IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 May 2023, accepted 30 May 2023, date of publication 5 June 2023, date of current version 9 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3283029

RESEARCH ARTICLE

LeL-GNN: Learnable Edge Sampling and Line Based Graph Neural Network for Link Prediction

MD GOLAM MORSHED[1,2], (Member, IEEE), TANGINA SULTANA[1,3], AND YOUNG-KOO LEE[1], (Member, IEEE)

[1]Department of Computer Science and Engineering, Kyung Hee University, Global Campus, Yongin 17104, South Korea
[2]Department of Computer Science and Engineering, International University of Business Agriculture and Technology, Dhaka 1230, Bangladesh
[3]Department of Electronics and Communication Engineering, Hajee Mohammad Danesh Science and Technology University, Dinajpur 5200, Bangladesh

Corresponding author: Young-Koo Lee (yklee@khu.ac.kr)

This work was supported by the Institute for Information & Communications Technology Promotion (IITP) funded by the Korean Government (MSIP) through the Development of a Distributed Graph DBMS for Intelligent Processing of Big Graphs under Grant IITP-2022-2021-0-00859.

ABSTRACT Graph neural networks lose a lot of their computing power when more network layers are added. As a result, the majority of existing graph neural networks have a shallow depth of learning. Over-smoothing and information loss are two of the key issues that restrict graph neural networks from going deeper. As network depth goes up, the embeddings of all the nodes eventually converge on the same value, which separates output representations from input vectors and causes over-smoothing. Moreover, layers of graph pooling are required in a deep learning model to retrieve specified features for prediction, which results in some degree of information loss. In this research, we present a new and multi-scale approach for overcoming these constraints by using concepts from graph theory, namely learnable edge sampling and line graphs. An edge-sampling mechanism that selects a particular number of edges through a learning parameter before training reduces oversmoothing, and the issue of information loss is alleviated using a line graph technique that converts the original graph into a similar line graph. Our method of edge sampling preserves the core spectral features of the graph without affecting its fundamental structure. Our suggested technique outperforms state-of-the-art models on publicly available datasets of diverse applications while having minimal constraints and great training skills.

INDEX TERMS Edge sampling, deep graph neural networks, line graph, link prediction.

I. INTRODUCTION

Numerous data is represented as graph topologies, where a set of nodes are connected in unexpected ways via edges. There are many kinds of graphs in the world, such as knowledge graphs [1], social media [2], image graphs [3], molecular interaction [4], etc. Graphs provide robust inductive biases to allow relational inference and comprehensive abstraction [5], which makes learning on graphs essential not just for the study of graph data but also for generalized interpretation. Graph neural networks (GNNs) [6] have grown in popularity as the go-to method for studying graphs in recent years. The primary impetus for the development of GNNs was to transfer the performance of neural network models (NNs) from tabular data to the graph field.

The core idea behind GNN is that, up to a given number of iterations, the feature vectors of a node and its neighbors can be combined using a recursive neighborhood aggregation function. Using a properly specified aggregation function, it is shown that such message passing is as effective as the Weisfeiler-Lehman (WL) network isomorphism analysis [7], which is known to differentiate a large class of graphs [8]. We focus on Graph Convolutional Networks (GCNs) [9], [10], [11], [12], [13], [14], a core family of GNNs that generalize the convolution operation from pictures to graphs. Specifically, this research focuses on how GCNs have been used for the purpose of link prediction.

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Fiumara.

The depth of the convolutional neural network (CNN) is known to have a significant impact on performance in the vision domain. The success of CNN suggests that stacking additional layers will provide GCN with greater expressivity, allowing it to define richer neighbor topology. Moreover, describing graph topology necessitates sufficiently deep structures, which is why the research and development of deep GCN is so significant. It has been demonstrated in [15], [16] that when the depth is limited, GCNs are not able to acquire a graph in an instant or predict certain graph features.

However, it is challenging to achieve the goal of creating GCN that is both expressive and deep. In reality, over-smoothing [17] and information loss hinder the expressive potential of deep GCN. An intuitive understanding of over-smoothing is that an indefinitely deep GCN's output tends to converge toward a space with little in the way of distinguishable information between nodes due to the mixing of neighborhood characteristics via graph convolution. Over-smoothing, from a training standpoint, eliminates valuable discriminative information from the input and results in poor trainability. Usually, the topology of a subgraph is represented by characteristics learned using graph neural networks [18]. This implies that graph pooling layers [19], [20], [21] need to be used to calculate a feature vector of a specific size from the whole graph, which might lead to a loss of information. For instance, just a subset of the graph's nodes may be chosen to represent it in a sort pooling operation [18]. Additionally, it usually takes longer for a training cycle to converge in a graph neural network that has pooling layers.

Many different approaches have been offered to investigate how to construct deep GCNs [2], [9], [11], [22]. However, none of them provides sufficiently expressive design, and it is still uncertain whether or not such architectures are guaranteed to avoid or alleviate over-smoothing. After first modeling GCN as Laplacian smoothing, [22] discovered that the characteristics of vertices inside each linked element of the graph converge to the same result. Building on the work of [17], Oono and Suzuki [23] established that GCN converges to a subspace formed on the basis of node degrees. However, this conclusion is restricted to conventional GCN [2] without consideration of alternative architectures.

There have been several heuristic approaches offered to address the issue of graph link prediction in a variety of contexts. However, there are still difficulties in deciding which heuristic functions to use when presented with a novel network. To address these issues, graph neural network-based link prediction models (SEAL) have been suggested to automatically infer heuristic methods from the k-hop neighborhood [24]. Their approach outperformed state-of-the-art methods on several different types of graphs. For each pair of target nodes, the SEAL model will extract an k-hop bounded subgraph and make a link prediction depending upon the structure of the enclosed subgraph. Thus, the challenge of predicting links is transformed into a graph classification issue, where the model predicts the presence of a connection between two vertices given only their surrounding subgraphs. Even though the SEAL performs well in many sorts of graphs, it does have some drawbacks because of its use of pooling operations.

Our proposed approach, Learnable Edge Sampling and Line Graph based Graph Neural Network (Lel-GNN), solves both the problems of over smoothing and information loss. We suggest directly learning the properties of a target edge, rather than pulling them out of the complete surrounding subgraph, to circumvent the information loss that occurs in pooling layers. Learning node embeddings is more efficient than feature extraction on the whole network. Successful results in learning node embeddings have been achieved using graph convolution networks [2], [12], [25], [26]. However, when it comes to learning edge embeddings from graphs, graph convolution layers fall short. We suggest transforming the original enclosing subgraph into a line graph as a solution to this problem.

Learnable edge sampling is a technique wherein a percentage of the input graph's edges are selected using a parameter before learning. Applying edge sampling to GCN training has various advantages. In the first place, edge-sampling may be seen as a method of augmenting data. By selecting edges from a graph, we make new graphs with random distortions. This makes the original data more diverse and hard to predict. Edge sampling may be thought of as a message-passing reducer as well. Messages in GCNs are transferred between neighboring nodes via edge connections. By selecting out some edges, the connections between nodes become less dense, and even when GCN is trained very deeply, over-smoothing is reduced to some degree.

In this paper, we provide a unified Learnable Edge Sampling and Line Graph based Graph Neural Network (Lel-GNN) architecture for link prediction, which simultaneously learns features at the node and edge levels. The overall ideas that make up the suggested paradigm are shown in Fig. 1. Line graph nodes all represent edges from the original graph. It's also possible for the topological data to be kept in good shape while transforming. Layers of graph convolution may be applied straight away in order to obtain the line graph embeddings of nodes. To infer whether or not a connection will exist between nodes in the original network, we utilize the line graph's node embeddings as features of the edges. So, in the framework we've suggested, the problem of predicting links can be seen as a problem with classifying nodes. The proposed Lel-GNN framework is a full-fledged, end-to-end trainable architecture that makes use of both node/edge integration and feature cross models. The proposed Lel-GNN architecture has the following contributions over current link prediction methods:

- We provide a novel and multi-scale deep graph neural network model that can retrieve and integrate features at different scales effectively, which in turn helps to alleviate oversmoothing and information loss caused by layer stacking and pooling operations, respectively.

