Node2Vec-based Graph Embedding Neural Network Implementation for Shortest Path Prediction

Qiwen Xue
Department of Computer Science
Wenzhou Kean University
Wenzhou, China
xueq@kean.edu

Chengyu Zhang
Department of Computer Science
Wenzhou Kean University
Wenzhou, China
zhchengy@kean.edu

Hemu He
Department of Computer Science
Wenzhou Kean University
Wenzhou, China
hehe@kean.edu

Kei Wai Hadassah Wong
Department of Computer Science
Wenzhou Kean University
Wenzhou, China
kwong@kean.edu

Linjie Ying
Department of Computer Science
Wenzhou Kean University
Wenzhou, China
yinglin@kean.edu

***Abstract*—Shortest path computation is a basic graph analysis task, but it can become computationally intensive for large-scale graphs using classical graph traversal techniques like Breadth-First Search. This paper describes a learning-based shortest path prediction system that leverages Node2Vec graph representation learning with a regression neural network model. Node2Vec is used to represent structural properties of graph nodes as low-dimensional vector embeddings. A landmark-based graph node pair sampling approach is used to obtain graph node pairs with ground truth shortest path distances that can be utilized as supervisory signals for learning. A multi-layer perceptron network is used to predict shortest path distances for each selected graph node pair by concatenating their vector embeddings. Experiments were conducted on synthetic social network graph datasets to show that the proposed prediction system can predict shortest path distances with small prediction error, thereby establishing its feasibility as a rapid alternative to classical shortest path algorithms.**

***Index Terms*—Shortest path, Node2Vec, Neural networks, Graph embedding**

I. Introduction

Data represented by graphs is found to be abundant in various real-world applications such as social networking platforms, communication networks, transport networks, and biological graphs of interactions. Of the various graph analysis problems that exist in literature, one of the most basic is finding the shortest path between nodes in an weighted graph, which is basically finding the minimum number of edges that need to be traversed between two nodes. Shortest path length is of critical importance for tasks such as calculating node centrality.

Classic algorithms for calculating shortest paths like Breadth-First Search (BFS) for finding distances in an unweighted graph or Dijkstra's algorithm for calculating distances in a weighted graph promise precise distance information with correctness assurance. But these approaches need graph traversal for calculating the distance between nodes for each query; therefore, their computational complexity increases for big graphs with multiple distance queries. In large-scale and low-latency–critical scenarios, such as emergency response and network transmission, heuristic algorithms are no longer particularly suitable. Although they often achieve high accuracy, they struggle to cope with highly dynamic and large-scale environments.

Recently, graph representation learning techniques were developed with graph embedding algorithms that embed nodes in low-dimensional vector spaces, while, at the same time, their structural properties are retained in the original graph. Of these approaches, Node2Vec is one of the models that received considerable attention for capturing both local and global structural properties of the graph using biased random walks. Node2Vec embeds nodes in vector spaces by combining both breadth-first and depth-first graph sampling techniques.

Based on graph embeddings, there are various works that apply machine learning models to approximate graph theoretical properties such as shortest path distances using machine learning models. These models do not necessarily require traversing the graph to compute distances; they try to learn a mapping that takes node embeddings as input to predict distances. These models become effective once trained for predicting distances, which act as an alternative solution for applications with large graphs.

This paper presents a graph embedding–based neural network framework for shortest path prediction in weighted graphs. Node2Vec is first applied to learn vector representations for all nodes, capturing the underlying structural characteristics of the graph. To construct the supervised training dataset, a landmark-based strategy is adopted, in which exact shortest path distances between landmark nodes and other nodes are computed using Dijkstra's algorithm. For each sampled node pair, their corresponding embeddings are concatenated to form a joint feature representation, which is then fed into a multi-layer perceptron (MLP) to predict the

shortest path distance between the two nodes.

The advantages of the proposed learning-based framework over classical shortest path algorithms can be summarized as follows:

- **Reduced online computational complexity.** Classical shortest path algorithms, such as Dijkstra's algorithm, require graph traversal for each query, resulting in a time complexity of $O(E + V \log V)$ for single-source queries in sparse graphs. In contrast, once the proposed model is trained offline, shortest path distances can be predicted via a single forward pass of the neural network, leading to an online inference complexity that is effectively $O(1)$ with respect to the graph size.
- **Elimination of repeated graph traversal.** Traditional algorithms must repeatedly traverse the graph when multiple shortest path queries are performed, which incurs substantial cumulative computational cost on large graphs. The proposed approach avoids explicit graph traversal during inference, significantly reducing runtime overhead for query-intensive tasks.
- **Suitability for large-scale graphs.** By decoupling inference complexity from the number of nodes and edges, the proposed framework scales favorably to large graphs, making it well suited for large-scale network scenarios.
- **Low-latency response capability.** Since distance estimation is achieved through direct neural network inference, the proposed method enables fast response times, which is critical for latency-sensitive applications.
- **Practical applicability to approximate shortest path estimation.** Although the framework provides approximate rather than exact shortest path distances, it is particularly effective in scenarios where approximate solutions are acceptable, such as real-time systems and dynamic environments.

