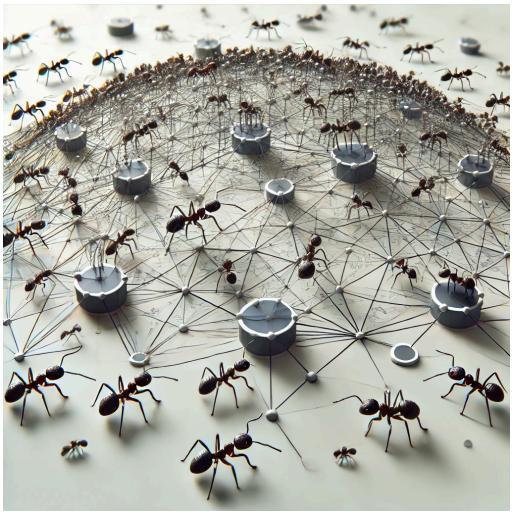
BRAUDE
College of Engineering, karmiel

Capstone Project Phase B 24-2-D-19

Citation Prediction using Ant Colony based Multi-Level Network Embedding



Yoni Azeraf - 209459239

Itamar Kraus - 318304763

Supervisor:

Dvora Toledano Kitai

GitHub

Table of Content

Abstract. This project addresses the challenge of academic papers including citations that are either weakly relevant or lack substantial significance. Such citations negatively impact the quality and relevance of research , as well as the ability of readers to navigate the vast landscape of academic information. Existing approaches, such as manual screening and expert reviews suffer from subjectivity and limited accuracy. To tackle this issue, this project introduces a novel approach based on Ant Colony based Multi-level Network Embedding (ACE), drawing insights from "ACE: Ant Colony based Multi-level Network Embedding for Hierarchical Graph Representation Learning" [11]. This project introduces a different approach. Specifically, it constructs a citation graph and applies an Ant Colony Optimization (ACO) -based algorithm [7] to identify and assess the relevance of citations. This method aims to enhance both the accuracy and objectivity of citations relevance evaluation in academic research.

Keywords: Network Embedding · Ant Colony Optimization · Coarsening · Academic Citation · Node Classification.

1. Introduction

Graph analysis is a powerful and widely used approach for representing and interpreting data structures across various domains, including social networks, biological networks, and the World Wide Web. Graph-based models enable the representation of intricate relationships between entities, facilitating the extraction of meaningful insights from large-scale datasets.

By analyzing graphs, researchers can uncover underlying structures, detect patterns, and reveal hidden relationships within intricate networks. These insights are crucial for applications such as fraud detection, recommendation systems, anomaly detection, and biological network analysis.
One fundamental task in graph analysis is clustering, which simplifies network complexity by identifying natural groupings of nodes. Effective clustering methods enable better organization, classification, and summarization of network data, making them essential for understanding large-scale structures.

Traditional network embedding techniques aim to capture node relationships by representing them in a lower-dimensional space while preserving their structural properties. However, many existing approaches primarily focus on local node neighborhoods, limiting their ability to capture hierarchical and global clustering structures within large and complex networks.

ACE [11] is an advanced algorithm that leverages ant colony principles to perform multi-level network embedding, effectively capturing the clustering properties of graphs across multiple scales. By drawing inspiration from the collective behavior of ant colonies, ACE introduces a biologically inspired approach to network representation learning.

Unlike conventional network embedding algorithms that primarily rely on local connectivity, ACE employs a novel ant colony-based coarsening algorithm. This technique transforms the graph into a hierarchy of multi-level clustering pyramids, enabling the identification of global clustering structures at different granularities. By iteratively coarsening and refining the graph, the algorithm preserves essential clustering properties while reducing computational complexity.

Furthermore, ACE utilizes an ant colony-based random walk algorithm to assess the strength of relationships between nodes, particularly within loop structures. This enhances its ability to model network connectivity and provides a more robust representation of graph topology. The combination of these techniques makes ACE a powerful tool for hierarchical graph representation learning, with potential applications in diverse fields such as citation analysis, social network modeling, and biological network research.

2. Literature Review

2.1 Naive Way

A fundamental method for representing nodes in a graph involves using the row vectors from the adjacency matrix as their feature vectors, as demonstrated in works such as [2] and [13].
An adjacency matrix is a square matrix used to represent a graph, where each row vector corresponds to a node and the matrix elements indicate the presence of edges between nodes.
While this approach is conceptually simple, it becomes impractical for real-world graphs, which can contain billions of nodes and edges. The resulting feature vectors are often high-dimensional and sparse, which leads to significant challenges in terms of computational efficiency and model performance. In particular, the sparsity of the feature vectors often results in poor performance when these representations are used as input for machine learning classifiers.

2.2 Embedding Approach

The goal of the embedding approach is to represent nodes in a graph using lower-dimensional embedding vectors, enabling the graph to be mapped into a continuous vector space. In this space, nodes with strong or similar relationships are mapped to nearby or similar embedding vectors.
These embeddings can then be processed using Machine Learning (ML) methods for various tasks such as link prediction, node classification and graph visualization, among others. By reducing the dimensionality of node representations while preserving their structural relationships, this approach facilitates more efficient and effective analysis of complex networks.

2.2.1 Embedding Algorithms Based on Matrix Factorization

Embedding algorithms based on matrix factorization are effective for processing small graphs and producing low-dimensional node vectors.
However, they face significant challenges when scaling to handle large-scale graphs due to computational complexity and the sparsity of the data.

2.2.1.1 Locally Linear Embedding

Locally Linear Embedding (LLE) method [16] focuses on preserving local relationships by reconstructing each node based on its nearest neighbors. This technique effectively captures the local structure of the graph, making it particularly useful for preserving neighborhood information in high-dimensional spaces.

2.2.1.2 Laplacian Eigenmaps

The Laplacian Eigenmaps approach [1] operates by focusing on the graph Laplacian, aiming to maintain the local connectivity of nodes. This method is highly effective for capturing the inherent geometry of the data, particularly in the context of graphs that exhibit smooth, low-dimensional manifolds.

2.2.1.3 GraRep

The GraRep technique [2] extends beyond immediate node neighborhoods by utilizing higher powers of the adjacency matrix. This allows the method to capture various scales of node relationships making it more robust in representing complex network structures.

2.2.1.4 HOPE

The HOPE procedure [13] is designed to preserve higher-order proximities by leveraging a similarity matrix. This approach allows the model to effectively capture the asymmetry in node relationships, providing a more nuanced understanding of graph structures that involve directional or weighted edges.

2.2.2 Random Walk-Based Network Embedding Algorithms

Random walk-based network embedding algorithms are particularly effective in addressing the scalability challenges encountered when working with large graphs. These methods leverage random walks to explore the graph structure and generate meaningful node embeddings.

2.2.2.1 DeepWalk

The DeepWalk technique [14] utilizes random walks on a graph to generate node sequences of nodes, which are then treated similarly to words in natural language processing. By applying techniques such as the Skip-gram model, DeepWalk creates vector representations of nodes that capture the underlying structure of the network, making it suitable for various tasks, including node classification and link prediction.