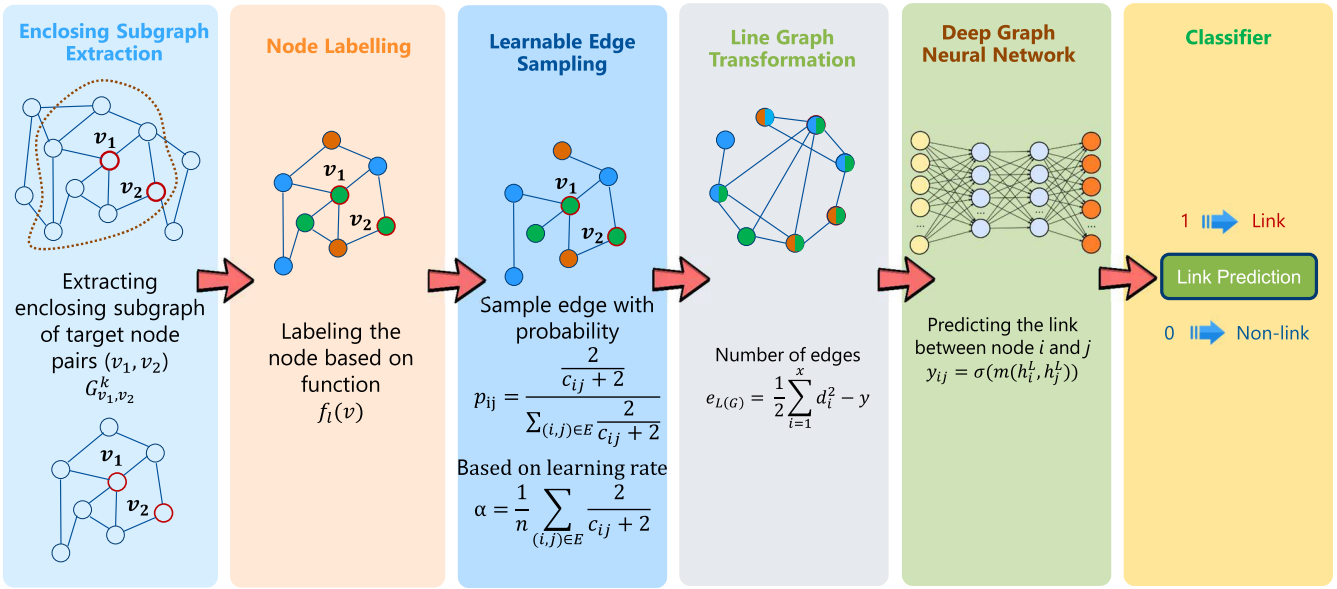

FIGURE 1. Proposed LeL-GNN architecture.

- The framework employs an end-to-end model to take advantage of learnable edge sampling and line graph, both of which are critical to lowering the convergence speed to the same value and reducing information loss.
- Our model performs edge sampling using a learning parameter that takes into account the need to preserve a robust spectral quality. Therefore, the original graph structure and any relevant data will be preserved throughout the sampling process.
- We test our methods using publicly available datasets from various fields. Our suggested technique outperforms the majority of existing baseline methods and the prior state-of-the-art model, indicating its potential for widespread use.

The remaining parts of the paper are organized as follows: In section II, we describe past and recent efforts in the field of link prediction using graph neural networks and how they connect to one another. In section III we described the preliminary notations related to graph. In section IV, we'll go through the architecture, methods, and algorithm that were used to create this link prediction technique. Section V describes the evaluation metrics, datasets, baseline methods and experimental setup of our research. Section VI displays the outcome and result analysis of our model. This paper is concluded in the section VII.

II. RELATED WORKS

The pioneering work on GCNs is found in [14], which introduces a new method of graph convolution that takes into account both spectral and spatial perspectives. Later, in [2], [27], [28], [29], and [30], spectral-based GCNs are subjected to enhancements, extensions, and approximations. Spatial-based GCNs have emerged as a viable solution to the scaling problem faced by spectral-based GCNs on massive graphs [12], [25], [26], [31]. Several new approaches have emerged in recent years for quick learning of graph representations using sample-based techniques, such as node-wise sampling methods [12], layer-wise strategies [10], [13], and graph-wise techniques [32], [33]. Dropout on the edges of attention has been specifically studied by GAT [34]. Despite being a post-processed variant of edge dropping preceding attention computing, the connection to over-smoothing is never investigated in [34]. However, in this study, we have explicitly introduced the concept of learnable edge sampling and offered a thorough explanation of its value in reducing over-smoothing.

Despite the beneficial advances, the deeper expansion of GCNs is seldom mentioned in the literature. The use of the residual mechanism in attempts to construct deep GCNs can be traced back to the GCN publication [2], and, surprisingly, their results demonstrate that residual GCNs continue to perform poorly at depths of 3 and beyond. Although the authors of [17] correctly identify over-smoothing as a major challenge when building deep networks, they fail to provide any solutions. In the subsequent research [11], the authors use a customized version of PageRank that also incorporates the rooted nodes into the signal forwarding loop, therefore resolving the issue of oversmoothing. Learnable edge sampling is compatible with the deep GCN formulation method used by JKNet [9], which makes use of dense links for multi-hop message forwarding. DAGNN [35], a recent improvement to the GCN architecture, accomplishes this by first isolating the representation conversion from the transmissions and then employing a flexible adjusting strategy to strike a better balance between the data collected from each node's local and global surroundings.

Computing the similarity score between two target nodes based on their immediate neighbors is the key notion behind heuristic approaches. We may classify heuristics into three orders: first, second, and third. These are the three broad categories of heuristics regarding the highest hop of neighbors utilized in the computing mechanism. Some popular first-order heuristics for similarity estimates include looking for common neighbors, looking for nearest neighbors, and favoring attachment [36]. Methods like Adamic-Adar [37] and resource allocation [36], [38] are examples of second-order heuristics that make use of two-hop neighbors. There have also been proposals for high-order heuristics to calculate the score to determine how similar two nodes are based on the full graph. These include Katz [39], rooted PageRank [40], and SimRank [41]. It is not uncommon for high-order heuristics to outperform their low-order counterparts, although they come at a higher computational cost. There are several heuristic approaches for dealing with various graphs, making it difficult to choose the best one.

Similarity measures between target nodes may also be computed using embeddings of nodes [42]. Matrix factorization [43], stochastic block [44], etc. are examples of common embedding techniques that were used to tackle the link prediction challenge because they can learn the properties of nodes from the topology of the network. Recent developments like deepwalk [45], LINE [46], and node2vec [47] have been presented to learn node embedding using the skip-gram approach; these methods were influenced by word embedding techniques used in natural language processing applications. Using a random walk generator, Deepwalk selects the subsequent visited node evenly from the neighborhood of the existing node, creating random walks of a certain length for each vertex. Node embeddings are then learned from the newly created node sequence using the skip-gram technique. In order to recreate the graph topology, the variational graph autoencoder (GAE) [48] is suggested to use of graph convolutional neural networks for the purpose of node embedding learning. The node embedding approaches may learn useful properties from the graph, leading to respectable results in the link prediction challenge. However, in cases when the network becomes very sparse, the efficiency of connecting node embedding approaches might suffer.

To get over the shortcomings of heuristic approaches, several deep learning methods were suggested for automatically inferring the structural properties of a graph [24], [49], [50]. For this purpose, the Weisfeiler-Lehman Neural Machine [49] developed a neural network with a fully connected layer relying on a specified enclosing subgraph that targeted the two nodes of interest in order to make a link prediction between them. SEAL [24] transforms the link prediction challenge into a problem of classification which may be addressed through the graph neural networks, allowing on account of the forecast of whether a connection exist from a broad enclosed subgraph. The SEAL method uses the robust learning capabilities of graph neural networks to provide state-of-the-art outcomes on the issue of link prediction. In order to enhance SEAL's functionality on elementary graphs, a multi-scale link prediction strategy was developed [50].

III. PRELIMINARIES

A. NOTATIONS

Assume that there are N nodes in an undirected graph $G = (V, E)$, where $V = (v_1, v_2, \ldots v_N)$ is a node collection and $E = (e_{ij}|v_i, v_j \in V)$ is a set of links connecting the nodes v_i and v_j in V. The adjacency matrix $A \in R^{N\times N}$ in the topology space is defined as $A_{ij} = 1$ if $e_{ij} \in E$ and 0 otherwise. Accordingly, the node degree matrix $D \in R^{NN}$ is a diagonal matrix where $D_{ii} = \Sigma_j A_{ij}$. $x_i \in R^F (i \in 1, 2, \ldots N)$ feature vectors are associated with each node in graph G, and their collection is denoted by the matrix $X \in R^{N\times F}$. Let the Laplacian be $L_G = D - A$ for graph G, and specify the new Laplacian L_H for the output of the weighted graph H in the same way by graph sparsification. As a result, H has the strong spectral property, which is expressed as the following inequality with any x:

$$(1-\epsilon)x^T L_G x \leq x^T L_h x \leq (1+\epsilon)x^T L_G x \tag{1}$$

B. LINK PREDICTION WITH GRAPH NEURAL NETWORKS

To learn meaningful node representations, graph neural networks(GNNs) iteratively aggregate processed representations of neighboring nodes in each l-th GNN layer, given a connected graph G and a feature matrix X, as shown below:

$$h^{(l+1)} = \sigma\left(\widehat{A}h^l W^l\right) \tag{2}$$

where $\widehat{A}$ is a normalized adjacency matrix (normalized in one of many ways depending on the GNN architecture), W^l is a weight matrix that can be trained, and $h^{(0)}$ is the X-dimensional feature matrix of the node. To determine whether a given connection (i, j) exists, we utilize the $h^{(L)}$ representations of the nodes in the network, which are generated by stacking L layers of GNN.

$$y_{ij} = \sigma\left(m(h_i^L, h_j^L)\right) \tag{3}$$

where m is the function of inner product or multi layer perceptron(MLP), and h_i^L is the embedded feature of the node i from $h^{(L)}$. In this study, sigma is implemented as ReLU, which is a nonlinear function. The representation of the graph is produced by pooling the last layer representations of the nodes after L GNN layers.

$$h_G = POOLING(h_v^L | v \in V) \tag{4}$$

IV. METHODOLOGY

A. PROPOSED SYSTEM ARCHITECTURE

Subgraph extraction, feature extraction based on the subgraph structure, and classification-based link prediction make up the gist of the supervised heuristic pipeline for link prediction

problems. When performing supervised heuristic link prediction, extracting the underlying graph structure is a critical criterion. To extract features from graphs, graph convolution networks are currently the best way [2], [4], [14], [25]. In a graph convolution network, a linear transformation is used to give each node an embedding feature, which is then shared with the nodes around it.