Contrary to other techniques that train a unique model for distinct graphs, the proposed framework employs a universal neural network that is shared and jointly trained on multiple graphs. This yields the ability of the learning model to capture universal distance patterns among other benefits of this approach. To test the effectiveness of the proposed framework, some experiments are carried out using synthetic scale-free social network graphs modeled using the Barabasi-Albert model. These experiments verify that it is possible for the proposed framework to predict the shortest path distances with a small mean absolute error (MAE) and mean relative error (MRE).

The remainder of this paper is organized as follows. Section II reviews related work on classical shortest path algorithms, landmark-based approximation methods, and recent advances in graph representation learning. Section III presents the proposed shortest path prediction framework, including the Node2Vec-based embedding generation, landmark-based training data construction, and the neural network regression model. Section IV describes the experimental setup and evaluation methodology, followed by a detailed analysis of the results on synthetic graph datasets. Section V discusses the experimental findings, analyzes the strengths and limitations of the proposed approach, and compares it with existing state-of-the-art methods. Finally, Section VI concludes the paper and outlines directions for future work.

II. Related Work

A. Shortest Path

The shortest path problem can be defined as finding the most efficient and cheap route in terms of total cost, weight and distance between two nodes/vertices in a graph [**?**]. There are four variants of the shortest path problem. The single-source, single-destination, single-pair, and all-pairs. Single-source shortest path problems, abbreviated as SSSP, are the tasks of computing the shortest path from a source node *S* to all other vertices. Examples of efficient algorithms that solve the SSSP are Dijkstra's algorithm and the Breadth-First Search (BFS). Dijkstra computes the SSSP in *O(m + n log n)* time and the BFS in *O (m + n)* time. [1] These algorithms fail in time complexity when processing large graphs with millions of nodes. To solve this problem, researchers have created scalable algorithms. These algorithms trade perfect accuracy with significant improvements in time complexities.

Landmark-based approaches are one such family of scalable algorithms that are better suited to solve the shortest path problems in large graphs [2]. These algorithms select a fixed set of landmark nodes as reference points. During data preprocessing, the exact distance from each landmark to all other nodes is computed using either BFS or Dijkstra. The total time needed to compute the exact distances is *O(1(n + m))*. At query time, the algorithm uses the triangle inequality with the precomputed landmark distances to estimate the approximate distance between the requested nodes *u* and *v* in *O(1)* time. For any landmark *L*, the distance *d = (u, v)* must satisfy the inequality:

$$d(u, v) \geq |d(u, L) - d(v, L)| \tag{1}$$

By taking the maximum of all the landmarks, landmark-based algorithms provide a lower bound approximation. Although these algorithms are able to greatly decrease time complexity, they produce relatively high error rates, limiting their practical usage in the types of applications that they can be used in [3], [4]. As said by Salehi Rizi, et al., "Although landmark-based algorithms do not provide strong theoretical guarantees on approximation quality, they have been shown to perform well in practice, scaling up to graphs with millions of edges with acceptable accuracy and response times of under one second per query." [1].

The inherent limitations of landmark-based algorithms include only lower bound approximation, no theoretical accuracy guarantees, and each query requiring checking all landmarks.

Although landmark-based algorithms have many limitations, systems like Orion and Rigel have demonstrated that with careful landmark selection of high-degree nodes or nodes with high between-ness centrality, the error rates may be decreased to acceptable rates [2], [3]. These two systems have been used

in real world social networks and web graphs with acceptable error rates and are able to answer millions of queries per second.

Recent advances in graph representation learning have produced new techniques that include neural networks as effective in capturing complex graph structure in low dimensional embeddings.

B. Graph Embeddings

While landmark based methods address scalability, they rely solely on distance computations and geometric approximations, making them inefficient and unsuitable for machine learning algorithms. Graph representation learning produced graph embedding techniques to automate the solution to this problem. They have emerged as powerful tools for representing graph structures. Graph embedding aims to learn a mapping that maps each node to a low dimensional vector while simultaneously preserving important graph properties. Traditional methods such as Multidimensional Scaling (MDS) [5] and Principal Component Analysis (PCA) [6] rely on matrix decomposition techniques for solutions. In large-scale networks they have proven to be inefficient and suffer from at least quadratic complexity in number of nodes [7]. Recent neural network advances inspired by Word2vec such as DeepWalk and Node2vec have revolutionized graph embedding techniques by treating the random walks in the graphs as sentences. Deepwalk samples random walks from graphs and predicts the probability of nodes in local neighborhoods [8]. Building further on the concepts of Deepwalk, Node2vec embedding introduces parameters that can control the walks, making them biased random walks [9]. These parameters allow Node2vec to balance between having a more BFS or DFS sampling behavior. Node2vec's flexibility to adjust the locality or globality of walks has made it particularly effective for downstream machine learning tasks. Some primary effective uses include link embedding, node classification and distance approximation i large-scale social networks [7]. Its distance approximation in large-scale social network graphs make Node2vec the most suitable graph embedding technique to utilize in our implementation.