2.2.2.2 Node2Vec

Node2Vec method [8] enhances the random walk approach by introducing a flexible strategy for learning node representations. It effectively balances the exploration of local neighborhoods with the sampling of more distant regions of the graph, allowing for the discovery of richer and more diverse relationships between nodes. This flexibility makes Node2Vec highly adaptable to various types of network structures.

2.2.2.3 LINE

The LINE approach [18] is designed to efficiently handle very large networks by preserving both first order (direct neighbors) and second order (neighbors of neighbors) proximity. This dual focus allows LINE to capture the local and global structural properties of the graph, making it highly effective for large-scale network analysis and representation learning.

2.2.3 Deep Learning-Based Network Embedding Algorithms

Deep learning-based network embedding algorithms leverage advanced models with higher capacities to uncover more intricate and deeper properties of nodes and their relationships within a graph. These methods utilize the power of deep neural networks to learn complex representations of graph structures, enabling improved performance in various tasks.

2.2.3.1 SDNE

The SDNE mechanism [19] employs a semi-supervised deep learning model that preserves both the first-order and second-order proximities between nodes. This dual focus enhances its ability to capture complex structural patterns within the graph , making it well-suited for tasks requiring detailed structural understanding.

2.2.3.2 GCN

The Graph Convolutional Network (GCN) method [6], [10] adapts convolutional neural network principles to graph data. By aggregating features from a node's neighbors, GCN learns powerful representations that integrate both local graph structure and node-specific features, facilitating the capture of both local and global relationships within the graph.

2.2.3.3 GAN

The GAN approach [5], [20] builds on attention mechanisms , allowing nodes to dynamically weigh the importance of their neighbors' features.
This adaptive attention leads to more precise and context-aware embeddings, improving the accuracy of tasks such as node classification and link prediction.

Despite the advances in these embedding algorithms, they predominantly capture the local structures surrounding a node, often failing to reflect the global hierarchical clustering properties that are essential for understanding the broader organization of a graph.

2.3 Advanced Embedding Algorithms

Advanced embedding algorithms are designed to address the challenge of representing the global structure of a network, aiming to capture both local and global relationships in a more holistic manner. These methods focus on uncovering hierarchical and multi-scale patterns within graphs, which are essential for understanding the broader structure of large, complex networks.

2.3.1 Walklets

The Walklets technique [15] enhances traditional random walk methods by selectively skipping nodes in random walk sequences. This approach allows for the exploration of various graph scales, thereby improving the ability to capture different levels of node relationships. By adjusting the length and connectivity of paths in its random walks, Walklets can uncover diverse structural patterns, providing a more nuanced representation of the graph.

2.3.2 HARP

HARP [3] is a hierarchical embedding method that focuses on large-scale graph structures. It progressively simplifies a graph by randomly merging connected nodes into a sequence of coarser graphs, and from these, recursively constructs embedding vectors. While this process allows for efficient large-scale graph representation, the random merging technique may fail to accurately capture the intrinsic clustering structures of the graph , potentially leading to the loss of important hierarchical information.

2.3.3 NetLay

The NetLay algorithm [9] is a Graph Network layering technique that simplifies complex graph networks by organizing them into hierarchical layers. Using random walks, it identifies and groups interconnected communities within the graph. These communities are then merged into super-nodes with new interconnections established to preserve the essential structure of the network. This method is particularly effective in capturing community structures and inter-community relationships at multiple scales.

The approach of uncovering the hierarchical structure through the constructing layered pyramid of the graph is also a defining characteristic of the ACE [11] algorithm, which is employed in this project.

3. Background

3.1 Preliminaries

3.1.1 Problem Definition of Network Embedding

Given a Graph $G =< V, E >$, the objective of network embedding is to map each node $V_i \in V$ into a low dimensional vector $v_i \in R^d (d \ll |V|)$ known as an embedding vector. The Similarity between two embedding vectors v_i and v_j reflects the relationship between the corresponding nodes V_i and V_j, in the original graph.

This approach allows the representation of the nodes of a graph in a feature matrix $M_{d \times |V|}$,which can then be further utilized for various tasks such as nodes classification, clustering, and link prediction. The embeddings help capture the structural relationships between nodes, making them more manageable for machine learning algorithms.

3.1.2 Graph Clustering Pyramid

To capture the hierarchical clustering properties of a graph, a Graph Clustering Pyramid is introduced.

A graph is modeled as a sequence of graphs $\Psi_G = \{G_0, G_1, ..., G_L\}$, where G_0 is the original graph and, G_{l+1} $(0 \leq l < L)$ is the clustering result of G_l.

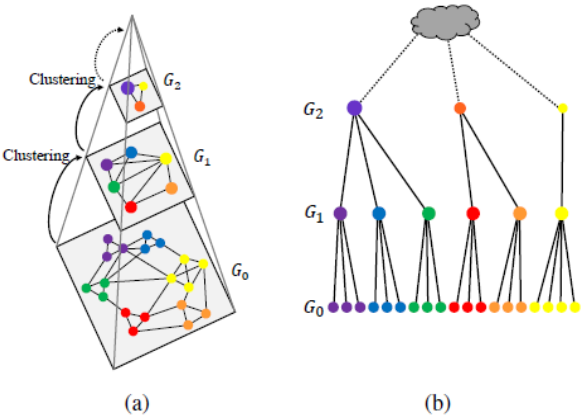


Figure 1: (a) Graph Clustering Pyramid (b) Clustering tree of nodes

In this model, the process of clustering nodes progressively simplifies the graph into multiple layers. Nodes in higher layers correspond to clusters in the lower layers, revealing the graph's complex structures through multi-layer decomposition. Higher-level graphs offer a more abstract view, clarifying the relationships between clusters and highlighting the global structures of the graph. In contrast, lower layers provide detailed local information, helping to preserve finer-scale relationships and node-level characteristics.

3.2 Multi Level Network Embedding

Algorithm 1 - MLNE (Multi-Level Network Embedding)

Input: Graph $G(V, E)$, embedding algorithm EMB,
Embedding representation size d
Output: Node representation vector embedding matrix $\Phi \in R^{|V| \times d \cdot L}$

#Apply Ant colony graph coarsening on the graph to receive the graph layers
1. $\{G_0, G_1, ..., G_L\} \leftarrow ACGraphCoarsening(G)$
2. For each layer $G_i \in \{G_0, G_1, ..., G_L\}$ do

   #Apply the embedding method on the layer
   2.1. $\Phi_i \leftarrow EMB(G_i, d)$

3. $\Phi \leftarrow \{\Phi_0, \Phi_1, ..., \Phi_L\}$

4. Return Φ

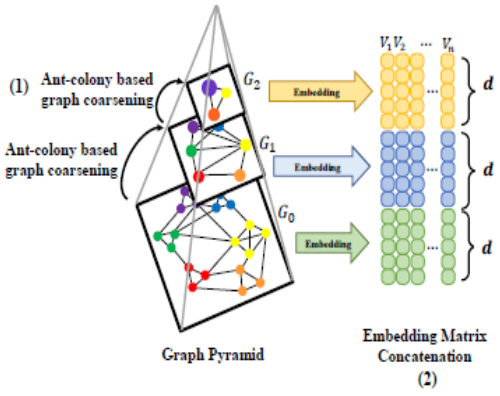


Figure 2: Multi-Level Network Embedding Framework

3.3 Ant Colony Based Graph Coarsening

The fundamental process in constructing a Graph Clustering Pyramid involves clustering closely related nodes and merging them into coarser graphs as the layers progress.
The key challenge in this process lies in accurately determining the relationship between nodes in a graph.

