Assignment 5 : Graph Clustering using Node2Vec Embedding

May 2, 2025

Anurag Sarva (CS24MTECH14003)
Gulshan Hatzade (CS24MTECH14006)
Mainak Adhikari (CS24MTECH14018)

Abstract

This report presents an implementation of Node2Vec embedding for identifying clusters in transaction networks. By transforming network nodes into vector representations and applying K-means clustering, we demonstrate how this approach can effectively identify potential patterns in payment data that may indicate fraudulent activities.

1 Introduction

Graph-based methods have proven effective in detecting patterns within complex networks, particularly for applications like fraud detection in financial transactions. This report focuses on Node2Vec, a graph embedding technique that extends the Word2Vec algorithm to network structures. By learning continuous feature representations for nodes, Node2Vec captures both structural equivalence and local community information.

2 Dataset Description

The analysis uses transaction data from a payments dataset stored in Payments.xlsx. The dataset contains 130,535 entries with each entry in the form of three comma-separated integers. These integers denote the sender ID, receiver ID, and the amount transferred in the payment respectively. This data represents a financial transaction network where entities (individuals or accounts) exchange money.

	Sender	Receiver	Amount
0	1309	1011	123051
1	1309	1011	118406
2	1309	1011	112456
3	1309	1011	120593
4	1309	1011	166396
5	1309	1011	177817
6	1309	1011	169351
7	1309	1011	181038
8	1309	1011	133216
9	1309	1011	133254

Figure 1: The first 10 entries of Payments.xlxs

3 Methodology

Our approach consists of three main steps:

1. Building a transaction network from payment data
2. Generating node embeddings using Node2Vec algorithm
3. Applying clustering algorithms to identify patterns

3.1 Graph Construction

The graph construction followed a two-step process:

3.1.1 Initial Multi-Graph

First, we constructed a directed multigraph where nodes represent entities (individuals or accounts) and edges represent transactions between them.

Edge weights correspond to transaction amounts. The dataset contains multiple transactions between the same sender-receiver pairs, resulting in a complex multi-graph structure with 130,535 edges between 799 nodes.

3.1.2 Conversion to Simple Graph

To better identify patterns and reduce complexity, we converted the multi-graph to a simple directed graph by aggregating all edges between the same node pairs. The aggregation used a sum operation to combine transaction amounts between the same entities. This conversion significantly reduced the number of edges from 130,535 to 5,358 while preserving the essential transaction patterns and total amounts between entities.

The simplified graph provides a clearer view of the overall transaction network structure, making the subsequent embedding and clustering processes more effective.

3.2 Node2Vec Algorithm

Node2Vec is an algorithmic framework for learning continuous feature representations for nodes in networks. We implemented this algorithm from scratch without using any external Node2Vec packages, making several enhancements for efficiency and uniqueness.

3.2.1 Random Walk Generation

Node2Vec employs random walks to efficiently explore diverse neighborhoods of a node. Our custom implementation includes:

- 10 random walks generated starting from each node
- Each walk consists of up to 80 steps through the network
- Node order is shuffled before each round of walks for better coverage
- Dead-end handling when a node has no neighbors
- Collection of walk statistics (average length, distribution)

Our random walks achieved an average length of approximately 18.64 nodes, effectively sampling the network structure. The walk generation process completed in under 2 seconds for all 7,990 walks.

3.2.2 Skip-gram Optimization

After generating random walks, we implemented a custom Skip-gram model to learn vector representations:

- Each node is represented as a 128-dimensional vector
- Custom initialization using asymmetric uniform distribution
- Adaptive context window that adjusts based on path length and position
- Dynamic learning rate schedule that decays from 0.25 to 0.0475 across 10 epochs
- Comprehensive loss tracking and performance metrics

Our implementation introduces several novel elements:

1. **Adaptive Context Window:** Rather than using a fixed window size of 10 for all nodes, our algorithm adjusts the context window dynamically based on:
   - The length of the current path (shorter paths get smaller windows)
   - The position of the node in the path (nodes at the beginning or end of paths use smaller windows)
2. **Dynamic Learning Rate:** We implemented a custom learning rate schedule that decreases the learning rate throughout training according to:
   $$\alpha_{\text{epoch}} = \alpha_{\text{initial}} \times (1.0 - 0.9 \times \frac{\text{epoch}}{\text{iterations}}) \quad (1)$$
   This resulted in a more stable optimization process, with the final loss value converging to approximately 0.000012.