Thus, each node is represented by a set of high-level properties drawn from itself and its immediate surroundings. Every node is mostly impacted by the nodes immediately around it, even if the node characteristic may be carried deeper by stacking additional graph convolution layers. A node can't effectively get data from distant neighbors if its representation is the same as those around it. When this happens, link prediction models might not work as well because the subgraph around it has too much redundancy. As we want to employ node embeddings to distinguish between nodes, the convergence to a single value after stacking several GNN layers is unacceptable. Each node in an L-layer GNN is surrounded by L-hops, creating a receptive field. As a result, the number of hops, representing GNN layers, rapidly increases, leading to an over-smoothing issue as a result of the receptive field's overlapping and sharing neighbors. Even more important, as we go through the layers, the model's performance gets worse because nodes next to each other share the same embedding, which leads to the vanishing gradient problem.

We present a learnable edge sampling approach that can rapidly eliminate duplicate features and change the graph into a new scale, allowing for the characteristics of a node to be efficiently propagated to its far-range neighbors. Our suggested edge sampling approach uses new subgraphs from inputs that are h-hop enclosed with fewer characteristics. To generate a range of subgraphs with varying sizes, we may aggregate the graph repeatedly. To forecast a link's existence, our proposed model takes as inputs multi-scale enclosing subgraphs and then extracts hierarchical properties. Our suggested strategy is based on the assumption that various-sized enclosing graphs might provide mutually-enhancing data for link prediction. Before moving on to the convolution layer, we also used the line graph technique to change the graph into a similar line graph. This would reduce the amount of data lost during the pooling process at the end of the chain.

B. LEARNABLE EDGE SAMPLING

Edge sampling refers to picking a random subset of edges in a graph. Let c_{ij} represent the total number of i and j's shared neighbors in the undirected graph $G = (V, E)$, where $(i, j) \in E$. For clarity, we will first focus on the situation when all c_{ij} values for all edges are already known. To put this into effect, we may either compute them precisely in parallel or use a sampling technique called ''neighbor sampling'' to get a close approximation. Using a multinomial distribution model with probability p, we randomly choose m edges from G to construct a sparsifier H. A sparsifier in the context of graphs is a subgraph that retains the connectivity features of the parent graph but has considerably fewer edges. A graph sparsifier, for example, may be used to speed up some graph algorithms by minimizing the number of edges that must be evaluated. The following equation is used to choose the edges for our proposed algorithm.

$$p_{ij} = \frac{\frac{2}{c_{ij}+2}}{\sum_{(i,j)\in E} \frac{2}{c_{ij}+2}} \tag{5}$$

We add the edge $(i, j) \in E$ on H also give it the value $k(mp_{ij})^{-1}$ if this is chosen $k \geq 1$ times. Take note that subgraph G containing edges (i, j) with c_{ij} pathways which have range 2 across i and j, the overall edge resistivity connecting i with j will be $2/(c_{ij} + 2)$. As such, it provides a ceiling on the efficient resistance of the G edge connecting points i and j.

Our sampling technique becomes more like graph sparsification with effective resistances as the number of shared neighbors grows. But as the number of shared neighbors goes down, the technique starts to resemble random sampling more and more. The following learnable parameter α is used to assess the quality of local network connections.

$$\alpha = \frac{1}{n} \sum_{(i,j)\in E} \frac{2}{c_{ij} + 2} \tag{6}$$

where c_{ij} is the number of neighbors that nodes i and j have in common. We'll demonstrate that α is connected to other well-known statistics that are analogous to it in nature, including the clustering coefficient [51] and the network curvature [52]. In addition, it determines how many edges must be sampled for the major spectral feature to be retained.

Definition 1: Given a connected and undirected graph $G = (V, E)$. We define the parameter α of the graph G by equation (6) and $\epsilon \in (0,1)$. Using the preceding method, choose $O(\alpha n \log n)$ edges from G to create a weighted network H. Then H has a probability of at least 1-1/n of meeting the strong spectral characteristics equation (1).

The parameter α determines the typical quality of the local network. A specific situation where $d_i = d$ over all nodes i as well as $c_{ij} = c$ over all edges $(i, j) \in E$ may be used to illustrate the concept of α. Since the number of elements in set M is denoted by $|M|$, we get $\alpha \approx 2|E|/(nc) = d/c$. In this case, the number of neighbors shared by i and j is roughly d/α providing $(i, j) \in E$. That is, $1/\alpha$ will be the total number of shared neighbors between i and j.

Definition 1 demonstrates that if we sample locally and keep $O(n \log n)$ edges, we may nearly preserve the network structure when the local connections are strong $\alpha = O(1)$. In comparison, if the local connection is low, as in the case when $c_{ij} = O(1)$, then p_{ij} are of a similar order, and the sampling strategy is very close to uniform sampling. Most of the graphs we have access to are rather sparse, so the values are fairly tiny; this is consistent with the observation that real-world networks tend to have high levels of local connectivity. For the aforementioned sampling strategy to work, we need to know how many neighbors c_{ij} all incident node

pairs share. The computing cost of the sampling approach has linear structure to the overall edge quantity $|E|$ if c_{ij} has free access, as they are in certain social networks like Facebook. If c_{ij} are unavailable, we may either use a parallel calculation method or make an approximation using a neighbor sampling method. And thus, to sum up, we provide a simple approach to sampling that takes advantage of the fact that real-world networks often have a lot of local connections. We show that by sampling $O(\alpha n log n)$ edges, we can ensure that the strong spectral condition equation (1) is satisfied.

Here we explore the challenge of computing c_{ij} (exactly or roughly) when the graph is too big to fit on a single machine. Once all c_{ij} are calculated, the sampling approach is as simple as randomly selecting edges from the graph and averaging the results. An MPI-based distributed memory parallel technique is used in [53] to efficiently determine the number of shared neighbors c_{ij}. The method first dissects the graph into smaller, overlapping subgraphs, which are then stored independently on different workstations. The number of shared neighbors between each pair of nodes in a subgraph is then tallied using a sequential algorithm that identifies all triangles in the subgraph. Because a single edge in the original graph could be part of many overlapped subgraphs, the scores from each local computer are combined before the final result is shown. [53] show that their method can handle networks with billions of edges and that the number of local machines can grow almost linearly. Moreover, sampling, or in other terms, dropping edges, can alleviate oversmoothing, which is proved through theoretical results by [54].

C. LINE GRAPH TRANSFORMATION

Graph neural networks can be used to predict a link's existence by learning properties out of a provided bordering subgraph $G^h_{v_1,v_2}$, whereas $G^h_{v_1,v_2}$ refer to h-hop bounding subgraph centering upon 2 specific nodes v_1 and v_2. Every node in the enclosing subgraph is labeled to indicate the structural significance of the target connection. The number of nodes in a graph's surrounding subgraphs may vary. When we attempt to extract a feature vector of a specific size for use in subsequent prediction, we will experience some data loss. As a solution to this problem, we suggest creating a line graph out of the enclosing subgraph to show the connections among the edges of the initial graph. So, graph convolution neural networks can be used to predict a link's properties by looking at its line-graph representation.

It has been proposed that the adjacencies of edges in an undirected graph G may be represented by its corresponding line graph $L(G)$ [55], [56]. A line graph, denoted by the notation $L(G)$, in mathematics has the following concept:

1) In creating the line graph $L(G)$, the links of the initial graph G are used as vertices.
2) In $L(G)$, a connection between two vertices occurs only when their respective links both point to the same node.

Fig. 2 shows the steps involved in transforming a line graph. We start with an undirected graph G with eight vertices and eight edges. Because of this, there are eight vertices in the line graph $L(G)$. According to the concept of the line graph, the edges that correspond to nodes (AB) and (AD) in the actual graph G all pass through the same node A. The following feature of $L(G)$ is derived from the concept of a line graph:

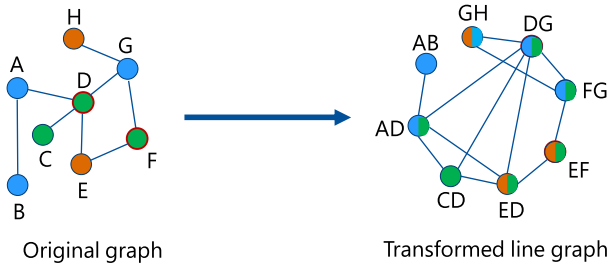

FIGURE 2. The method for transforming a line graph. Each line graph node is labeled with the names of the two nodes at each end, representing edges from the original graph.

