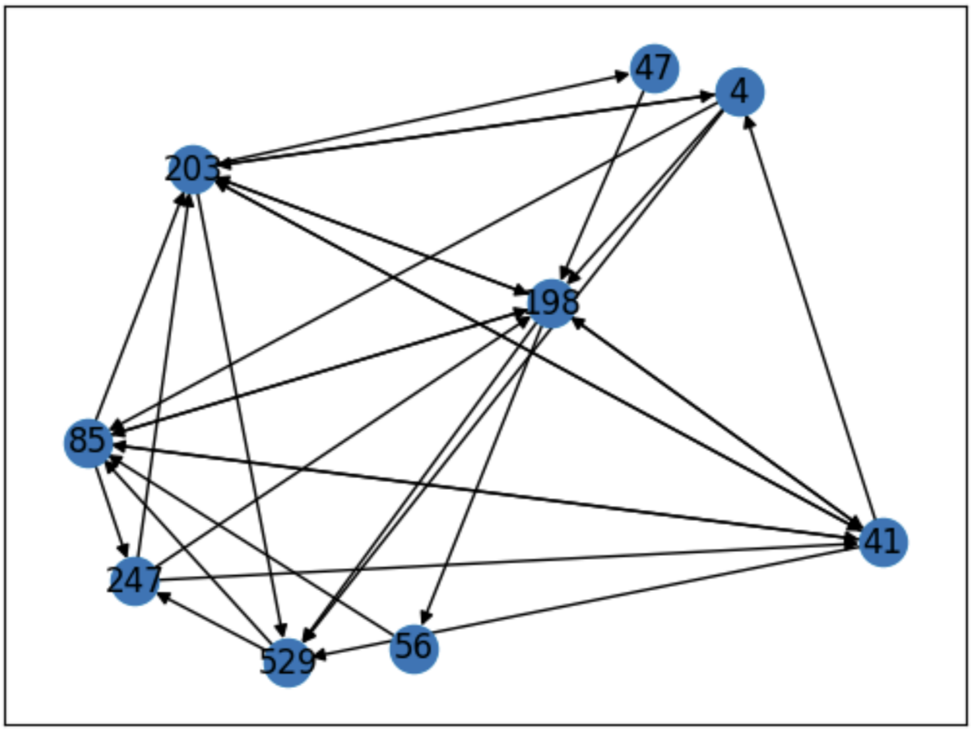


Figure 16: 2-Cycle, 3-cycle, 4-cycle and 5-cycle Community. Community detected by Node2vec Embd.