III. Methodology

The methodology used in this paper was proposed by Salehi Rizi et al. in February 2020. [1] This section describes our take on their approach as described in their research paper and our implementation of Dijkstra's Algorithm and Node2vec embedding techniques.

A. Overview

The original study proposed a method to approximate the shortest path distances in large graphs using graph embeddings and neural networks as an alternative to other expensive methods like BFS or Dijkstra's Algorithm. Their proposed method had three main steps. The first step entailed learning graph embeddings using embedding techniques to create vector representations for each node in the graph. They compared two embedding techniques for mean absolute error (MAE) and mean relative error (MRE). The relative error is defined as,

$$RE = \frac{\left|\hat{d} - d\right|}{d} \tag{2}$$

where d is the actual distance measured by the algorithm and $\hat{d}$ is the approximation [1]. MRE is calculated as a percentage of the actual distance. MAE is simply the actual distance subtracted by the predicted distance. For MAE, lower values are better results, however, MRE is biased towards longer paths as the longer paths produce smaller values. The experiment results of the original study show that the best results were achieved by the Node2vec embeddings of size 128. The embeddings achieved an average MAE of 15 % across the four datasets used in the original experiment. With these prior results, we decided to implement using Node2vec embeddings.

B. Problem Formalization

To define the problem, let *G = (V, E)* represent an weighted, directed graph consisting of n vertices and m edges. For any pair of nodes *u, v* $\in$ *V*, let *d(u, v)* denote the true shortest path distance between them, measured as the minimum number of edges traversed. The aim of our implementation is to develop a function $\hat{d}$ that efficiently approximates *d(u, v)* with minimal error while maintaining computational tractability for large-scale graphs. Therefore the formal problem can be defined as:

Given a trained model *M* and graph embeddings

$$\varphi : V \to \mathbb{R}^d \tag{3}$$

that maps each node to a d-dimensional vector space, we seek to construct:

$$\hat{d} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+ \tag{4}$$

subject to query time constraints that enable real-time distance estimations independent of graph sizes.

C. Overall Approach

Our methodology consists of three primary phases: graph embedding generation, supervised learning data construction and neural network training for distance regression. Phase 1 involves using the algorithms for Node2vec embedding to generate continuous vector representations of graph nodes. Node2vec embedding techniques are able to capture structural properties via biased random walks. These random walks create latent spaces where topological proximity can correlate with geometric proximity. Instead of computing all $O(n)^2$ pairwise distances, we plan on selecting a strategic subset of landmark nodes in phase 2 and execute Dijkstra's Algorithm from each node. This yields a training dataset of known distances with $O(ln$ complexity where $l \ll n$. Phase 3 includes training a feedforward neural network to learn the mapping from embedded pairs to distance values. The network takes combined embedding representations as input and outputs predicted shortest path distances.

This approach achieves constant time *O(1)* distance queries after linear time *O(n)* preprocessing, making it suitable for interactive applications on large-scale graphs with millions of nodes.

D. Dijkstra's Algorithm

Dijkstra's algorithm was used for computing the shortest path distances in weighted graphs, graphs where edges have associated costs/weights. In our implementation, we utilize the algorithm as a base generator for training data. After a random sample of 20 nodes as landmarks in phase 1 are completed, Dijkstra's algorithm is used to compute the distances from each landmark to all other nodes. We then create training pairs consisting of a landmark, a node and a distance. to filter the data, we exclude self loops where the node == the landmark. Phase 1 also includes optimally dropping the direct neighbor nodes, which mean nodes with distance-1, and limiting maximum distance focus on relevant ranges []

Dijkstra was chosen for its realistic distance captures, its accurate labeling and linear complexity for sparse graphs.

E. Node2vec Embeddings

Node2vec generates node embeddings by optimizing the possibility of persevering network neighborhoods through random walks. The core concept of the Node2vec algorithm is random walking. Random walks are sequences of nodes where each step moves from the current node to one of its neighbors. These walks capture the local and global structure around each node, and are treated like "sentences" and nodes like "words". In Node2vec, these random walks are biased random walks controlled by 2 parameters: p and q.