Definition 2: If G is the graph having x number of vertices with y number of edges, then the amount of vertices in the corresponding line graph L(G) is y. L(G) has the following amount of edges:

$$e_{L(G)} = \frac{1}{2}\sum_{i=1}^{x} d_i^2 - y \tag{7}$$

where d_i is the i-th node's degree in graph G. This condition ensures that learning about line graph features does not significantly increase computational complexity. Furthermore, the time complexity of creating the line graph $L(G)$ from the initial graph G is linear [57], [58].

D. PROPOSED ALGORITHM

Here, we break down the proposed framework into its three constituent parts. The order of the three stages is flexible. The graph topology labeling and enclosing subgraph extraction functions presented in this section are suitable for the majority of networks.

1) EXTRACTION OF ENCLOSING SUBGRAPHS

The topology of a network focused on two nodes may reveal whether or not a connection between them exists. Increased use of topological data usually results in improved performance. On the downside, it will need more computational resources. In an effort to strike a good compromise between speed and computing cost, we use a k-hop enclosing subgraph to anticipate the presence of the connection between nodes v_1 and v_2, as shown below:

$$G^k_{v_1,v_2} = \{v | min(d(v, v_1), d(v, v_2)) \leq k\} \tag{8}$$

where, $d(v, v_1)$ is the geodesic distance or shortest route between v and v_1.

2) LABELLING OF NODES

All we know about a graph is its topology, which we infer from a subgraph's enclosure. This is because we need to use a labeling function to determine the significance of each node

in the graph before we can learn properties about the target connection. The following requirements must be met by the node-labeling function: The first task is to locate the two nodes of interest. Two, tell the intended nodes how crucial they are to the structure. An efficient node labeling function, as suggested by [24], is used in this study.

$$f_l(v) = 1 + min(d(v, v_i), d(v, v_j)) + (\frac{d_s}{2})\{(\frac{d_s}{2}) + (d_s\%2) - 1\} \quad (9)$$

where, $(d_s/2)$ and $(d_s\%2)$ refer to output of integer division, leftover from d_s split in half, and d_s=$d(v, v_i)$+$d(v, v_j)$. Moreover, $f_l(v_i) = 1$ and $f_l(v_j) = 1$ designate the target nodes v_i and v_j with label 1. If a node v meets the conditions $d(v, v_i) = \infty$ or $d(v, v_j) = \infty$, we labeled it as zero, because $f_l(v) = 0$. Label $f_l(.) \in \mathbb{R}$ is returned by the node labeling function. The label of a node is usually expressed as a one-hot encoded vector.

3) EDGE ATTRIBUTE GENERATION

In the process of creating a line graph, the original graph's edges are transformed into nodes. Only original graph nodes are given the label. Instead of copying the labels of individual nodes from the source graph, a transformation function must be used to map node labels to edge attributes. Therefore, in a line graph, the edge property may be used as the node feature without any further transformation. Based on the label of the node, we recommend the following way to build the edge attribute:

$$l_{(v_i,v_j)} = \|(min(f_l(v_i), f_l(v_j)), max(f_l(v_i), f_l(v_j)), avg(f_l(v_i), f_l(v_j))) \quad (10)$$

where v_i and v_j are the two nodes at the edge's endpoints, $f_l(.)$ is the function for node labeling, and $\|$ is the concatenation process for the three inputs. The attribute of the edges (v_i, v_j) and (v_j, v_i) should be same, since we only examine undirected graphs in this study for link prediction. Changing the end nodes while maintaining the same edge attribute given by equation (10) is easily provable. Furthermore, the function can reliably keep the node's structural relevance information.

Simple graphs with edge attributes may be transformed effectively using the approach provided in equation (10). Node properties are sometimes included with graphs. In citation networks, for instance, a synopsis of the article might be included as a node property in the graph. In the case of attributed graphs, the problem of link prediction is also heavily influenced by the node characteristics. This calls for a more generic treatment of attributed graphs in the method for transforming edge attributes. Based on the equation (10), we may combine the original node property and the node label to generate the edge attribute. While reversing two endpoint nodes position in an undirected graph, the edge property will not be consistent. One method to get around this restriction is by treating the actual node label and node property with a slightly new manner.

$$l_{(v_i,v_j)} = \|(min(f_l(v_i), f_l(v_j)), max(f_l(v_i), f_l(v_j)), avg(f_l(v_i), f_l(v_j)), x_{v_i} + x_{v_j}) \quad (11)$$

where x_{v_i} and x_{v_j} are the node v_i and v_j's initial attributes. To ensure that edge properties remain consistent even after swapping the end nodes, we suggest combining them using a summing operation on the node attributes. In a line graph, the newly created edge attribute $l_{(v_i,v_j)}$ may serve as the node attribute. So, graph neural networks are used to turn the job of predicting links into a problem of classifying nodes. The process of link prediction in our proposed framework is shown in algorithm 1.

Algorithm 1 Proposed LeL-GNN Model Algorithm

1: **Input:** Node (v_1) and (v_2), Graph G
2: **Output:** Link or Non Link
3: Enclosing K-hop subgraph $G^k_{(v_1,v_2)}$ extraction for target nodes (v_1) and (v_2) using equation (8)
4: Labeling the nodes using the function from equation (9) to $G^k_{(v_1,v_2)}$
5: Sampling $O(\alpha n \log n)$ edges from graph $G^k_{(v_1,v_2)}$ where α is a learnable parameter from equation (6)
6: **if** attributed graph **then**
7: Get the edge attribute $l_{(v_i,v_j)}$ by equation (11)
8: **else**
9: Get the edge attribute $l_{(v_i,v_j)}$ by equation (10)
10: **end if**
11: Transform the sampled subgraph $G^k_{(v_1,v_2)}$ to corresponding line graph $L(G^k_{(v_1,v_2)})$
12: Apply deep graph neural networks on line graph $L(G^k_{(v_1,v_2)})$ for feature extraction
13: Classify Link or Non Link

V. EXPERIMENTS

In this part, we will describe the evaluation measures, datasets, baseline procedures, and experimental design that we used in this study, which in turn will help others reproduce our results.

A. EVALUATION METRIC

A common approach to the link prediction issue treats it as a simple case of binary classification. One of the node pairs will have a positive label if there is a connection between the two nodes, and a negative label otherwise. As can be shown in Fig. 3, the confusion matrix [59] is an effective tool for assessing the quality of a binary classifier. The confusion matrix for link prediction in graph shows:

- True Positive (T_P): In a graph, true positive values are available links in the original graph and are predicted as available links by the model.

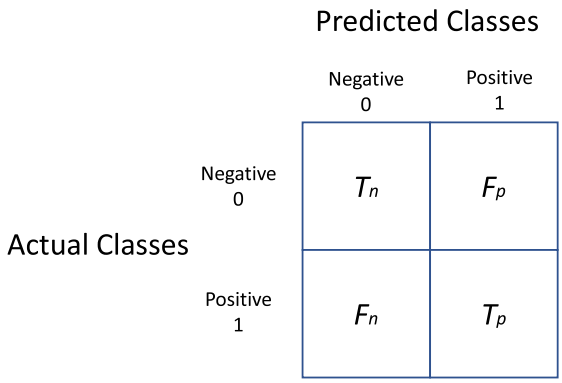

FIGURE 3. Confusion matrix for link prediction.

- True Negative (T_N): In a graph, true negative values are not available links in the original graph and are predicted as not available links by the model.
- False Positive (F_P): In a graph, false positive values are links that were not available in the original graph but that the model thought were available.
- False Negative (F_N): In a graph, false negative values are links that were available in the original graph but that the model thought would not be available.

Based on the aforementioned definitions, we propose the following two assessment criteria:

AUC (Area Under the Curve) used to evaluate the whole set of rankings, the AUC measures the likelihood that a missing link will be assigned a greater value than an absent connection. If, out of a total of n comparisons, n_1 of the missing connections have a greater value and n_2 have the equal value, then:

$$AUC = (n_1 + 0.5\, n_2)/n \tag{12}$$

It's important to keep in mind that an AUC close to 0.5 is expected if the scores come from a truly random and uniform distribution. Therefore, the performance of the algorithms relative to chance may be evaluated by seeing how much their AUCs surpass 0.5.

$$Precision = T_p/(T_p + F_p) \tag{13}$$

The precision measures how well the projected number of positive links between node pairs matches the actual quantity of positive links between node pairs. Better prediction efficiency is associated with higher precision values. The following equation describes how precision is determined.