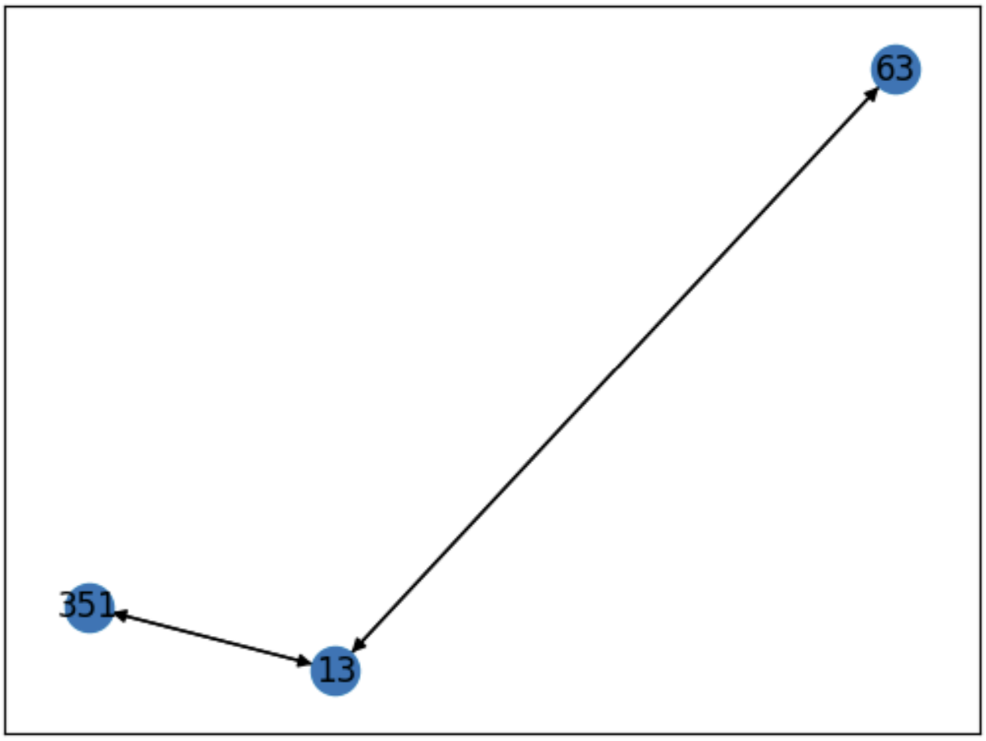
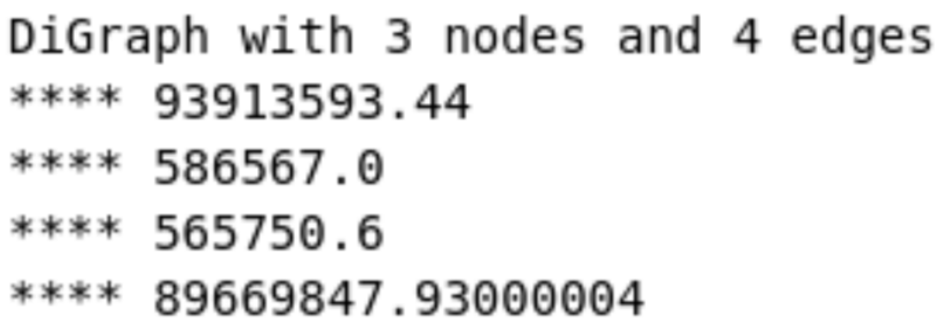


Figure 17: 2-Cycles Community. Community detected by Node2vec Embd.



```
DiGraph with 3 nodes and 4 edges
**** 93913593.44
**** 586567.0
**** 565750.6
**** 89669847.93000004
```

Figure 18: 2-Cycles Community having pair-wise weight similarity. Community detected by Node2vec Embd.