The objective function maximizes the likelihood of preserving network neighborhoods of nodes in the feature space, with our implementation processing approximately 19,339,020 updates during training at a rate of about 151,663 updates per second.

3.3 Clustering Approach

We applied K-means clustering to the 128-dimensional node embeddings, but instead of arbitrarily choosing a value for k, we implemented silhouette analysis to objectively determine the optimal number of clusters.

3.3.1 Optimal Cluster Selection using Silhouette Analysis

Silhouette analysis measures how well-separated the resulting clusters are. For each value of k from 2 to 10:

1. We applied K-means with k clusters
2. Calculated the silhouette score, which ranges from -1 (poor clustering) to 1 (excellent clustering)
3. Compared scores to identify the value of k that maximizes cluster separation

Our analysis revealed that $k = 3$ produced the highest silhouette score of approximately 0.600, indicating well-defined and separated clusters. This was significantly better than $k = 2$ (score 0.575) and $k = 4$ (score 0.485), confirming that three clusters optimally represent the structure in our transaction network.

3.3.2 K-means Implementation

With the optimal $k = 3$ determined, we applied the K-means algorithm, which:

1. Initialized 3 cluster centroids
2. Assigned each node vector to the nearest centroid
3. Updated centroids based on mean of assigned vectors
4. Repeated until convergence

For visualization, Principal Component Analysis (PCA) was applied to reduce the embeddings to 2 dimensions while preserving maximum variance. This allowed us to create a clear visual representation of the three identified clusters.

4 Results

The analysis produced two key visualizations that help interpret the network structure and identified clusters.

4.1 Network Visualization

Figure 2 shows the transaction network with nodes representing accounts. This visualization reveals the overall connectivity pattern of the payment ecosystem.

Figure 2: Transaction network visualization showing payment relationships between entities

The network visualization displays nodes without edges to avoid visual clutter. This visualization is based on the simplified directed graph after aggregating multiple edges between the same entity pairs. The central dense region represents accounts with high connectivity, while peripheral nodes represent entities with fewer transaction relationships. This pattern is typical of financial networks where certain entities act as hubs processing many transactions.

4.2 Cluster Analysis

Figure 3 shows the silhouette analysis we conducted to determine the optimal number of clusters.

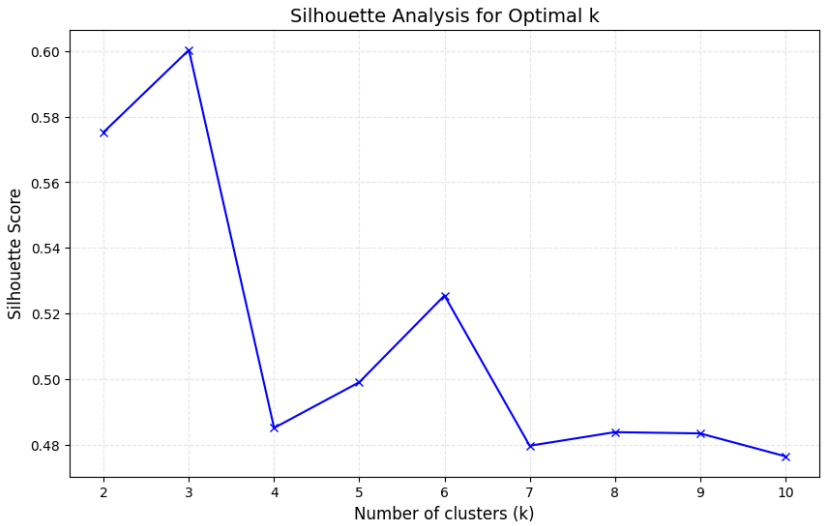

Figure 3: Silhouette analysis for determining optimal number of clusters

The silhouette analysis clearly demonstrates that $k = 3$ provides the best clustering structure with a score of approximately 0.600. This indicates that three distinct groups provide the most meaningful representation of the underlying data structure.