B. DATASETS AND BASELINE METHODS

1) NON ATTRIBUTED DATASETS

We evaluate our proposed Learnable Edge Sampling and Line Graph based Graph Neural Network (LeL-GNNs) model on 10 non attributed datasets: BUP, HPD, CEG, YST, NSC, LDG, GRQ, UPG, UAL, and ADV [51], [60]. We gather 10 datasets from various domains to show that our suggested strategy is applicable in a wide variety of settings. Furthermore, the studies employ graphs of varying complexity in terms of the number of nodes and the number of connections. Table 1 displays information about the datasets. In this study, we examine two advanced heuristic techniques—Katz [39], and PageRank (PR) [40]—and compare them to our own suggested technique. Moreover, the SEAL [24] method is chosen as a benchmark alongside the graph auto-encoder (GAE) [48] as well as graph embedding technique node2vec (N2V) [61]. Furthermore, our LeL-GNN method is compared to two recent state-of-the-art benchmark LGLP [62] and mLink [50] methods for link prediction.

2) ATTRIBUTED DATASETS

As node-attribute graphs, we employ Cora [63], Cite-seer [64], and Pubmed [65]. Table 2 contains details about the datasets, such as their characteristics and statistics. We use LeL-GNN in conjunction with the variational graph autoencoder (VGAE) [48], the adversarially regularized graph variational autoencoder (ARGVA) [66], and the Graph Info Clust model (GIC) [67], all of which are unsupervised GNN models and compared with the recent state-of-the-art method WalkPool [68], to provide the best results for datasets including node attributes.

3) OPEN GRAPH BENCHMARK DATASETS

Our LeL-GNN is tested on the following Open Graph Benchmark (OGB) [69] datasets for link prediction accuracy: ogbl-ppa, ogbl-collab, ogbl-ddi, and ogbl-citation2. Table 3 summarizes important statistics from each dataset. Ranking the efficiency of positive sample edges against negative sample edges is used to evaluate link prediction in OGB datasets. In particular, the Hits@K metric is used to determine the percentage of positive sample edges that get ranked in the top K positions relative to the negative edges that were randomly selected. This metric is used in ogbl-ppa, ogbl-collab, and ogbl-citation2. The Mean Reciprocal Rank (MRR) is used as an assessment measure in ogbl-citation2; this metric averages the reciprocal ranks of the actual link and the negative candidates at each source node.

We compare our LeL-GNN against three heuristic approaches, three embedding-based techniques, as well as five GNN models to show how well they perform in the task of link prediction. We employed three popular heuristic approaches—Common Neighbors, Adamic Adar [37], as well as Resource Allocation [38]—that are based on overlapping neighborhoods. Without going through a learning process, they are able to forecast linkages by using each other's predefined structural data about overlapping regions. Matrix factorization, Node2Vec [61], as well as Multi-Layer Perceptron (MLP) were the embedding-based algorithms we employed. We also evaluate our approach against other GNN-based models, including GCN [2], GraphSAGE [12], JK-Net [9], GAT [34], as well as recent state-of-the-art method Labeling Trick [70] which is based on SEAL. Using a similarity score calculated between the target node and the source node associated with the target links, GCN, GraphSAGE,

TABLE 1. Non attributed graph datasets summary.

Datasets	Vertices	Edges	Average Degrees	Types
BUP	105	441	8.4	Blogging Site
HPD	8756	32331	7.38	Biological networks
CEG	297	2148	14.46	Biological networks
YST	2284	6646	5.82	Biological networks
NSC	1461	2742	3.75	Author Networks
LDG	8324	41532	9.98	Author Networks
GRQ	5241	14484	5.53	Author Networks
UPG	4941	6594	2.669	Power Networks
UAL	332	2126	12.81	Airport Traffic Networks
ADV	5155	39285	15.24	Social Media

TABLE 2. Attributed graph datasets summary.

Dataset	Vertices	Edges	Classes	Node Features
Cora	2708	5429	7	1433
Citeseer	3327	4732	6	3703
Pubmed	19717	44338	3	500

JK-Net, and GAT are able to forecast target connections. Target linkages may be predicted using representations of the surrounding subgraphs as graph categorization, which is extracted using SEAL.

C. EXPERIMENTS SETUP

In order to evaluate the efficacy of our suggested approach, we randomly pick 80% of all existing edges to serve as positive training sets and the other 20% to serve as positive test sets. On top of that, an equal number of false links are drawn at random from the graph to be used as negative samples during both the training and testing phases. Performance on datasets may be improved by adjusting the settings of baseline approaches. In the Katz approach, we use a damping factor of 0.001. PageRank's damping factor is at its default value of 0.85. For node2vec, we chose an embedding dimension of 128.

We replicate the original paper's [24] environment to test the SEAL architecture. In this work, for the SEAL architecture, we compute the K-hop containing subgraph, then we use the same function for node labeling as in equation (9). For calculating the node embeddings, we utilize sixteen graph convolution layers, and sort pooling [18] is employed to build a feature vector for the surrounding subgraph that is fixed in size. We use 64 channels of output feature map from our sixteen-layer graph convolutional neural network. In this configuration, the sort pooling layer's ratio is 0.6. To determine whether a connection exists, a classifier is used, which consists of two 1-dimensional convolution layers with 32 and 64 output channels and 2 fully connected layers. On each data set, the SEAL method is trained for 100 epochs.

Node embeddings in our method are calculated using a graph neural network, the same kind of network used in the SEAL model, so that comparisons between the two methods are fair. An important aspect of our approach is that it does not rely on graph pooling or 1-D convolution layers. As a result, our suggested technique uses a much lower number of parameters than the SEAL model does. For each dataset, we train our technique for 32 epochs.

In order to construct a preliminary node representation, we use an unsupervised strategy on the three datasets (cora, citeseer and pubmed) that include node features. This is due to the fact that traditional two-layer GCNs lack expressive power, making it difficult to extract meaningful node characteristics when there are node attributes present [71].

The size of hops between samples in the surrounding subgraphs is set to 2, with the exception of the UPG dataset, where it is set to 3. The LDG and ADV datasets, as well as the Pubmed and OGB benchmark datasets, yielded a large number of nodes from 2-hop subgraphs. Therefore, we constrain each dataset to a maximal number of nodes per step that can be stored in memory. In specific, if the number of nodes chosen surpasses 100 at any given step, we pick 100 nodes at random.

We used PyTorch to reimplement the neighborhood heuristic approach described in the cited works. We utilized the GAT, GraphSAGE, GCN, JK-Net, and Node2Vec implementations from PyTorch Geometric [72] alongside the SEAL implementation from its official github source. The common hyperparameters we used in our entire experiment are as follows: optimizer: Adam; learning rate: 0.001; and weight decay: 0. We run all of our experiments on a server equipped with an Nvidia GeForce RTX 3090 Ti (24GB) and a Quadro RTX (48GB) GPU card.

VI. RESULTS AND ANALYSIS

1) NON ATTRIBUTED DATASETS

We test 10 non attributed datasets using our suggested technique and the baseline approaches to see how well our approach compares to others. Each dataset is randomly divided into a training set and a test set ten times. Table 4 displays the average AUC comparison obtained using 80% training connections. Table 5 shows the results based on AP. Since the heuristic method is manually developed, it is

TABLE 3. OGB link prediction datasets' statistical and evaluation measures.

Dataset	Vertices	Edges	Average Degrees	Density	Split Ratio	Metric
ogbl-ppa	576,289	30,326,273	73.7	0.018%	70/20/10	Hits@100
ogbl-collab	235,868	1,285,465	8.2	0.0046%	92/4/4	Hits@50
ogbl-ddi	4,267	1,334,889	500.5	14.67%	80/10/10	Hits@20
ogbl-citation2	2,927,963	30,561,187	20.7	0.00036%	98/1/1	MRR

TABLE 4. AUC comparison with state-of-the-art methods. Results are average AUC with standard deviations for 10 runs with 80% training links.

Datasets	Katz	PR	N2V	GAE	SEAL	mLink	LGLP	LeL-GNN
BUP	87.10±2.73	90.13±2.45	80.25±5.55	90.16±1.65	93.32±0.84	93.54±0.63	95.24±0.53	**95.83±0.38**
HPD	85.47±0.35	87.19±0.34	79.61±1.14	85.21±0.45	92.26±0.09	92.64±0.08	92.58±0.08	**93.33±0.55**
CEG	84.84±2.05	89.14±1.35	80.08±1.52	83.73±0.75	87.44±1.21	89.08±0.86	90.16±0.76	**92.01±0.95**
YST	80.56±0.78	81.40±0.75	77.07±0.36	77.07±0.36	82.07±0.96	91.40±0.13	91.97±0.12	**92.47±0.35**
NSC	98.00±0.31	98.05±0.29	96.23±0.95	98.83±0.33	99.55±0.01	99.65±0.01	99.82±0.01	**99.88±0.01**
LDG	92.96±0.19	94.46±0.19	91.88±0.56	93.84±0.21	96.44±0.13	96.62±0.11	96.70±0.07	**97.19±0.13**
GRQ	89.81±0.59	89.98±0.57	91.33±0.53	91.15±0.45	97.10±0.12	97.56±0.10	**97.68±0.10**	97.63±0.12
UPG	59.59±1.51	59.88±1.51	70.37±1.15	69.84±0.96	81.37±0.93	**83.14±0.61**	82.17±0.57	82.57±0.21
UAL	92.01±0.88	93.74±1.01	85.40±0.96	91.80±0.86	95.21±0.77	96.43±0.33	97.44±0.32	**97.73±0.38**
ADV	92.13±0.21	92.78±0.18	77.70±0.83	90.55±0.23	95.07±0.13	95.21±0.10	95.40±0.10	**95.63±0.14**

TABLE 5. AP comparison with state-of-the-art methods. Results are AP with standard deviations for 10 runs with 80% training links.