In an undirected graph, the local context of two nodes can be described by a set of closed loops. A higher number of loops that contain both nodes suggests a stronger relationship between them, while smaller loops indicate a closer , more direct connection.

To identify the close connections, which are defined by loops, we proposed the use of an Ant Colony Optimization (ACO) [7]- based algorithm known as the *Ant Colony Walking* algorithm.

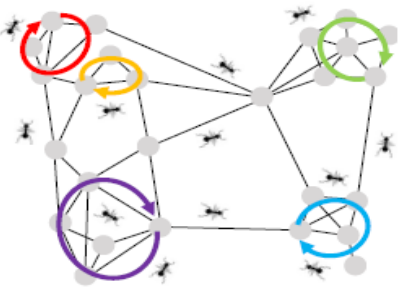


Figure 3: Ant walking in the graph

The process begins by releasing several ants from every node in the graph to start their random walks. The number of ants released from each node is proportional to

the degree of the node.
These ants are trashed with identifying loops within the graph. When a loop is found, each ant leaves a trail of pheromones along the edges that comprise the loop.

The intensity of the pheromone trail is inversely proportional to the length of the loop. This means that shorter loops result in higher pheromone levels on the edges, highlighting the strength of the connection between two nodes. Over time, edges that are part of many short loops accumulate stronger pheromone levels, indicating that they form stronger connections within the graph.

3.3.1 Ant Colony Walking

As outlined in the previous section, we used a random walking approach to identify close connections in the graph, which are defined by loops.

The Ant Colony Walking algorithm (Algorithm 2) takes a graph as the input and operates in the following steps:

1. Initialization of Transition Probabilities:
   The transition probability $P(u_i \rightarrow u_j)$ between two nodes u_i, u_j is initialized as the weight of the edge between them divided by the sum of the weights of the weights of all edges originating from node u_i.
   This is expressed as $(u_i \rightarrow u_j) = \frac{W_{ij}}{\Sigma_k W_{ik}}$, where W_{ij} is the weight of the edge between u_i and u_j and $\Sigma_k W_{ik}$ is the sum of the weights of all edge originating from u_i.

2. Ant Initialization:
   The algorithm runs for k iterations (where k is a hyper parameter). For each node in the graph, the degree of the node is calculated. The number of ants to start walking from each node is determined by the formula $n * degree$ where, n is the number of ants assigned to each degree.

3. Ant Walking:
   Each ant's goal is to discover loops within the graph. The maximum loop length that can be discovered is constrained by the hyper parameter max_steps. If an ant returns to its starting node, a loop is considered detected.

4. Pheromone Update:
   Upon discovering a loop the ant retraces its steps through the loop, scattering pheromone on the edges. The amount of pheromone deposited is inversely proportional to the loop's length, meaning shorter loops receive more pheromone, indicating stronger connections. These pheromone amounts are stored in a matrix.

5. Recalculation of Transition Probabilities:
   After each iteration, the algorithm recalculates the transition probabilities based on the pheromone levels on each edge. This is done using the following formula:

$$P(u_i \to u_j) = \frac{W_{ij} \cdot \rho_{ij}^{\alpha}}{\Sigma_k W_{ik} \cdot \rho_{ik}^{\alpha}}$$

   where $\rho_{i,j}$ represents the pheromone level on edge (u_i, u_j) and α is a hyper-parameter that adjusts the importance of the pheromone. This update ensures that ants are more likely to follow edges that have higher pheromone levels, signifying stronger connections that appear in more loops.

6. Final Output:
   After completing all k iterations, the algorithm returns the pheromone matrix, information about the strength of connections between nodes, based on the frequency and length of the loops discovered.

Algorithm 2 - Ant Colony Walking (ACWalk):

Input: Graph $G(V, E)$

Output: Graph edge pheromone matrix ρ

1. Initialize the transition probabilities

2. Repeat k (hyper parameter) iterations

    2.1. For every node $V_{init} \in V$

        2.1.1 $d_{init} \leftarrow Degree(V_{init})$ # d_{init} represent its degree.

        2.1.2. Repeat $|V| * d_{init}$ iterations #Simulate $|V| * d_{init}$ ant walks

            2.1.2.1. $path \leftarrow \{V_{init}\}$.

            2.1.2.2. While $|path| < max\_steps$ (hyper parameter) do

                2.1.2.2.1. Set V_{next} to the next node according the transition probabilities

                2.1.2.2.2. $path \leftarrow path \cup \{V_{next}\}$.

                2.1.1.2.3. If $v_{next} == V_{init}$ then

                    2.1.1.2.3.1. For consecutive nodes V_i, V_j in the $path$ array set

                    $\rho_{ij} \leftarrow \rho_{ij} + \frac{1}{length(path)}$

    2.2 Update the transition probabilities

3. Return ρ

3.3.2 Adaptive Threshold Selection

Observations from the ant-colony walking process reveal that the pheromone distribution on graph edges in real-world networks typically exhibits a segmented pattern [11]. This distribution.
naturally classifies edges into two distinct groups:

1. Strong relationship edges - these accumulate high pheromone levels and primarily serve as internal links within clusters.
2. Weak relationship edges – these have significantly lower pheromone levels and often function as bridges between different clusters.

To construct the Graph Clustering Pyramid, we clustered nodes connected by strong relationship edges into a single super-node in the upper-layer graphs. The challenge lies in determining an appropriate threshold to distinguish between strong and weak edges.
A straightforward approach was to merge nodes connected by edges with pheromone levels above a fixed threshold. However, real-world graphs exhibit diverse structures, making a static threshold ineffective. Instead, we employ *Adaptive Threshold Selection*, which dynamically determines an optimal threshold using the Elbow Method (Algorithm 3).

The algorithm gets as input the graph edge pheromone matrix ρ from algorithm 2. first, excluding the edges that do not contain any pheromone and saving it in a list ρ' and the length of that list in l.

After that, sorting the list ρ' in descending order and "draw" a line between two points.
one point is the "highest" point in the graph and the other one is the "lowest" in the graph. (The graph always has a shape that resembles the "L" letter).
Then iterate over the points and find the elbow point which is the point that has the maximum distance from the line, the Y value, amount of pheromone,
will be selected as the threshold.
The elbow point represents a natural cutoff where the pheromone values sharply decline, separating high-strength edges from low-strength edges. This ensures that only strongly connected nodes are merged into super-nodes, preserving the hierarchical structure of the graph.

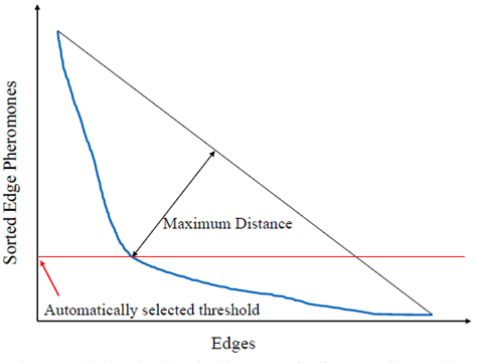


Figure 4: Select the threshold automatically according to the sorted edge pheromones distribution.