Parameter p is the return parameter. It controls the likelihood of returning to the previous node. A low p value determines that the biased random walk has a higher probability to backtrack and stay local. A high p lowers the possibility of backtracking, making the walk a less local and more global walk. The q parameter is the in-out parameter, controlling the exploration and exploitation trade-off. Low q values indicate a higher possibility of exploring further nodes, displaying a BFS-like behavior. High values limit the exploration to stay near the starting point, displaying DFS-like behavior [9].

To decide which neighbor node to go to when performing a walk, the transition probability is used. It can be defined as: The probability of moving to node x from current node t, given we came from node v, is proportional to π_{tx} if an edge exists, and zero otherwise. The formula can be defined as:

$$P(x|v,t) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } x \in N(v) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Where the unnormalized transition probability π_{tx} is based on the parameters p and q.

$$\pi_{vx} = \alpha_{(t,x)} \cdot w_{vx} \tag{6}$$

And a_{tx} is the bias weight determined by the shortest path distance d_{tx} between the previous node t and the destination node x.

$$\alpha_{(t,x)} = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \quad \text{(back-tracking to } t\text{)} \\ 1 & \text{if } d_{tx} = 1 \quad \text{(neighbor of } t\text{)} \\ \frac{1}{q} & \text{if } d_{tx} = 2 \quad \text{(neighbor of neighbor of } t\text{)} \end{cases} \tag{7}$$

in the above equations, *N(v)* defines the set of neighbors of node *v*. w_{vx} is the static edge weight between the nodes *v* and *x*. *Z* defines the normalization constant, ensuring that all the probabilities sum up to 1 for all neighbors of *v*. Parameters *p* (return parameter) and *q*, (in-out parameter) were used in the equations to control the probability of visiting previously visited nodes and whether the walk is global or local [9].

Node2vec was used because of its flexible exploration in natural ability to capture local structure while also being able to scale up to global structures. Its embeddings provide smooth continuous features that neural networks can effectively process. For shortest path problems, Node2vec embeddings are able to capture topological proximity, enabling a neural network to learn to map embedding pairs to distance estimates.

F. Algorithm Description

The proposed framework follows a sequential pipeline to approximate shortest-path distances in graphs. For each input graph, node embeddings are first generated using the Node2Vec algorithm, which encodes local and global structural information into low-dimensional vectors. To construct supervised training data efficiently, a landmark-based sampling strategy is adopted. A small set of nodes is randomly selected as landmarks, and exact shortest-path distances from each landmark to all other nodes are computed using a classical shortest path algorithm.

Based on these distances, training samples are formed by pairing each landmark with other nodes in the graph, while excluding trivial cases such as self-pairs and direct neighbors. For each node pair, their corresponding embeddings are combined to form a feature representation, which is then used as input to a feedforward neural network. The network is trained to regress the shortest-path distance between the two nodes.

Once trained, the model can be applied to unseen graphs. Node embeddings are computed once per graph, and shortest-path distances between arbitrary node pairs are predicted through a single forward pass of the neural network, without performing any online graph traversal. An example of the predicted distance behavior is illustrated in Fig. 1.

```
pair: (3, 2000) pred: 90.16939544677734 actual: 81
```

Fig. 1. Training MSE using BFS with unit edge weights.

Algorithm 1 Training: Node2Vec + Landmark Sampling + Shared MLP Regression

Require: Training graph set $\mathcal{G}_{train}$; embedding dim d; #landmarks L; walk length ℓ_{train}; #walks W; combine op $\oplus$ (concat); epochs per graph T; batch size B; learning rate η

Ensure: Trained distance regressor f_θ

Initialize MLP regressor f_θ with output constrained to be non-negative
for each graph $G \in \mathcal{G}_{train}$ **do**
  $\Phi \leftarrow \text{NODE2VEC}(G;\ d,\ \ell_{train},\ W)$ ▷ $\Phi(v) \in \mathbb{R}^d$
  $S \leftarrow \text{SAMPLELANDMARKS}(V(G), L)$
  $\mathcal{P} \leftarrow \emptyset$
  for each landmark $s \in S$ **do**
    $D_s(\cdot) \leftarrow \text{SHORTESTPATH}(G, s)$ ▷ BFS if unit weights, else Dijkstra
    for each node $v \in V(G)$ **do**
      if $v = s$ **or** $D_s(v) = 1$ **then continue**
      end if
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(s, v, D_s(v))\}$
    end for
  end for
  Build dataset $\mathcal{D}_G = \{(x_{sv}, y_{sv})\}$ where $x_{sv} = \Phi(s) \oplus \Phi(v)$ and $y_{sv} = D_s(v)$
  for $t = 1$ to T **do**
    for each mini-batch (X, Y) from $\mathcal{D}_G$ of size B **do**
      $\hat{Y} \leftarrow f_\theta(X)$
      $\mathcal{L} \leftarrow \text{MSE}(\hat{Y}, Y)$
      $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$ ▷ Adam optimizer in implementation
    end for
  end for
end for
return f_θ