Datasets	Katz	PR	N2V	GAE	SEAL	LGLP	LeL-GNN
BUP	85.94±3.46	89.53±3.11	81.47±4.48	89.26±2.10	93.58±0.68	**95.46±0.43**	94.34±0.29
HPD	89.52±0.32	91.01±0.23	80.57±0.81	86.62±0.39	93.41±0.09	**93.65±0.08**	93.55±0.06
CEG	85.94±3.46	87.96±1.69	77.98±1.54	82.53±1.51	86.49±1.08	89.70±0.53	**91.08±0.31**
YST	85.76±0.64	86.34±0.72	78.48±1.03	82.65±0.86	91.85±0.20	92.98±0.10	**93.73±0.23**
NSC	98.02±0.43	98.08±0.34	96.81±0.86	98.93±0.31	99.51±0.01	**99.82±0.01**	99.57±0.01
LDG	94.91±0.27	96.26±0.22	92.12±0.50	95.24±0.19	96.55±0.11	96.86±0.06	**97.81±0.21**
GRQ	93.08±0.29	93.18±0.34	93.92±0.31	93.78±0.33	97.86±0.11	98.14±0.10	**98.87±0.16**
UPG	74.29±0.83	74.74±0.81	76.55±0.75	75.04±0.87	83.91±0.83	84.78±0.53	**85.48±0.31**
UAL	93.51±0.79	94.30±1.27	82.53±1.12	93.41±0.67	95.46±0.59	97.37±0.25	**97.85±0.34**
ADV	93.72±0.16	94.03±0.24	79.02±0.65	90.87±0.26	95.18±0.12	95.72±0.08	**96.26±0.10**

evident from the findings that the heuristic approach cannot provide good performance on all datasets. Due to its ability to automatically infer the edge distribution from given datasets, the SEAL model is consistently shown to beat all heuristic approaches and embedding methods. The findings also demonstrate that directly learning the embedding of the target link is superior to the alternative of learning a graph embedding. Two recent state-of-the-art methods, mLink and LGLP, have shown great performance in the task of link prediction. These models and many other recent models also use the same method of SEAL architecture for subgraph extraction and node labeling. Considering the idea of converting the extracted subgraph into a corresponding line graph LGLP, our proposed method is similar, but we introduced learnable edge sampling before line graph transformation, which ensures major spectral features are retained.

Our suggested LeL-GNN model outperforms the state-of-the-art approaches on most of the datasets, including SEAL and LGLP, on both of the assessment measures used. As a result, we know that our suggested technique can learn superior characteristics to describe the corresponding link for prediction in the LeL-GNN setting. As an added bonus, our suggested technique is more reliable than the baselines.

In order to better demonstrate the efficacy of our suggested approach, we visualized features of edges utilizing t-distributed stochastic neighbor embeddings [73] and took the output of the final fully connected layer as a feature of each edge. Fig. 4 displays the actual class difference as the outcomes of the t-SNE. For 20% of the test edges, we demonstrate the rendering on the BUP, HPD, LDG, UAL, and ADV datasets. Green indicates a positive connection, while red indicates a negative one. The outcomes show that characteristics taught using our suggested model can be categorized with little effort. We compared the time complexity of our method with SEAL and LGLP to show the efficiency of our

algorithm. Table 6 is the representation of single epoch time on the same device.

2) ATTRIBUTED DATASETS

In addition, we undertake tests on attributed graphs. Through the use of LeL-GNN, we are able to extrapolate higher-order data from unsupervised learning's representations of nodes. We chose 10 random samples and present the average AUC and standard deviations for these in Table 7; for AP, please refer to Table 8. Table 7 and Table 8 displays the outcomes of unsupervised models in a LeL-GNN setting and is compared with the recent state-of-the-art WalkPool. The accuracy of predictions is enhanced by LeL-GNN for all unsupervised methods and datasets. The most notable improvements from using LeL-GNN can be seen when using the Pubmed dataset, in which topology is more important than features.

We achieve better performance than SEAL for attributed datasets because we use a unique method for integrating node functionalities. By incorporating a regularization term that moves the learned embeddings closer to the associated node attributes in feature space, our method modifies the goal function. When the node properties are useful and applicable to the link prediction job, this regularization term may assist in collecting more information from them and increasing performance.

One other factor is that our objective function was created to maximize different criteria than the SEAL's. While SEAL is concerned with maintaining the topological facts of links, our method aims to maintain both the network's topological structure and the local connections of nodes in feature space. As a result, our method and SEAL respond differently to the same node properties. In conclusion, discrepancies in how SEAL and our method include node features and improve their objective functions are likely to be blamed for why employing node features could result in a performance loss in SEAL but not in our method.

3) OPEN GRAPH BENCHMARK DATASETS

Using the Open Graph Benchmark (OGB) datasets, Table 9 displays the performance of various baselines and LeL-GNN in terms of link prediction. With the exception of ogbl-citation2, we employ GCN as an adaptation to combine with LeL-GNN on all datasets. Due to the high memory requirements of GCN during training, we bypassed it while training our LeL-GNN in ogbl-citation2. Table 9 demonstrates that, across the board, LeL-GNN perform at the state-of-the-art level. In particular, LeL-GNN outperforms the best baselines on ogbl-collab and ogbl-ddi. It is also important to note that LeL-GNN in ogbl-citation2 achieved state-of-the-art performance without the aid of GCN, i.e., by merely using graph topologies devoid of input node properties.

On both the ogbl-ppa and the ogbbl-collab datasets, traditional feature-based GNNs perform much worse than neighborhood overlap-based heuristic techniques. This means that feature-based GNNs struggle to make direct use of structural information, such as degree and overlapping neighbors, when making link predictions. This conclusion suggests that LeL-GNN and SEAL-based methods like the labeling trick are superior to traditional GNNs because of their ability to learn topological information. LeL-GNN and SEAL-based methods outperform heuristic approaches across all datasets because they are able to capture the structural information that they use. Although the labeling trick performs well when compared with heuristic approaches, it underperforms when compared to feature-based techniques in ogbl-ddi. It might be interpreted as the labeling trick being unable to make use of the input node characteristics and structure information in an adaptive manner. Instead, LeL-GNN adaptively integrates LeL-GNN and GCN, utilizing learnable edge sampling for each dataset, which demonstrates even greater performance than the efficiency of LeL-GNN and GCN individually.

Our suggested technique can learn the target link's attributes while still in the network. The only layers needed for feature extraction are graph convolution ones. To do this, the SEAL model employs graph convolution and pooling layers, since the method is carried out in the original graph. While the SEAL model has many parameters, our suggested technique has few and converges quickly. Using the loss of training and testing for each epoch, we are able to see how quickly our model converges across various datasets. Fig. 5 shows the outcome generated through TensorBoard [74]. Our proposed model shows smooth convergence, as shown by the findings. Our suggested technique just needs 30-35 epochs to reach its optimal performance. So, our proposed method reduces the number of model parameters and shortens the time it takes to train.

From the training and testing loss curves (Fig. 5), we can clearly see that there is no over-smoothing problem like in regular deep GCN models, as we used 16-layer models. So the over-smoothing problem caused by layer stacking is successfully handled in our model while maintaining state-of-the-art performance. We tried with a more deep architecture, like 32 or 64 layers, but unfortunately there is not much performance gain in terms of AUC and AP using our method. We believe that in the near future, researchers will explore more deep GNN with some excellent methods to get the expressive power of GNN like CNN.

Over-smoothing problems have been shown to be alleviated by the drop edge [54], but our method is different from the drop edge idea. In DropEdge, they drop out a certain rate of edges from the input graph at random. But in our method, we sampled edges based on a learning parameter that tells us how many edges we need to sample in order to keep the main spectral feature. In drop edge, they proved that removing edges can reduce the over-smoothing problem in a graph neural network model [54].