Algorithm 3 - Adaptive Threshold Selection (ATS):

Input: Graph edge pheromone matrix ρ

Output: $threshold$

#Set a list ρ' as the list of all the positive pheromone edges in the matrix.

1. $\rho' \leftarrow (\rho_{ij} \mid \rho_{ij} > 0\}$
2. $l \leftarrow length(\rho')$
3. Sort the list ρ' in descending order (ρ'_1 as the largest amount of pheromone, ρ'_l as the lowest amount)
4. Connect the points $(1,\ \rho'_1)$ and $(l,\ \rho'_l)$ by a $line$
5. For each point $\left(i, \rho'_i\right)$ do

   #Calculate the distance of the point to the line

   5.1. $distance_i \leftarrow CalcDistance(\left(i, \rho'_i\right), line)$

#Find i of the maximum distance to the line

6. $i \leftarrow max(\{distance_0,\ distance_1,\ \ldots, distance_l\})$
7. Return ρ'_i

3.3.3 Graph Clustering Pyramid Building

Once the adaptive threshold is selected, we can construct the Graph clustering pyramid building using Algorithm 4. This algorithm outlines the hierarchical coarsening process, where closely related nodes are iteratively merged into higher-level super-nodes, forming a multi-layer representation of the graph.

The algorithm receives as an input a Graph G, sets L to be 0 and G_0 to be G. There is a stopping $ratio$ (hyperparameter) that determines when to halt the coarsening process. This ratio compares the number of nodes and edges in the current coarsened graph G_l to those in the **original graph** G_0.

Each layer $G_{l'}$ is constructed by the following steps:

1. Execution of Algorithm 2 (Ant Colony Walking) f to compute the pheromone edges matrix, which captures the strength of relationships between nodes based on loop structures.

2. Execute Algorithm 3 (Adaptive Threshold Selection) to determine the optimal pheromone threshold for merging nodes.

3. Graph Coarsening: Nodes connected by edges with pheromone levels above the threshold are merged to form a new coarsened layer G_{l+1}

This process continues iteratively until the stopping criterion is met. Finally, the hierarchical layers of the Graph Clustering Pyramid are returned.

Algorithm 4 - Graph clustering pyramid building:

Input: Graph $G(V, E)$

Output: Graph Clustering Pyramid Ψ_G

1. $L \leftarrow 0$
2. $G_0 \leftarrow G$
3. While $\frac{|V_L|}{|V|} > ratio$ AND $\frac{|E_L|}{|E|} > ratio$ do
   3.1. $\rho \leftarrow ACWalk(G_L)$ # Algorithm 2
   3.2. $threshold \leftarrow ATS(\rho)$ # Algorithm 3
   3.3. $G_{L+1} \leftarrow$ Coarsen G_L based on the $threshold$
   3.4. $L \leftarrow L + 1$
4. $\Psi_G \leftarrow \{G_0, G_1, \ldots, G_L\}$
5. Return Ψ_G

3.4 Analysis of graph pyramid construction methods

The ACO-based graph coarsening algorithm (Algorithm 4) is not the only approach to constructing multi-level graph structures. Several alternative methods exist, each with its own limitations.

One alternative is random graph coarsening, as adopted by HARP [3]. In each iteration, the method randomly merges connected nodes to simplify the graph. However, since the merging process is random, nodes grouped into the same super node in the upper layers do not necessarily share strong relationships.
As a result, HARP struggles to capture the intrinsic clustering structures within the graph.

Another common approach is community detection [4], [17].
This method identifies and merges communities— groups of nodes with denser internal connections than external ones. However, even within a detected community, the strength of relationships between node pairs can vary significantly. Thus, community detection does not guarantee that all merged nodes are equally related.

In this project, we propose a multi-level network embedding (MLNE) method that leverages Ant Colony Optimization (ACO) for graph coarsening. In MLNE, nodes merged into the same super-node are assigned a shared embedding vector in the upper layers. This ensures that only nodes with strong relationships are grouped together,

leading to more meaningful hierarchical representations and potentially improving embedding quality compared to the aforementioned methods.

3.5 Multi-Level Embedding Encoding

In the Multi-Level Network Embedding (MLNE) each graph G_x ($0 \leq x \leq L$) in the Graph Clustering Pyramid $\Psi_G = \{G_0, G_1, ..., G_L\}$ serves as a distinct layer.Nodes in higher layers represent aggregated clusters of nodes from the lower layers.

At the base layer G_0 , nodes are denoted as $V_j^{(0)}$ and their corresponding ancestor nodes in a higher layer x are represented as $V_{Pj}^{(x)}$ where P denotes the parent node of j.

The embedding vectors for these nodes are $v_j^{(0)}$(at G_0) and $v_{Pj}^{(x)}$ (at layer x).
The embedding vector $v_{Pj}^{(x)}$ is referred to as the x-layer embedding vector of $V_j^{(0)}$.

These embedding vectors are structured into layer-specific matrices M^x, where each matrix has dimension is $d \times |V_0|$, capturing embeddings at different levels.
The matrices are then concatenated to construct a comprehensive mixed embedding matrix $M_{dL \times |V_0|}$, effectively integrating information from multiple layers.
This structured approach enables seamless integration with existing graph embedding algorithms, generating multi-level node embeddings without requiring modifications to the original methods.

Consequently, ACE [11] offers flexibility and scalability, allowing the application of diverse embedding techniques within this framework.

4. Proposed Research Plan

To detect irrelevant citations, we propose the following research plan:

Dataset and Graph Construction
Initially, we utilized a graph $G(V, E)$ based on the CORA dataset ([12], collected from the Core website), where nodes (V) represent academic articles and edges (E) represent the citations between them.

Multi-Level Network Embedding (MLNE) and Similarity Computation

Our proposed method is executed for K iterations, following these steps:

1. Graph Modification:

- First, we randomly remove a predetermined percentage of edges from $G(V, E)$ to create the modified graph $G'(V, E')$.

2. Embedding Generation via MLNE:

- Apply Algorithm 1 - MLNE on the modified graph G' (as previously mentioned in section 3.2), constructing the graph pyramid.
- Use node2vec [8] embedding algorithm on each layer separately.
- Construct the concatenated embedding matrix $\Phi \in R^{|V|\times d\cdot L}$, where each column represent the multi-layered embedding vector, of a node across all layers

3. Cosine Similarity Calculation:

- Compute the cosine similarity between every two node embedding vectors in Φ.
- Nodes (V_i, V_j) will be considered close if their similarity is below a predefined $threshold\ t_1$.
- The results are stored in a binary matrix $M_{|V|\times|V|}$, where 0 indicates a close relationship and 1 indicates a weak or no relationship. The matrix generated in each iteration k ($1 \le k \le K$) is denoted as M_k .