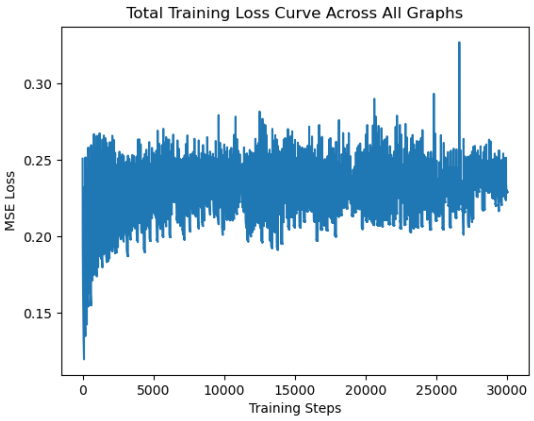

Fig. 2. Training MSE using BFS with unit edge weights.

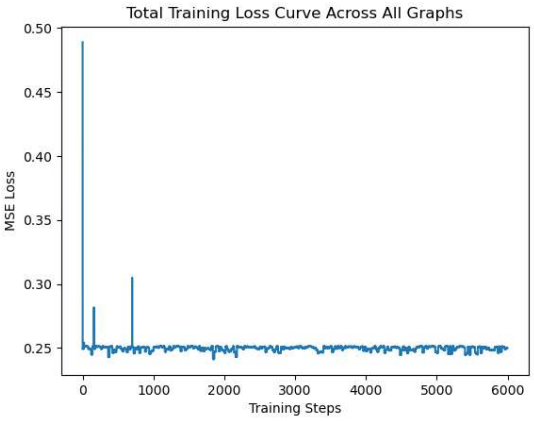

Fig. 3. Testing MSE using BFS with unit edge weights.

IV. Experimental Evaluation

In this section, we evaluate the experimental results of the proposed method. In our study, node2vec was applied to each graph independently to generate graph embeddings. For each graph, a set of landmarks was sampled, and the resulting landmark–node pairs were used as the training data for the neural network model. This section presents the experimental setup for both the training and testing datasets, the parameters adopted in the experiments, the evaluation metrics, and the corresponding experimental results.

A. Dataset

The size of the dataset used in this study was motivated by a publicly available Facebook social network dataset, and a graph model consisting of 4,092 nodes was adopted. We first simulated the Facebook network structure by generating 100 Barabási–Albert (BA) graphs. The edge weight between any two connected nodes was set to 1 by default, and the Breadth-First Search (BFS) algorithm was employed to compute the shortest path in terms of the minimum number of edges, which under unit edge weights directly corresponds to the shortest path length.

However, during training, the experimental performance was unsatisfactory. Despite extensive tuning of hyperparameters, the network consistently exhibited severe *representation collapse*, where the learned function converged toward an averaged constant value. As a result, regardless of the input feature matrix, the final output collapsed to a fixed value.

Figures 2 and 3 illustrate a total of 30,000 and 6,000 training epochs, respectively. It can be observed that the loss values fluctuate within a narrow range.

We attribute this path collapse phenomenon to the following factors:

- **Small edge values.** Under unit edge weights, the neural network struggled to capture meaningful relational information between nodes. The distance values of node pairs were easily smoothed out during function approximation, making it difficult for the model to learn discriminative representations.
- **Overly dense graph structure.** An analysis of the Facebook dataset revealed that user-to-user relationships