Over-smoothing is relevant to link prediction because it may reduce the discriminative capability of machine learning models' learned representations of network nodes, making it harder for these models to correctly forecast whether or not connections exist between pairs of nodes. It is

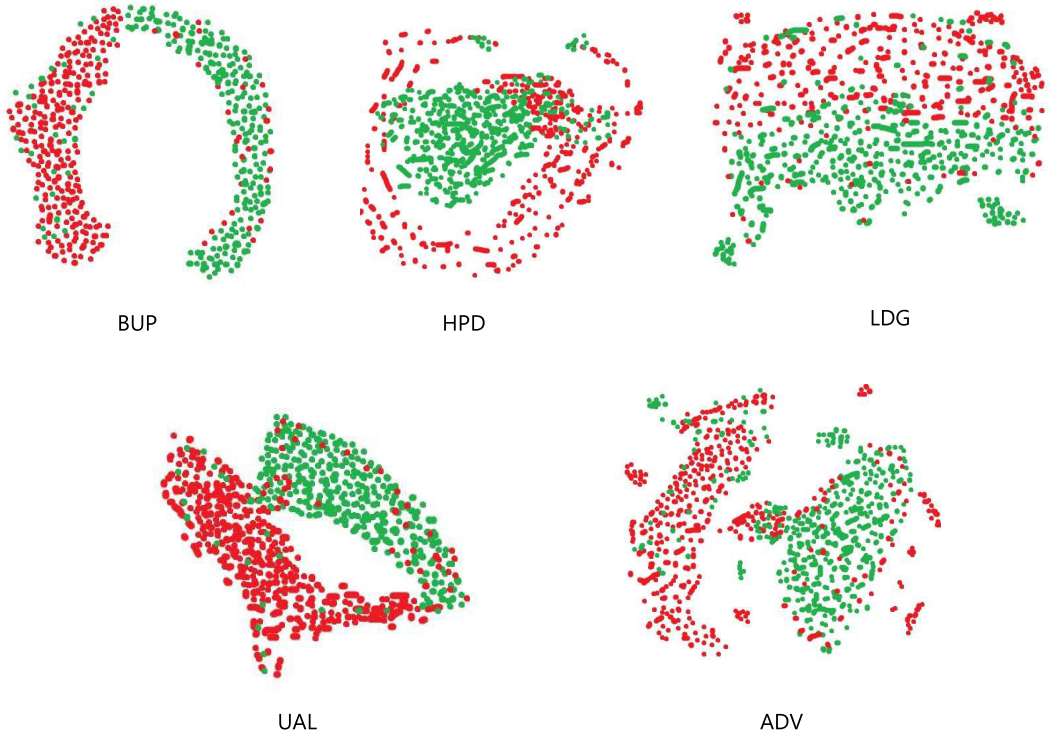

FIGURE 4. The BUP, HPD, LDG, UAL, and ADV datasets are visualized using t-SNE [73]. Green indicates a positive connection, while red indicates a negative one.

TABLE 6. Comparison of single training epoch time on the same device. Results are in seconds, with standard deviations for 10 runs.

Method	BUP	HPD	LDG	UAL	ADV
SEAL	1.81±0.26	2.75±0.35	2.80±0.52	2.91±0.88	3.55±0.36
LGLP	1.68±0.17	2.68±0.26	2.68±0.35	2.75±0.53	**3.18±0.25**
LeL-GNN	**1.56±0.12**	**2.55±0.32**	**2.60±0.28**	**2.53±0.72**	3.32±0.28

TABLE 7. AUC on attributed datasets using state-of-the-art architectures. Results are reported with standard deviations for 10 runs with 90% training links.

	VGAE		ARGVA		GIC	
	WP	LeL-GNN	WP	LeL-GNN	WP	LeL-GNN
Cora	94.64±0.55	95.70±0.62	94.71±0.85	95.83±0.70	95.90±0.50	**96.45±0.42**
Citeseer	94.32±0.90	94.58±0.75	94.53±1.77	94.73±0.68	**95.94±0.53**	95.61±0.55
Pubmed	98.49±0.13	98.30±0.22	98.52±0.14	98.45±0.52	98.72±0.10	**98.95±0.22**

TABLE 8. AP on attributed datasets using state-of-the-art architectures. Results are reported with standard deviations for 10 runs with 90% training links.

	VGAE		ARGVA		GIC	
	WP	LeL-GNN	WP	LeL-GNN	WP	LeL-GNN
Cora	95.11±0.53	95.50±0.60	95.23±0.84	95.74±0.76	95.97±0.57	**96.31±0.43**
Citeseer	94.89±0.89	95.22±0.75	95.04±1.46	95.63±0.20	96.04±0.63	**96.48±0.48**
Pubmed	98.46±0.14	98.50±0.10	98.49±0.14	98.67±0.12	98.65±0.15	**98.80±0.10**

standard practice in neural networks for link prediction to develop low-dimensional representations of network nodes that reflect their structural and relational features. A neural network model that predicts whether or not a given pair of nodes are connected uses these representations as input characteristics.

TABLE 9. Comparison of our LeL-GNN and baselines' link prediction (%) results on the Open Graph Benchmark (OGB) datasets. Each value is the mean efficiency over 10 independently initiated trials. Out of memory is abbreviated as OOM. The top results are highlighted here.

Method	ogbl-ppa	ogbl-collab	ogbl-ddi	ogbl-citation2
Common Neighbors	27.65±0.00	50.06±0.00	17.73±0.00	76.20±0.00
Adamic Adar	32.45±0.00	53.00±0.00	18.61±0.00	76.12±0.00
Resource Allocation	**49.33±0.00**	52.89±0.00	6.23±0.00	76.20±0.00
Matrix Factorization	27.83±2.02	38.74±0.30	17.92±3.57	53.08±4.19
Node2Vec	17.24±0.76	41.36±0.69	21.95±1.58	53.47±0.12
MLP	0.47±0.05	19.98±0.96	N/A	28.99±0.16
GCN	16.98±1.33	47.01±0.79	44.60±8.87	84.79±0.24
GraphSAGE	13.93±2.38	48.60±0.46	48.01±9.02	82.64±0.01
JK-Net	11.40±2.04	48.84±0.83	57.98±6.88	OOM
GAT	OOM	44.89±1.23	29.51±6.40	OOM
Labeling Trick	48.80±3.16	64.74±0.43	30.56±3.86	**87.67±0.32**
LeL-GNN	49.20±0.40	**65.52±0.50**	**60.37±3.47**	85.58±0.28

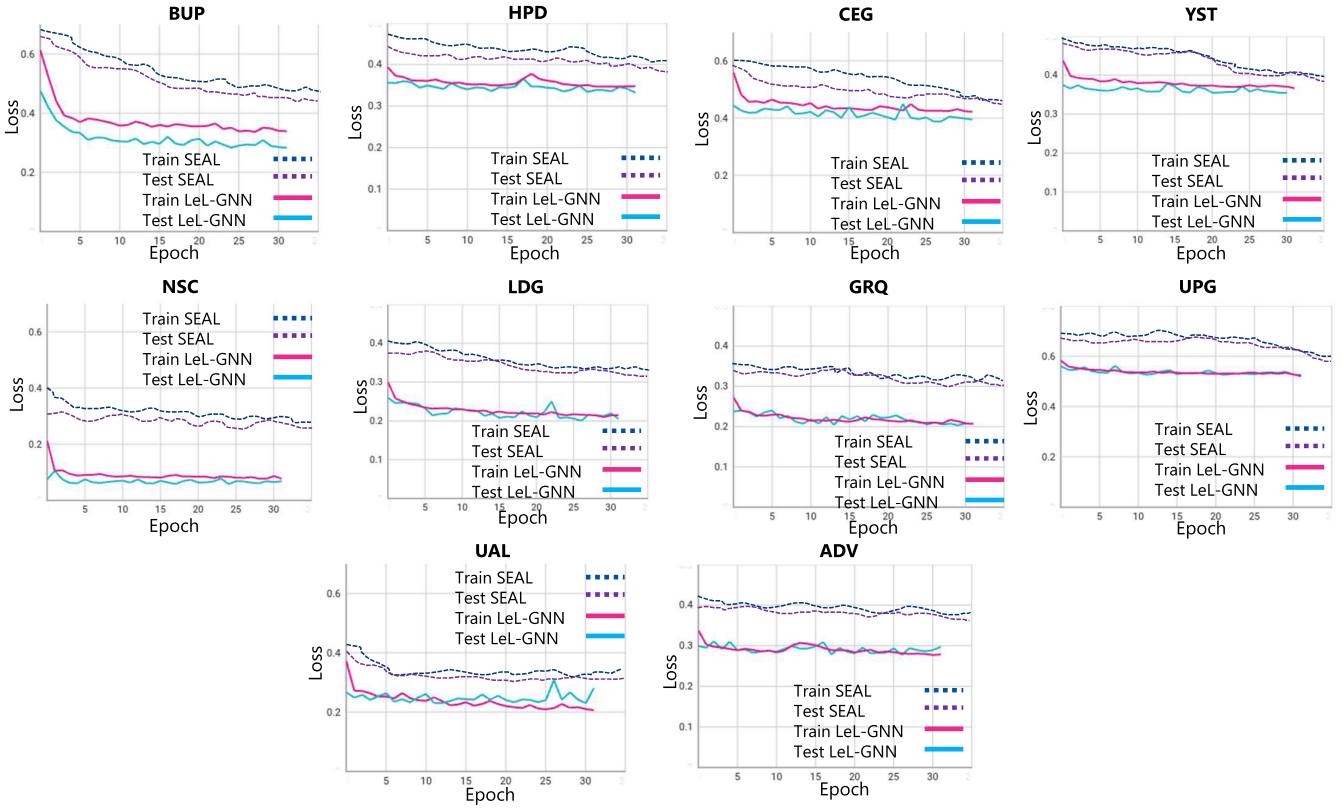

FIGURE 5. Training and testing loss comparison with SEAL.