4. Statistical Aggregation over K Iterations:

- At the end of the K iterations, a statistical similarity matrix M_{stat} is constructed, summarizing all K iterations.

$$M_{stat} = \begin{bmatrix} 0 & s_{1,2} & \cdots & s_{1,|V|} \\ s_{2,1} & 0 & \cdots & s_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ s_{|V|,1} & s_{|V|,2} & \cdots & 0 \end{bmatrix}$$

- The similarity score s_{ij} for a node pair $\left(V_i, V_j\right)$ is computed as:

$$s_{ij} = \frac{\sum_{k=1}^{K} (M_k)_{i,j}}{K} \cdot 100$$

- This represents the percentage of iterations where (V_i, V_j) were classified as close.

- The statistical matrix M_{stat} is symmetric, and its main diagonal consists of zeroes, indicating that each node is trivially identical to itself.

5. Identification of Irrelevant Citations

- The statistical matrix M_{stat} and a predefined threshold t_2 enables to classify node relationships:
  A pair of nodes (V_i, V_j) with similarity value $s_{ij} \geq t_2$ indicates q strong relationships and a relevant citation. Vice versa, similarity value $s_{ij} < t_2$ indicates weak relationships and an irrelevant citation.

6. Graph Refinement

- Based on these metrics, nodes in the graph which are linked by an edge e that have a weak relation, e will be considered irrelevant. Conversely, if the relation between the nodes is strong, e is likely to be significant.

- Removing edges $e \in E$ where $s_{ij} < t_2$, enable to create a refined graph $G_R(V, E_R)$ that retains only strong, reliable connections.
  This ensures that G_R accurately reflects meaningful citation relationships, reducing noise in the citation network.

Proposed research plan Pseudo-Code (ResearchPlanAlg) :

Input: CORA Graph $G(V, E)$, embedding algorithm EMB,
Embedding representation size d, Threshold t_1 ,Threshold t_2 , Percentage p,
Number of iterations K
Output: Refined Graph $G_R(V, E_R)$

1. For i from 1 to K do:

#Remove a certain percentage of the edges randomly and obtain the modified graph
1.1 $G' \leftarrow RemoveEdges(G,\ p)$

#Apply MLNE Algorithm and obtain the concatenated embedding matrix
1.2 $\Phi' \leftarrow MLNE(G',\ EMB,\ d)$

#Calculate the cosine similarity between every two embedding
Vectors of the matrix, then check the closeness based on t_1 and save the corresponding flag in the new matrix
1.3 For m from 1 to $|V|$ do

1.3.1 For n from m to $|V|$ do

1.3.1.1 $res \leftarrow \left(\frac{\vec{v_m} \cdot \vec{v_n}}{\left|\vec{v_m}\right| \cdot \left|\vec{v_n}\right|}\right)$

1.3.1.2 If $res < t_1$ then

1.3.1.2.1 $(M_i)_{m,n} \leftarrow 0$

1.3.1.3 If $res \geq t_1$ then

1.3.1.3.1 $(M_i)_{m,n} \leftarrow 1$

#Calculate the statistical matrix
2. For m from 1 to $|V|$ do

2.1 For n from m to $|V|$ do

2.1.1 $(M_{stat})_{i,j} \leftarrow \frac{\sum_{1}^{K}(M_k)_{i,j}}{K} \cdot 100$

#Copy the CORA graph into a new graph
3. $G_R \leftarrow G(V, E)$

#Iterate over each value in the average matrix
4. For each $value_{m,n}$ in M_{stat} do

#If the value is above the threshold
4.1 If $value_{m,n} < t_2$ then

#Remove the edge between the corresponding nodes
4.1.1 $G_R \leftarrow RemoveEdge(G_R, m, n)$

5. Return G_R

5. Evaluation/Verification Plan

To assess the effectiveness of the proposed method, we designed an evaluation framework based on controlled graph modifications and quantitative performance metrics.

Experimental Setup

- A predetermined percentage of random edges was added to the original CORA-based graph $G(V, E)$ to create a modified graph $G'(V', E')$.
- The set of added edges was denoted as E', representing artificial connections that are likely to link weakly related nodes.
- The proposed research plan (Section 4) was executed on G', generating the refined graph $G_R(V, E_R)$.

Performance Metric: Edge Removal Rate

- Since the added edges E' were introduced randomly, they should be classified as irrelevant and removed by the proposed model.
- The success rate of the algorithm was evaluated based on the percentage of edges from E' that were eliminated in the final output G_R:

$$Success\ Rate = \left(\frac{\left|\left(E' - E_R\right)\right|}{|E'|}\right) \times 100$$

Where:

$\left(E' - E_R\right)$ represents the subset of added edges successfully removed.

$|E'|$ is the total number of added edges.

Statistical Validation

- To ensure robustness and statistical reliability, the evaluation experiment was repeated multiple times with different random edge additions.

- The final success rate was obtained by averaging the results across all trials.

This evaluation methodology ensures that the proposed approach effectively identifies and removes weak relationships, refining the citation network and improving its structural integrity.

Evaluation plan Pseudo-Code:

Input: CORA Graph $G(V, E)$, embedding algorithm EMB,
Embedding representation size d, Threshold t_1, Threshold t_2, Percentage p,
Number of iterations K
Output: $SuccessRate$

#Adding a certain percentage of the edges randomly and obtain the modified graph

1. $G', e' \leftarrow AddingEdges(G,\ p)$

2. $G_R \leftarrow ResearchPlanAlg(G',\ EMB,\ d,\ t_1,\ t_2,\ p,\ K)$

3. $Count \leftarrow 0$

4. For edge e in e' do
   3.1 If $e \notin E_R$ then
      3.1.1 $Count \leftarrow Count\ +\ 1$

5. $SuccessRate \leftarrow \frac{Count}{|e'|} \cdot 100$

6. Return $SuccessRate$

6. Research Process

Our project was inspired by the paper “ACE: Ant Colony based Multi-level Network Embedding for Hierarchical Graph Representation Learning” [11], which introduces a Multi-level Network Embedding (MLNE) approach utilizing Ant Colony Optimization (ACO). Under the guidance of our academic advisor, we tailored this outlined framework to address our specific research objective: identifying relevant citations within academic articles.

Understanding the Baseline Approach

At the outset, we examined the source code associated with the ACE paper to gain a comprehensive understanding of its methodology.
This involved:

- Comparing the implementation with the theoretical framework described in the paper.
- Identifying key steps in the MLNE pipeline and the sequence of operations.

However, the original code presented several challenges:

- It was implemented using a combination of Cython and C++, technologies unfamiliar to us.
- The code had not been updated in approximately five years, leading to compatibility issues with modern libraries.
- Several functions and dependencies were deprecated or no longer supported, preventing the code from running in its original form.

Code Rewriting and Adaptation

To address these challenges, we:

- Refactored the code to ensure compatibility with modern programming environments.
- Updated outdated dependencies, replacing deprecated functions with equivalent alternatives.
- Modified the implementation to align with our research objectives and verification process.

Once we successfully executed the updated code, we proceeded to the experimentation phase, focusing on fine-tuning hyperparameters to optimize performance.

Computational Challenges and Resource Constraints

During experimentation, we encountered an unexpected challenge:

- Each experiment run required up to 12 hours on standard hardware, significantly impacting the research timeline.
- The computational demands highlighted the need for more powerful resources, such as GPUs, to accelerate processing.