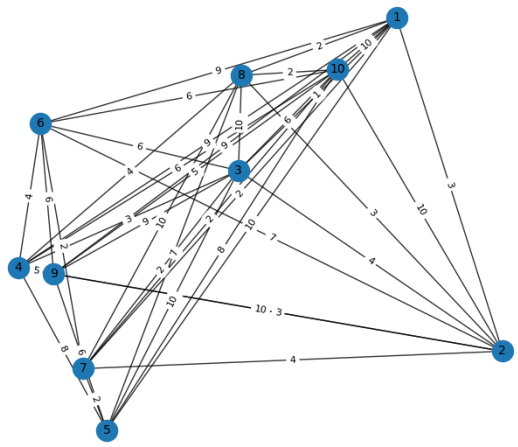

Fig. 4. Social Network Graph

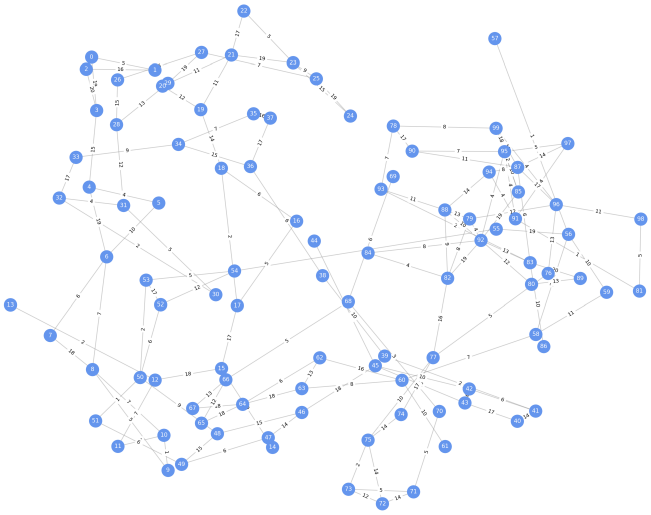

Fig. 5. Experimental Graph

in social networks are highly compact, with the distance between any two users typically not exceeding five hops. This limited variation in inter-node distances significantly reduced the structural diversity available for learning, thereby hindering the network's ability to distinguish relationships among different users. Figure 4 illustrates a dense social network model.

Based on these observations, we conclude that dense social networks with unit edge weights are not well suited for neural network–based analysis. Furthermore, BFS-derived shortest-path distances are not an appropriate foundation for neural network learning in this context.

In practical experiments, Python was used to generate the training graphs. Graph sparsity was explicitly controlled by configuring the number of layers, the number of edges between adjacent layers, and the base number of random intra-layer edges. This design allowed precise regulation of node connectivity to construct sparse graph structures. The parameters were set as follows:

- Number of layers: 30
- Number of edges between adjacent layers: 3
- Base number of random intra-layer edges: 2

Figure 5 shows an undirected weighted graph with 100 nodes randomly generated using the aforementioned architecture.

Using this configuration, 100 training graphs were generated. Graph embedding was performed independently for each graph, and the resulting embeddings were concatenated and jointly fed into a single neural network for representation learning.

B. Parameters and Experimental Environment

All experiments were implemented in Python, with graph processing conducted using the NetworkX library and neural network training performed using PyTorch. Node embeddings were generated using the node2vec algorithm. The experiments were executed on a workstation equipped with GPU acceleration when available; otherwise, CPU execution was adopted.

For graph representation learning, node2vec was applied independently to each graph. The embedding dimensionality was set to 128. During training, the random walk length was set to 40, and 50 walks were generated per node, while the number of worker threads was fixed at 4. In the testing phase, the walk length was increased to 80 to enhance global structural exploration, whereas the remaining parameters were kept unchanged.

For distance supervision, landmark-based sampling was employed. A fixed number of landmarks (20 by default) was randomly sampled from each graph. For each landmark, shortest-path distances to other nodes were computed using Dijkstra's algorithm. Node pairs with distance equal to 1 were excluded to avoid trivial neighborhood information, and node pairs exceeding a predefined maximum distance were optionally filtered.

The node pair representations were constructed by concatenating the embeddings of the two target nodes. These representations were then fed into a multilayer perceptron (MLP) for distance regression. The MLP consisted of two hidden layers with 128 neurons each, followed by ReLU activations, and a Softplus activation at the output layer to ensure non-negative distance predictions.

Model training was conducted using the Adam optimizer with a learning rate of 0.05. The mean squared error (MSE) loss was adopted as the optimization objective. Each graph was used sequentially to train the same model for 30 epochs, with a batch size of 64. During evaluation, a batch size of 256 was used. Performance was assessed using mean absolute error (MAE) and mean relative error (MRE) between predicted and ground-truth shortest-path distances.

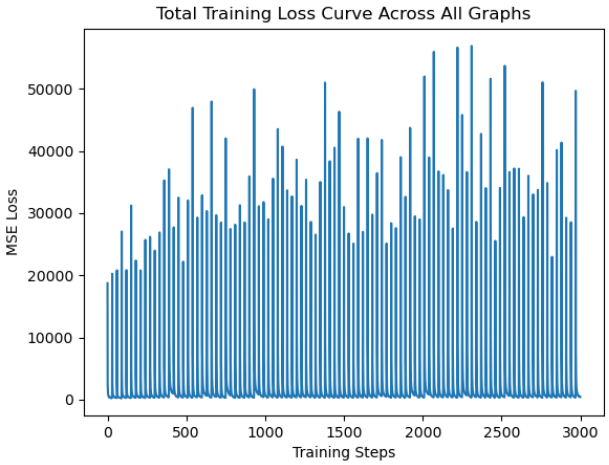

Fig. 6. Experimental Graph

C. Experiment Result

The training results are illustrated in Fig. 6. In total, 100 graphs were used for training, and each graph was trained for 30 epochs. Our observations indicate that the mean absolute error (MAE) converges to a relatively stable level after approximately 30 epochs. However, the final MAE values predominantly fluctuate within the range of 200 to 300.