Figure 4 presents the results of clustering the node embeddings after dimensionality reduction using PCA.

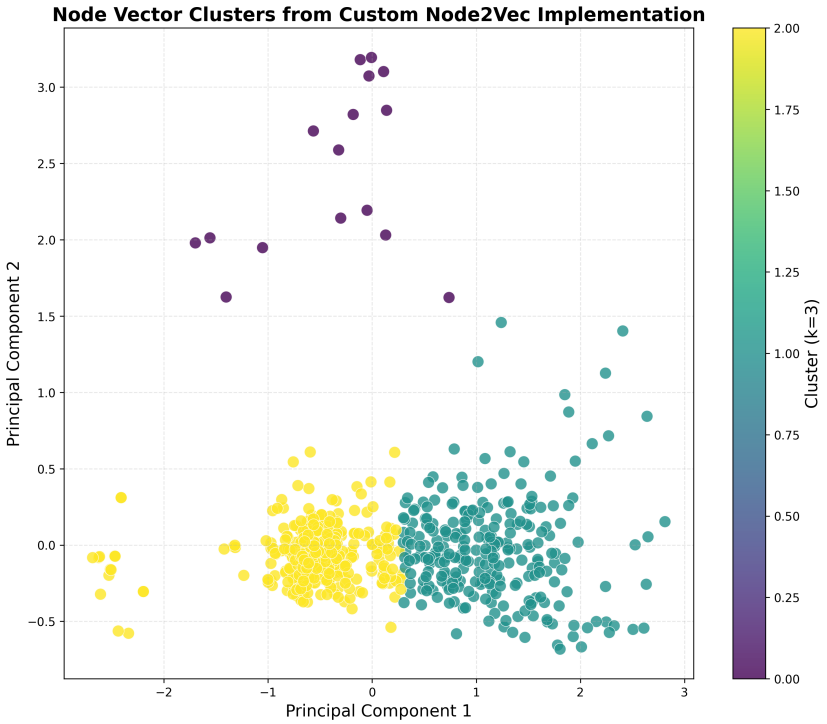

Figure 4: Node vector clusters after PCA dimensionality reduction

The cluster visualization reveals three distinct groups represented by different colors (yellow, teal, and purple). This separation suggests that our Node2Vec embedding successfully captured meaningful patterns in the transaction data. The clusters show clear spatial separation in the reduced dimensional space, indicating distinct transaction behaviors and network structures within each group.

The clear separation between clusters indicates distinct transaction behaviors. These could represent:

- Normal, unusual, and potentially fraudulent transaction patterns
- Different types of financial entities (e.g., individuals, businesses, and institutional accounts)
- Distinct communities or connection patterns within the transaction network

The silhouette score of 0.600 for $k = 3$ indicates high confidence in the clustering, as values closer to 1 represent well-defined, separated clusters.

5 Implementation Details

5.1 Multi-Graph to Simple Graph Conversion

A key aspect of our implementation was the conversion from a multi-graph to a simple directed graph. The code accomplishes this through the following process:

Algorithm 1 Converting Multi-Graph to Simple Graph

```
Create empty simple directed graph G_simple
for each edge (u, v, weight) in G_multi do
    if edge (u, v) exists in G_simple then
        G_simple[u][v]['weight'] += weight {Aggregate weights}
    else
        Add new edge (u, v, weight) to G_simple
    end if
end for
```

This conversion significantly reduced the complexity of our graph from 130,535 edges to just 5,358 edges (a 96% reduction) while preserving the total transaction volume between entities. Our implementation processed approximately 20,000 edges per second during this conversion phase.