These constraints required careful adaptation and resource planning to balance feasibility with the research timeline, ensuring successful completion of the project.

7. Tools and Resources

Throughout the development of our project, we utilized a variety of tools and libraries that were instrumental in implementing and optimizing our citation prediction system. Below is an overview of the key technologies employed:

1. **PyTtorch**:
   PyTorch is an open-source deep learning framework that provides flexibility and efficiency for building and training neural networks. It played a crucial role in implementing the Multi-Level Network Embedding (MLNE) and optimizing the ant colony-based hierarchical graph learning model.

2. **NetworkX**:
   NetworkX is a Python library designed for the creation, manipulation, and study of complex networks and graphs. It was extensively used to model the academic citation network, enabling efficient graph-based analysis and experimentation.

3. **NumPy Documentation**:
   NumPy is a fundamental package for numerical computing in Python. Its array processing capabilities were essential for handling numerical data and performing matrix operations which were critical to the functionality of our system.

4. **Cython**:
   Cython is a programming language that facilitates writing C extensions for Python, allowing performance optimization. It was used primarily to analyze and adapt computationally intensive components from the original project's code.

5. **Scikit-learn**:
   Scikit-learn is a versatile machine learning library for Python. It was leveraged for implementing key machine learning algorithms such as regression, classification, and clustering, which were integral to various aspects of our experiments.

8. Challenges and Solutions

8.1 Understanding the Paper's Concepts

One of the primary challenges we encountered was understanding the processes and core concepts presented in the reference paper. This paper focused on machine learning techniques and graph analysis, which required us to expand our knowledge in these areas before proceeding with the implementation.

Solution:

To address this challenge, we dedicated significant time to deepen our knowledge in fundamental topics in machine learning and graph analysis to establish a strong theoretical foundation. Our approach included:

- leveraging online educational resources: We utilized video tutorials on platforms such as YouTube to gain visual explanations of complex concepts.
- Utilizing AI-assisted learning: we employed ChatGPT to clarify technical questions and assist with understanding unfamiliar terms and methodologies.
- Reviewing official documentation: we explored the official documentation of key libraries and technologies, including PyTorch, NetworkX, and NumPy to gain a deeper understanding of their functionalities and best practices.

This multi-faceted approach enabled us to comprehensively grasp the required concepts ensuring a smoother transition into the implementation phase.

8.2 Outdated and Unfamiliar Code

The original code provided by the authors was written over five years ago, relying on outdated libraries, deprecated methods, and unfamiliar technologies such as Python, C++, and Cython.

Solution:

To address these challenges, we undertook a systematic process of debugging and updating the code. We replaced deprecated methods and libraries with their upgraded equivalents, , ensuring compatibility with the latest versions of the tools and dependencies. Furthermore, we needed to understand the role of Cython in the project. This required us to study the role of Cython in the project, as it was unfamiliar to us.

To achieve this, we:

- Studied the purpose of Cython as a tool for writing Python code with C-like performance.
- Explored how Cython interacted with the other components of the project.
- Learned the Cython syntax, which was a new skill for us, by referring to the official Cython documentation.
- Conducted small-scale experiments to familiarize ourselves with Cython's structure and syntax, and integrated Cython code into our Python workflows.

Through this effort, we successfully gained the necessary skills to analyze, adapt and optimize the Cython components, ensuring they worked seamlessly with the updated libraries and dependencies.

8.3 Errors in Running the Original Code

Running the original code led to numerous errors , hindering initial progress.

Solution:

We tackled the errors systematically, reviewing the error messages and addressing each issue incrementally through code rewrites and updates. Debugging was particularly challenging due to complexity and the number of errors, especially those related to Cython components. Cython errors were notably difficult to debug in an IDE, as standard debugging tools often failed to r pinpoint the root causes of the issues.

Initially, we explored specialized tools for debugging Cython code, but these efforts were unsuccessful. Consequently, we resorted to manual debugging,, which required an in- depth understanding of Cython's syntax and functionality. This process was time-intensive and required careful attention to detail to resolve the problems effectively.

One of the most challenging issues involved an iterator class used to iterate over the graph edges. The original code relied on deprecated iteration methods, which were no longer supported by the updated libraries. To resolve this, we had to completely reimplement the entire class. This required designing a new iterator class and coding its methods in Cython—a language we were still learning at the time. Despite our initial lack of familiarity with Cython, we consulted documentation, analyzed the codebase, and conducted incremental testing to ensure the new implementation was functional and efficient.

By systematically addressing each issue, we successfully modernized the code, making it compatible with the latest tools and dependencies. This allowed us to progress and continue with the project.

8.4 Hardware Limitations

During the experimentation phase, our personal computer hardware proved insufficient for running the experiments efficiently, leading to execution times of up to 12 hours per experiment.

Solution:

To overcome this limitation, we collaborated with the IT department at our institution to explore the possibility of utilizing more powerful institutional servers. In parallel, we also explored cloud-based solutions and GPU services that could accelerate the computations and improve overall efficiency. Ultimately, we decided to leverage the institution's virtual desktop infrastructure (VDI), which provided access to more powerful GPUs compared to our personal computers. This upgrade significantly reduced the runtime of our experiments, enabling us to complete the tasks much faster and stay on track with our project timeline.

8.5 Classifying and Analyzing Graph Edges

One of the key challenges in our project was determining how to classify weak edges in the graph and how to store and analyze the experimental results effectively.

Solution:

We initially used cosine similarity to calculate the proximity of nodes, which allowed us to classify edges as strong or weak based on a predefined numerical threshold. The classification results were stored in a matrix, enabling us to perform further calculations and analysis.

At first, we planned to normalize the cosine similarity results to a custom range using a sigmoid function, aiming to map the similarity scores into a bounded range for more consistent interpretation. However, after gathering and analyzing the experimental results, we realized that the normalization process did not align well with the objectives and the nature of the data. Specifically, the sigmoid function distorted the distribution of the results, making it harder to classify the edges accurately.

After re-evaluating our approach, we decided to remove the normalization step r, and work directly with the raw cosine similarity values. This change allowed us to preserve the integrity of the similarity scores.

To further improve the classification, we introduced an additional threshold during subsequent iterations, to ensure the edges were classified as either strong or weak with greater accuracy.

This iterative process of adjusting thresholds and analyzing the results ultimately led to a robust classification mechanism that successfully met the goals of the project.

9. Results and Conclusions

9.1 Evaluation Plan Results

To verify the proposed solution, we applied our Evaluation Plan algorithm, testing multiple hyperparameters while considering the following key assumptions:

1. Threshold $t_1 = 0.9$: The refined graph should retain only edges representing strong relationships. Thus, we set a cosine similarity threshold of 0.9 for edge classification in each iteration.
2. Edge Removal Rate: A fixed 30% edges were randomly removed in each iteration.
3. Number of iterations: We conducted - 30 Iterations to ensure stability and robustness in our results.
4. Number of Layers: The graph pyramid structure was maintained at 8 layers to preserve hierarchical embeddings.
5. Alpha Parameter ($\alpha = 0.5$) We used a fixed α value as recommended in the original paper.