We attribute this behavior primarily to the following factors:

- **Limited ability of the neural network to capture path information.** For the experimental graphs used in this study, the sparsity and density patterns across layers were controlled as constants. Although graph embeddings are able to partially encode structural features, the neural network receives supervision solely in the form of shortest path distance values. Since the graph generation process involves randomness and provides limited explicit path-related information, the learned representations may be insufficient to accurately predict distances, leading to relatively high prediction errors.
- **Insufficient model capacity.** The neural network employed in this work is a basic three-layer multilayer perceptron (MLP). For graphs with approximately 4,000 nodes, such a network architecture may be underparameterized. As a result, the model may converge before fully capturing the underlying structural information. Increasing the network depth or width will be explored in future work to enhance model expressiveness.
- **Suboptimal hyperparameter configuration.** Although the number of training epochs is limited, the MAE values exhibit stable convergence behavior, suggesting that the performance bottleneck is unlikely to stem from insufficient training iterations. Instead, there remains room for further optimization of hyperparameters, which may lead to improved predictive accuracy.

V. Discussion

This study employs a feedforward neural network to calculate the mean relative error and mean absolute error during the test phase. The experiments demonstrate that node2vec embeddings with dimension 128 achieve optimal performance. This indicates that our method achieves considerably low approximation error across different network scales, significantly outperforming the baseline approach.

A. Embedding Dimension Selection and Performance Analysis

The number of dimensions plays a crucial role in model performance. While high-dimensional embeddings can reduce errors, they simultaneously increase the computational cost for embedding generation and distance calculation, necessitating a balance between accuracy and efficiency. The features learned by node2vec are fundamentally determined by its random walk strategy. Since node2vec adopts short random walks for exploration, it effectively learns the structural information of local neighborhoods. In scale-free networks, the average shortest path distance grows logarithmically, making the shortest path distance itself a local feature. When the embedding dimension is 128, node2vec successfully learns distance features between nodes, achieving a favorable balance between accuracy and computational efficiency. As the dimension increases, the model's expressive capability strengthens, but the benefits gradually diminish. Our experiments demonstrate that dimension 128 is an ideal choice point, ensuring high prediction accuracy while avoiding excessive computational overhead.

B. Error Distribution Across Different Path Lengths

We analyzed the prediction accuracy across different path lengths. The results reveal that when using node2vec embeddings, longer paths lead to larger errors. This phenomenon can be attributed to two factors: First, the insufficient number of long-distance samples in the training set results in inadequate model learning for long paths. Second, node2vec's random walk strategy excels at capturing local structural information but has relatively limited capability in learning structural features of distant nodes. This finding suggests that in practical applications, our method is more suitable for distance prediction tasks involving short paths. Despite this limitation, our method still performs excellently on short paths, which aligns perfectly with the requirements of real-world application scenarios such as social networks, where most practical queries concentrate on shorter path distances.

C. The Role of Binary Operators

To generate feature representations for training node pairs, we employ simple binary operators to combine the learned embeddings of individual nodes. This compositional approach enables node2vec embeddings to be applied to distance prediction tasks for node pairs. Binary operators exhibit inconsistent performance across different datasets and dimensions. This variation may be related to the topological characteristics of different networks: Facebook, as a social network, exhibits strong community structure, while YouTube's video

association network presents different connectivity patterns. Future research will delve into the following questions: Why do different operators perform differently across networks? Are there more optimal operator combination approaches? Which topological features of networks determine operator effectiveness?

VI. Limitations and Future Works

Although this project validates the effectiveness of embedding-based approximation algorithms in static scale-free networks, we plan to extend this research in three key dimensions to enhance the model's robustness and applicability in complex real-world scenarios.

A. Adaptability to Dynamic Network Evolution

The current Node2Vec embedding algorithm is inherently static. When applied to real-world social networks that are highly dynamic—characterized by the continuous addition or removal of nodes and edges—the current model faces a significant maintenance bottleneck. Any minor topological perturbation currently necessitates re-running random walks and retraining the embedding model for the entire graph, resulting in substantial computational overhead and latency.

To address this, future work will incorporate Dynamic Graph Representation Learning techniques [10]. Specifically, we plan to explore incremental learning frameworks that allow the model to capture temporal evolution patterns. By updating node embeddings in real-time for only the affected local subgraphs without retraining from scratch, we can reduce the maintenance complexity from a global scale to a local scale. This capability would enable the shortest path prediction service to handle streaming data, significantly enhancing its utility in latency-sensitive applications such as real-time recommendation systems.

B. End-to-End Optimization via Hybrid Architectures

The current project utilizes a two-stage pipeline consisting of feature extraction (Node2Vec) and distance regression (MLP). This decoupled approach introduces potential information loss, as the unsupervised embeddings are generated to preserve general neighborhood structures rather than being explicitly optimized for the specific task of shortest path prediction. Consequently, the subsequent regressor may fail to fully leverage deep topological features.