When too much information is aggregated across nearby nodes during the learning process, a phenomenon known as "over-smoothing" occurs, in which the representations of nodes in a graph become too similar to one another. This may make it hard for a neural network model to effectively discriminate between nodes and forecast connections since critical information is lost and distinctions between them are diluted. By employing regularization methods like dropout or graph convolutional networks with connections skipped, which avoid excessive information aggregation and retain the discriminative capability of the learned node representations, we may reduce the likelihood of over-smoothing occurring.

VII. CONCLUSION

In this research, a new and adaptable link prediction model (LeL-GNN) based on learnable edge sampling and line graphs to aid in the creation of deep GNNs is proposed. Using simply the number of neighbors in common, we provide a

learnable edge sampling approach. This basic statistic offers a straightforward approach for measuring network local connection through the learnable parameter α, which affects the sampling method's efficiency directly. In order to prevent over-smoothing in graph convolution, our technique incorporates additional variety into the training dataset by arbitrarily sampling edges at a fixed rate, α. When predicting the presence of a connection across varying-sized networks, a feature vector of a predetermined size is generated using pooling layers. However, the pooling process might end up discarding useful data. In addition, training times for graph neural networks containing pooling layers are often longer. In order to get around these restrictions, we suggest generating a line graph from the original graph, where the target link feature can be obtained straight away from a line graph without resorting to a pooling operation. Results from 10 distinct datasets demonstrate that our suggested strategy beats most of the baseline methods and state-of-the-art models. Additionally, the LeL-GNN model converges substantially quicker compared to the other model. We anticipate that our study will pave the way for further investigation into deep GNNs and their wide range of possible applications.

REFERENCES

[1] H. Ren, W. Hu, and J. Leskovec, "Query2box: Reasoning over knowledge graphs in vector space using box embeddings," 2020, *arXiv:2002.05969*.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[3] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3097–3106.

[4] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–11.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, and C. Gulcehre, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[7] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Ser.*, vol. 2, no. 9, pp. 12–16, 1968.

[8] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018, *arXiv:1810.00826*.

[9] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.

[10] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–9.

[11] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," 2018, *arXiv:1810.05997*.

[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[13] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," 2018, *arXiv:1801.10247*.

[14] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*.

[15] A. Loukas, "What graph neural networks cannot learn: Depth vs width," 2019, *arXiv:1907.03199*.

[16] N. Dehmamy, A.-L. Barabási, and R. Yu, "Understanding the representation power of graph neural networks in learning graph topology," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.

[17] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[18] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–8.

[19] H. Yuan and S. Ji, "StructPool: Structured graph pooling via conditional random fields," in *Proc. 8th Int. Conf. Learn. Represent.*, 2020, pp. 1–12.

[20] H. Gao and S. Ji, "Graph U-Nets," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2083–2092.

[21] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–8.

[22] G. Li, M. Müller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?" in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9266–9275.

[23] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," 2019, *arXiv:1905.10947*.

[24] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.

[25] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.

[26] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1416–1424.

[27] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–12.

[28] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.

[29] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.

[30] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[31] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5425–5434.

[32] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," 2019, *arXiv:1907.04931*.

[33] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 257–266.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[35] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 338–348.

[36] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[37] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social Netw.*, vol. 25, no. 3, pp. 211–230, Jul. 2003.

[38] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *Eur. Phys. J. B*, vol. 71, no. 4, pp. 623–630, Oct. 2009.

[39] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, Mar. 1953.

[40] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Comput. Netw.*, vol. 56, no. 18, pp. 3825–3833, Dec. 2012.

[41] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2002, pp. 538–543.

[42] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 14, 2001, pp. 1–11.

[43] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009.
[44] E. M. Airoldi, D. Blei, S. Fienberg, and E. Xing, "Mixed membership stochastic blockmodels," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 21, 2008, pp. 1–11.
[45] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 701–710.
[46] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, May 2015, pp. 1067–1077.
[47] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, Feb. 2018, pp. 459–467.
[48] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*.
[49] M. Zhang and Y. Chen, "Weisfeiler-lehman neural machine for link prediction," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 575–583.
[50] L. Cai and S. Ji, "A multi-scale approach for graph link prediction," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, pp. 3308–3315, 2020.
[51] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, nos. 66–84, pp. 440–442, 1998.
[52] F. Bauer, J. Jost, and S. Liu, "Ollivier–Ricci curvature and the spectrum of the normalized graph Laplace operator," 2011, *arXiv:1105.3803*.
[53] S. Arifuzzaman, M. Khan, and M. Marathe, "Fast parallel algorithms for counting and listing triangles in big graphs," *ACM Trans. Knowl. Discovery Data*, vol. 14, no. 1, pp. 1–34, Feb. 2020.
[54] Y. Rong, W. Huang, T. Xu, and J. Huang, "DropEdge: Towards deep graph convolutional networks on node classification," 2019, *arXiv:1907.10903*.
[55] F. Harary and R. Z. Norman, "Some properties of line digraphs," *Rendiconti del Circolo Matematico di Palermo*, vol. 9, no. 2, pp. 161–168, May 1960.
[56] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," 2017, *arXiv:1705.08415*.
[57] N. D. Roussopoulos, "A max m, n algorithm for determining the graph H from its line graph G," *Inf. Process. Lett.*, vol. 2, no. 4, pp. 108–112, Oct. 1973.
[58] P. G. H. Lehot, "An optimal algorithm to detect a line graph and output its root graph," *J. ACM*, vol. 21, no. 4, pp. 569–575, Oct. 1974.
[59] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*, vol. 39. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[60] M. E. J. Newman, "The structure of scientific collaboration networks," *Proc. Nat. Acad. Sci. USA*, vol. 98, no. 2, pp. 404–409, Jan. 2001.
[61] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864.
[62] L. Cai, J. Li, J. Wang, and S. Ji, "Line graph neural networks for link prediction," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5103–5113, Sep. 2022.
[63] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Inf. Retr.*, vol. 3, no. 2, pp. 127–163, 2000.
[64] C. L. Giles, K. D. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Proc. 3rd ACM Conf. Digit. Libraries (DL)*, 1998, pp. 89–98.
[65] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu, "Query-driven active surveying for collective classification," in *Proc. 10th Int. Workshop Mining Learn. Graphs*, vol. 8, 2012, p. 1.
[66] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," 2018, *arXiv:1802.04407*.
[67] C. Mavromatis and G. Karypis, "Graph InfoClust: Leveraging cluster-level node information for unsupervised graph representation learning," 2020, *arXiv:2009.06946*.
[68] L. Pan, C. Shi, and I. Dokmanić, "Neural link prediction with walk pooling," 2021, *arXiv:2110.04375*.
[69] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 22118–22133.
[70] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Labeling trick: A theory of using graph neural networks for multi-node representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 9061–9073.
[71] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: A comprehensive review," *Comput. Social Netw.*, vol. 6, no. 1, pp. 1–23, Dec. 2019.
[72] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*.
[73] G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 15, 2002, pp. 1–8.
[74] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and S. Ghemawat, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

MD GOLAM MORSHED (Member, IEEE) received the bachelor's degree in computer science and engineering from the International University of Business Agriculture and Technology, Dhaka, Bangladesh, in 2011, and the M.S. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2017. He is currently pursuing the Ph.D. degree with the Data and Knowledge Engineering (DKE) Laboratory, Department of Computer Science and Engineering, Kyung Hee University, Global Campus, South Korea.

He is also a Lecturer (study leave) with the Department of Computer Science and Engineering, International University of Business Agriculture and Technology. His research interests include deep learning, computer vision, graph neural networks, data mining, big data analytics, and distributed learning. He was a recipient of the Best Paper Award in IEEE BigComp 2022.

TANGINA SULTANA received the B.Sc. degree from the Department of Telecommunication and Electronic Engineering, Hajee Mohammad Danesh Science and Technology University, Bangladesh, in 2010, the M.Sc. degree from the Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology (BUET), Bangladesh, in 2015, and the Ph.D. degree in computer science engineering from Kyung Hee University, Republic of Korea, in 2022.

She is currently an Assistant Professor (study leave) with the Department of Electronics and Communication Engineering, Hajee Mohammed Danesh Science and Technology University, Bangladesh. She is also a Postdoctoral Researcher with the Department of Computer Science and Engineering, Kyung Hee University. Her current research interests include graph mining, graph compression, wireless communication, edge computing, and machine learning. She was a recipient of the gold prize in KDBC 2020 and KDBC 2021 and Best Paper Award in IEEE BigComp 2021.

YOUNG-KOO LEE (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 1992, 1994, and 2002, respectively. Since 2004, he has been with the Department of Computer Science and Engineering, Kyung Hee University (Global Campus). His research interests include data mining, online analytical processing, and big data processing.