To validate the model using the evaluation plan, we conducted multiple experiments, progressively adjusting key parameters to assess their impact on performance.

Embedding vector size	T1	T2	Edge removal (%)	Iterations(K)	Alpha	Pyramid scales	Success Rate (%)
128	0.9	50	30	30	0.5	8	84.396
256		40					98.989
512		35					96.926
1024		30					96.968

Table 1: Evaluation Plan Results

1. Effect of Embedding Vector Size on Cosine Similarity

In each experiment, we gradually increased the embedding vector size, which directly influences the magnitude of the embedding vectors at each pyramid layer.
A larger embedding vector size allowed for more precise cosine similarity measurements, as the additional features provided a richer representation of node relationships. This improvement enhanced the accuracy of similarity assessments across graph layers.

2. Effect of Threshold t_2 on Edge Classification

Simultaneously, we progressively decreased the threshold t_2, which determines whether an edge is classified as strong or weak. This threshold, expressed as a percentage, represents the fraction of iterations in which an edge is considered strong:

- If an edge is classified as strong in a sufficient number of iterations - exceeding the set threshold - it is retained in the graph.
- Conversely, if an edge falls below this threshold, it is deemed weak and removed.

The results, summarized in Table 1, reveal that as the embedding vector size increases, the threshold t_2 percentage decreases. This gradual adjustment ensures that while vector similarity accuracy improves, a lower threshold is required to differentiate between strong and weak edges effectively.

3. Key Observations

- The experiments yielded promising results, with over 95% of the artificially and randomly added edges being removed. This aligns with expectations, as most of these edges were introduced at random and thus were likely to be weak connections.
- A slight decrease in success rates as embedding vector sizes increased.
  This suggests that achieving an optimal balance between embedding size and threshold t_2 requires further fine-tuning.
- Future experiments could explore more precise parameter adjustments to ensure the model's robustness and optimize edge classification accuracy.

9.2 Research Plan Results

After validating the model and obtaining good results, we conducted further experiments to analyze the behavior of the original CORA graph - without the addition of artificial edges.

These experiments aimed to assess how effectively the model could classify and refine edges within a real-world citation network.

Experiment Setup

The experimental parameters were selected based on the optimal results obtained in previous tests. We used an embedding vector size of 256, a threshold t_2 of 40 and edge removal percentages: 30%, 40%, and 50% (varied across experiments).

Embedding vector size	T1	T2	Edge removal (%)	Iterations(K)	Alpha	Pyramid scales	Weak edges classified (%)
256	0.9	50	30	30	0.5	8	51.25
			40				53.22
			50				62.59

Table 2: Proposed Model Results

According to Table 2, Each experiment differed in the percentage of edges randomly removed per iteration during the Research Plan process. The one involved removing 30% of the edges, the second one 40%, and the third one 50%.

Observations and Findings

- Edge Removal and Weak Edge Classification

  A direct relationship was observed between the percentage of edges removed randomly in each iteration and the percentage of edges ultimately classified as weak. As shown in Table 2:

  - The higher the random removal rate in each iteration, the higher the percentage of edges classified as weak by the end of the process.
  - This suggests that processing a smaller graph at each step improves the identification of weak edges.

- Alignment with CORA Graph Properties

  Prior research indicates that in the CORA citation network, approximately two-thirds of the edges represent weak relationships (~66.6%).
  Our experimental results align closely with this statistic, further validating the model's effectiveness. In future experiments, we anticipate achieving a

classification accuracy even closer to 66.6%. However, time constraints limited our ability to refine the model further.