Inspired by recent research on joint planning with CNN-GNN architectures [11], we propose investigating Hybrid Neural Architectures in future iterations. On one hand, Graph Neural Networks (GNNs), such as GraphSAGE or GAT, could replace the shallow Node2Vec model. GNNs utilize message-passing mechanisms to dynamically aggregate multi-hop neighborhood information, offering stronger inductive reasoning capabilities. On the other hand, for networks containing spatial attributes, Convolutional Neural Networks (CNNs) could be fused to extract grid-like features. By constructing an end-to-end differentiable model, we can simultaneously optimize both graph representation learning and distance prediction, potentially breaking through existing accuracy bottlenecks for long-distance paths and complex topologies.

C. Integration of Hyperbolic Geometry

Our current implementation forces nodes into a flat Euclidean embedding space. However, scale-free networks often exhibit latent tree-like hierarchical structures. Representing these hierarchies in Euclidean space results in severe geometric distortion, particularly when representing long-tail links far from central nodes.

Future iterations will focus on implementing Poincaré embeddings to represent node vectors in Hyperbolic space. According to theoretical studies, the exponential expansion property of hyperbolic geometry allows it to preserve tree-like hierarchical structures with near-perfect fidelity using fewer dimensions. We anticipate that migrating from Euclidean to Hyperbolic geometry will allow the model to more accurately capture the latent hierarchical relationships in social networks, thereby significantly reducing the relative error of long-path predictions without increasing computational resource requirements.

VII. Conclusion

As the scale of modern information networks grows exponentially, the $O(n + m)$ time complexity required by exact shortest path algorithms has become an insurmountable performance barrier for large-scale real-time graph analysis [12]. To address this challenge, this project designed, implemented, and evaluated a deep learning-based heuristic approximation framework. This framework creatively combines Node2Vec graph embedding techniques with a neural network regressor, aiming to find the optimal balance between precision and computational speed.

Distinct from previous studies that were primarily limited to transductive learning within single static graphs [1], this project introduced two critical methodological innovations:

First, our dataset departs from the original dense graph architecture. We implemented a sparse graph generation framework in Python, which allows explicit control over the number of nodes per layer and the sparsity of inter-node connections.

Second, the constructed graphs are applicable to weighted graph settings. The original BFS-based approach is limited to graphs with unit edge weights, whereas in our experiments, Dijkstra's algorithm was employed to obtain accurate shortest-path distances for both the training and testing sets. This extension enhances the applicability of the proposed framework to real-world scenarios.

Third, the proposed design effectively avoids the representation collapse observed in the original setting. Under the previous graph structure and algorithmic configuration, the differences among graphs were extremely subtle, causing the model to converge toward averaged outputs and leading to collapse. In contrast, the graphs considered in our study exhibit richer structural information and larger inter-node distances. When combined with weighted edges, this design enables

the model to more effectively capture discriminative graph features.

Empirical results indicate that the proposed method successfully reduces query response time to constant time $O(1)$. This represents a significant speed advantage over traditional algorithms while maintaining high prediction accuracy within local neighborhoods. Although challenges remain regarding long-path prediction and dynamic adaptability, our findings conclude that embedding-based neural approximation offers a highly efficient and viable solution for processing massive, latency-sensitive graph applications.

REFERENCES

[1] F. Salehi Rizi, J. Schloetterer, and M. Granitzer, "Shortest path distance approximation using deep learning techniques," *arXiv preprint arXiv:2002.05257v1*, 2020.

[2] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao, "Orion: shortest path estimation for large social graphs," *Networks*, vol. 1, p. 5, 2010.

[3] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao, "Efficient shortest paths on massive social graphs," in *2011 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pp. 77–86, IEEE, 2011.

[4] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum, "Fast and accurate estimation of shortest paths in large graphs," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pp. 499–508, ACM, 2010.

[5] J. B. Kruskal and M. Wish, *Multidimensional Scaling*, vol. 11. Sage, 1978.

[6] I. T. Jolliffe, "Principal component analysis and factor analysis," *Principal Component Analysis*, pp. 115–128, 1986.

[7] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *arXiv preprint arXiv:1705.02801*, 2017.

[8] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710, ACM, 2014.

[9] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 855–864, ACM, 2016.

[10] L. Yang, C. Chatelain, and S. Adam, "Dynamic graph representation learning with neural networks: A survey," *IEEE Access*, vol. 12, pp. 34233–34268, 2024.

[11] S. Bose, T. Anitha, N. Maheswaran, and D. Prabhu, "Adaptive deep learning techniques for real-time shortest path optimization in drone ambulance operations during disaster," in *Proceedings of the 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, IEEE, 2024.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, 4th ed., 2022.