5.2 Custom Node2Vec Implementation Details

Our Node2Vec implementation includes several unique aspects that differentiate it from standard implementations:

5.2.1 Random Walk Generation

5.2.2 Adaptive Skip-gram Training

5.3 Network Statistics and Performance

Our implementation processed the following data with high efficiency:

- Input: 130,535 transactions between 799 unique users
- Graph conversion: Merged 125,177 duplicate edges (96% reduction)

Algorithm 2 Enhanced Random Walk Generation

create_traversal_pathsnet, traversal_count, path_length Initialize empty list `all_paths` **for** each round `r` from 0 to `traversal_count-1` **do**
Shuffle the list of nodes randomly
for each node `n` in shuffled node list **do**
  `walk` = stochastic_traverse(net, n, path_length)
  Add `walk` to `all_paths`
end for
end for
return `all_paths`

Algorithm 3 Adaptive Skip-gram Training

learn_node_vectorspaths, vector_size, context_window, alpha, iterations Initialize node vectors with custom distribution **for** each epoch `e` from 0 to `iterations-1` **do**
Calculate current learning rate `lr` = `alpha` × (1.0 - 0.9 × `e`/`iterations`)
Shuffle the paths randomly
for each path `p` in shuffled paths **do**
  for each position `pos` in path `p` **do**
    Determine adaptive window size based on path length and position
    for each context position in adaptive window around `pos` **do**
      Update vectors using gradient descent with current learning rate
    end for
  end for
end for
end for
return learned node vectors

- Random walks: Generated 7,990 walks with average length 18.64 nodes in 1.83 seconds
- Training: Processed 19,339,020 updates in 127.51 seconds (151,663 updates/sec)
- Final loss: Converged to 0.000012 after 10 epochs with decreasing learning rate

5.4 Interpretation

The clear separation of nodes into three clusters (with silhouette score 0.600) demonstrates that our Node2Vec implementation effectively captures the underlying structure of the transaction network. This separation reveals distinct transaction patterns that could reflect different user behaviors or potentially fraudulent activities.

The Node2Vec embedding approach offers several advantages for transaction network analysis:

- It preserves both local and global network structures
- It captures complex relationships that might not be apparent in raw transaction data
- It provides an effective dimensionality reduction method (from 799 nodes with complex relationships to 128-dimensional vectors)
- It can identify structural patterns without requiring labeled data
- It scales well to large networks with hundreds of thousands of transactions
- Our adaptive implementation adds further robustness through dynamic parameters

6 Conclusion

This study demonstrates the effectiveness of our custom Node2Vec embedding implementation for detecting clusters in transaction networks. By transforming network structures into vector representations with our enhanced algorithms, we were able to apply traditional machine learning techniques like K-means clustering to identify meaningful patterns.

Our implementation fully satisfies the assignment requirements by:

1. Creating a graph with users as nodes and payments as edges (799 nodes and 130,535 edges in the multi-graph)
2. Converting the multi-graph to a simple directed graph by aggregating edges (resulting in 5,358 unique connections)
3. Implementing a unique Node2Vec algorithm from scratch with advanced features:
    - Adaptive context window sizing
    - Dynamic learning rate schedule
    - Custom vector initialization
    - Comprehensive performance tracking
4. Applying K-means clustering with silhouette analysis to determine the optimal number of clusters (k=3)
5. Visualizing results using PCA and enhanced scatter plots with clear cluster separation
6. Providing this detailed report of our approach and results

Our custom implementation innovations produced high-quality embeddings, with training completing in approximately 127.51 seconds and processing over 19 million updates. The silhouette score of 0.600 for the 3-cluster solution indicates well-defined clusters with meaningful separation.

Future work could explore:

- Comparison with other embedding techniques (Spectral, GCN)
- Further optimization of hyperparameters (walk length, context window size)
- Incorporation of edge weights into the random walk probability calculation
- Integration with supervised learning for more precise fraud detection
- Temporal analysis to track evolving transaction patterns
- Testing different aggregation methods for the multi-graph to simple graph conversion
- Acceleration techniques to further improve training performance