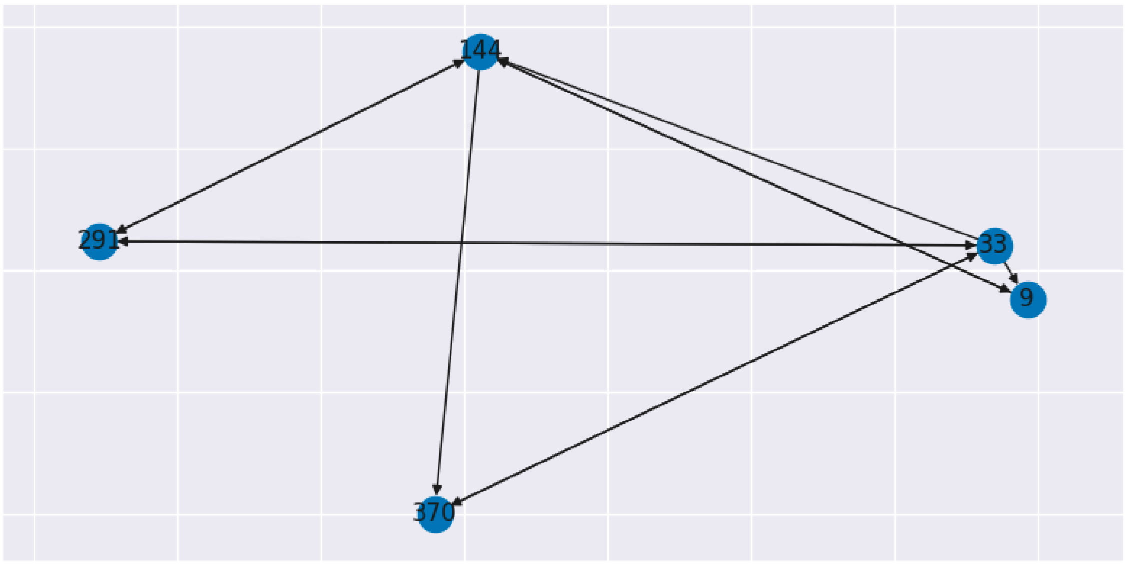


Figure 19: 2-Cycle, 3-Cycle and 4-Cycle Community. Community detected by Node2vec Embd.

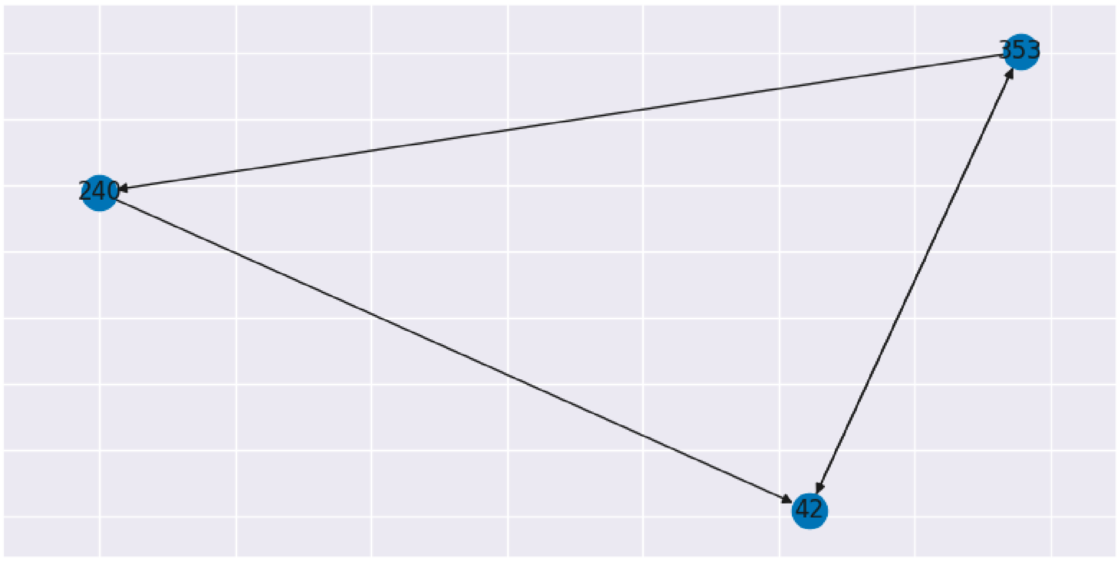


Figure 20: 2-Cycles, 3-Cycle Community. Community detected by Node2vec Embd.

References

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu (1996). "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, pp. 226–231.

Grover, Aditya and Jure Leskovec (2016). "node2vec: Scalable Feature Learning for Networks." In: *CoRR* abs/1607.00653. arXiv: 1607.00653. URL: http://arxiv.org/abs/1607.00653.

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

Klymko, Christine, David Gleich, and Tamara G. Kolda (2014). *Using Triangles to Improve Community Detection in Directed Networks*. arXiv: 1404.5874 [cs.SI].

Mehta, Priya, Sanat Bhargava, M. Ravi Kumar, K. Sandeep Kumar, and Ch. Sobhan Babu (2022). *Representation Learning on Graphs to Identifying Circular Trading in Goods and Services Tax*. arXiv: 2208.07660 [cs.LG].