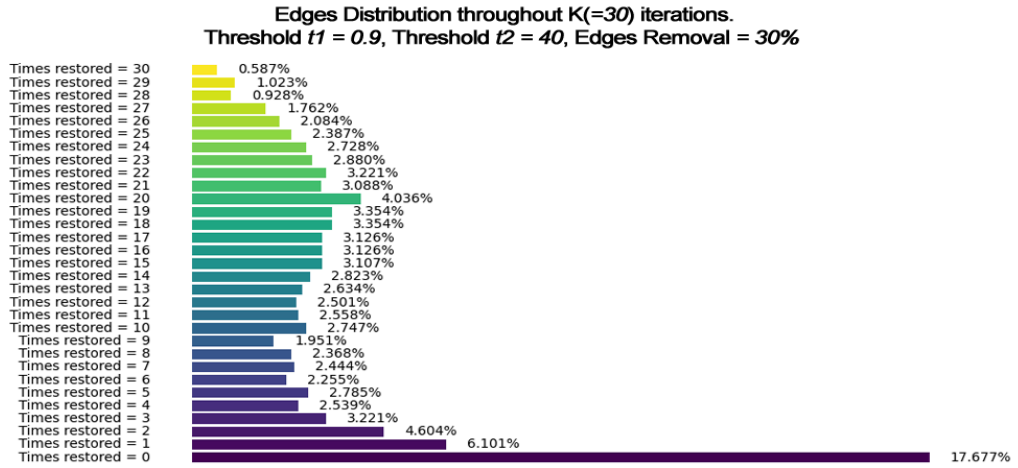


Figure 5: Research Plan Test no.1

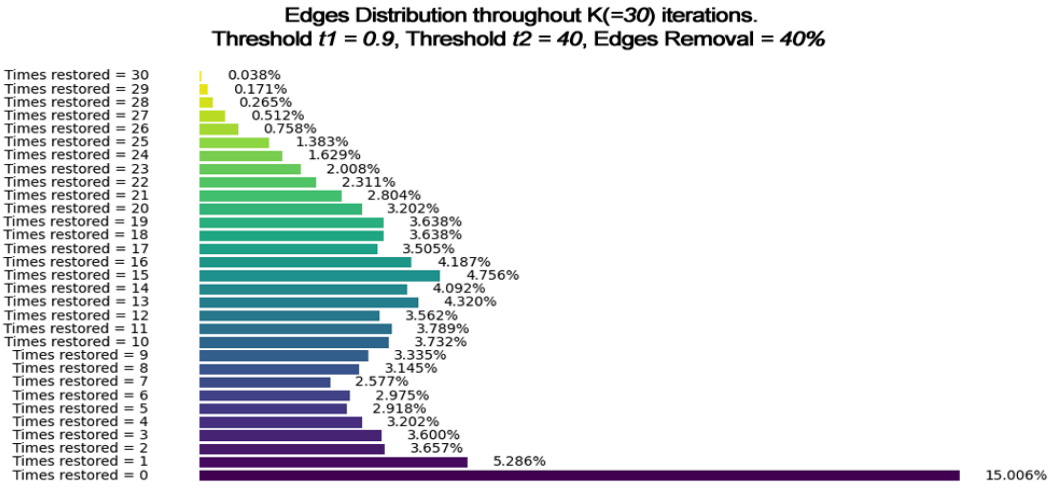


Figure 6: Research Plan Test no.2

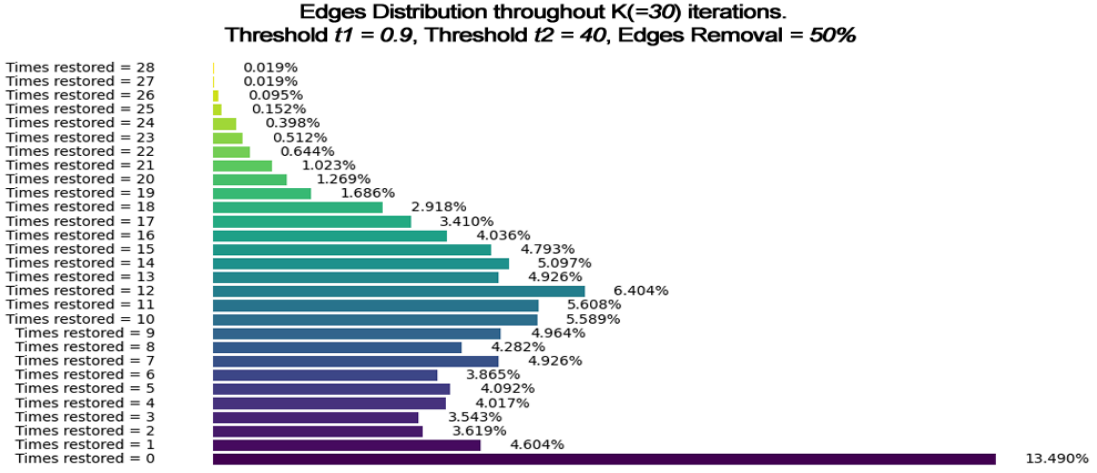


Figure 7: Research Plan Test no.3

Graph Distribution and Symmetry Analysis

The distribution graphs (Figures 5-7) reveal that:

- The symmetry of the graph tends toward a lower representation of the edges recovered. Thus, the majority of edges classified as weak are located in the lower range of the distribution.

- The peak of the distribution occurs at 0 recovered edges, reinforcing the conclusion that a large proportion of CORA edges are inherently weak.

This trend highlights the model's ability to successfully filter out weak connections and suggests that further refinements in threshold tuning could enhance accuracy in future studies.

9.3 Validating Article Connections in Practice

To further evaluate the effectiveness of our model, we analyzed the edges classified as the strongest and weakest throughout our experiments. Specifically, we:

1. tracked edges that consistently passing the Threshold t_2 in each iteration (strong connections).

2. Identified edges that failed to meet Threshold t_2 in any iteration (weak connections).

3. Examined the corresponding articles to assess the validity of the connections our model identified.

Tables 3-4 display the strongest and weakest edges. After identifying these edges, we used their corresponding labels to trace and analyze the articles they connect.

Edge	Title of ID1	Title of ID2
(636098, 260121)	NP-Completeness of Searches for Smallest Possible Feature Sets	The wake-sleep algorithm for unsupervised neural networks
(50381, 28489)	Training algorithms for hidden Markov models using entropy-based distance functions	Learning without state-estimation in Partially Observable Markovian Decision Processes
(40151, 24966)	Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation	A practical Bayesian framework for backpropagation networks
(27241, 27246)	Error Reduction through Learning Multiple Descriptions	Working Notes of AAAI Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms
(646195, 260121)	Applications and extensions of MCMC in IRT: Multiple item types, missing data, and rated responses	The wake-sleep algorithm for unsupervised neural networks

Table 3: Top 5 strongest citations

Edge	Title of ID1	Title of ID2
(8699, 733167)	Theory revision in fault hierarchies	Soft vector quantization and the EM algorithm
(116621, 346292)	Improving Generalization with Active Learning	Hyperplane "spin" dynamics, network plasticity and back-propagation learning
(223605, 733167)	Techniques for extracting instruction level parallelism on MIMD architectures	Soft vector quantization and the EM algorithm
(93318, 110163)	Discovering Structure in Multiple Learning Tasks: The TC Algorithm	Learning to integrate multiple knowledge sources for case-based reasoning
(249421, 29492)	On the perception of time as phase: Toward an adaptive-oscillator model of rhythm	Induction of multiscale temporal structure

Table 4: Top 5 weakest citations

To validate our results and gain insights into the relevance of the identified citations, we analyzed specific examples from our dataset.

To illustrate the findings, we selected one example from each category and analyzed its relevance.

Strongest Citation Analysis:

- Edge: (646195, 260121)
- Title of Article 1: Applications and extensions of MCMC in IRT: Multiple item types, missing data, and rated responses.
- Title of Article 2: The wake-sleep algorithm for unsupervised neural networks.

Reason for Strong Citation: Both articles focus on complex statistical methods for handling incomplete datasets, demonstrating a meaningful scholarly link. The citation appears valid and well-justified.

Weakest Citation Analysis:

- Edge: (232605, 733167)
- Title of Article 1: Techniques for extracting instruction level parallelism on MIMD architectures.
- Title of Article 2: Soft vector quantization and the EM algorithm.

Reason for Weak Citation: The first article discusses parallel computing architectures, while the second focuses on machine learning algorithms. These fields are largely unrelated, suggesting that the citation may be contextually weak or irrelevant.

Key Findings and Implications

- The results reinforce the model's ability to distinguish between meaningful and questionable citations.
- Strong citations tend to connect conceptually related research, whereas weak citations often lack thematic coherence.
- This validation process provides a foundation for refining the model further, potentially incorporating additional semantic analysis techniques to enhance citation evaluation.

10. References

[1] Belkin, M., & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 14.

[2] Cao, S., Lu, W., & Xu, Q. (2015, October). GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international conference on information and knowledge management* (pp. 891-900).

[3] Chen, H., Perozzi, B., Hu, Y., & Skiena, S. (2018, April). Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

[4] Cordasco, G., & Gargano, L. (2010, December). Community detection via semi-synchronous label propagation algorithms. In *2010 IEEE International Workshop on Business Applications of Social Network Analysis (BASNA)* (pp. 1-8). IEEE.

[5] Dai, Q., Li, Q., Tang, J., & Wang, D. (2018, April). Adversarial network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

[6] Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 29.

[7] Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3), 243-278.

[8] Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 855-864).

[9] Jin, H., Xu, G., Cheng, K., Liu, J., & Wu, Z. (2022). A link prediction algorithm based on GAN. Electronics, 11(13), 2059.

[10] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[11] Lv, J., Zhong, J., Liang, J., & Yang, Z. (2019). ACE: Ant colony based multi-level network embedding for hierarchical graph representation learning. *IEEE Access*, 7, 73970-73982.

[12] McCallum, A. K., Nigam, K., Rennie, J., & Seymore, K. (2000). Automating the construction of internet portals with machine learning. *Information Retrieval*, 3, 127-163.

[13] Ou, M., Cui, P., Pei, J., Zhang, Z., & Zhu, W. (2016, August). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1105-1114).

[14] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 701-710).

[15] Perozzi, B., Kulkarni, V., Chen, H., & Skiena, S. (2017, July). Don't Walk, Skip! Online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (pp. 258-265).

[16] Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500), 2323-2326.

[17] Shen, H., Cheng, X., Cai, K., & Hu, M. B. (2009). Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications*, 388(8), 1706-1712.

[18] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015, May). LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 1067-1077).

[19] Wang, D., Cui, P., & Zhu, W. (2016, August). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1225-1234).

[20] Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., ... & Guo, M. (2018, April). GraphGAN: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No.1).