Improving Node Classification Accuracy of GNN through Input and Output Intervention

ANJAN CHOWDHURY, Indian Statistical Institute, India
SRIRAM SRINIVASAN, Bowie State University, USA
ANIMESH MUKHERJEE, IIT Kharagpur, India
SANJUKTA BHOWMICK, University of North Texas, USA
KUNTAL GHOSH, Indian Statistical Institute, India

Graph Neural Networks (GNNs) are a popular machine learning framework for solving various graph processing applications. This framework exploits both the graph topology and the feature vectors of the nodes. One of the important applications of GNN is in the semi-supervised node classification task. The accuracy of the node classification using GNN depends on (i) the number and (ii) the choice of the training nodes. In this paper, we demonstrate that increasing the training nodes by selecting nodes from the same class that are spread out across non-contiguous subgraphs, can significantly improve the accuracy. We accomplish this by presenting a novel input intervention technique that can be used in conjunction with different GNN classification methods to increase the non-contiguous training nodes and, thereby, improve the accuracy. We also present an output intervention technique to identify misclassified nodes and relabel them with their potentially correct labels. We demonstrate on real world networks that our proposed methods, both individually and collectively, significantly improve the accuracy in comparison to the baseline GNN algorithms. Both our methods are agnostic. Apart from the initial set of training nodes generated by the baseline GNN methods, our techniques do not need any other extra knowledge about the classes of the nodes. Thus, our methods are modular and can be used as pre-and post-processing steps with many of the currently available GNN methods to improve their accuracy.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Machine learning approaches**; **Neural networks**; *Modeling methodologies*;

Additional Key Words and Phrases: GNN, PaRWalk, DeepWalk, K-NN, K-Means

ACM Reference Format:
Anjan Chowdhury, Sriram Srinivasan, Animesh Mukherjee, Sanjukta Bhowmick, and Kuntal Ghosh. 2024. Improving Node Classification Accuracy of GNN through Input and Output Intervention. 1, 1 (June 2024), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Node classification, which involves identifying nodes of the same class in a complex network, is a fundamental problem in data mining. With the advent of information rich datasets, features associated with nodes also contribute to the

Authors' addresses: Anjan Chowdhury, anjan_r@isical.ac.in, Indian Statistical Institute, CSCR, BT Road, Kolkata, West Bengal, India, 700108; Sriram Srinivasan, Bowie State University, 14000 Jericho Park Rd, Bowie, Maryland, 20715, USA, ssrinivasan@bowiestate.edu; Animesh Mukherjee, IIT Kharagpur, Computer Science Department, Kharagpur, West Bengal, India, 721302, animeshm@cse.iitkgp.ac.in; Sanjukta Bhowmick, University of North Texas, Department of Computer Science and Engineering, 1155 Union Circle 311366, Denton, TX, 76203-5017, USA, sanjukta.bhowmick@unt.edu; Kuntal Ghosh, Indian Statistical Institute, MIU, BT Road, Kolkata, West Bengal, 700108, India, kuntal@isical.ac.in.

classification. Therefore, semi-supervised techniques such as graph neural networks (GNN [21, 25, 36, 44, 45, 84, 94, 99]), that can incorporate both structural as well as node feature information, have become popular tools for node classification.

Beginning with a training set of nodes whose true class (or label) is known, the features of the nodes are updated based on the features of their neighbors using a form of Laplacian smoothing. Nodes having similar features after smoothing are assigned to the same class.

After implementing any GNN model, the primary focus is to change the hyperparameters to achieve better accuracy. Two naive approaches for increasing the accuracy of a neural network model consists of (i) increasing the number of hidden layers and (ii) increasing the the number of training nodes.

Increasing the number of hidden layers. Sometimes, a neural network model with low representational capacity [32] may struggle to fit the training set. By increasing the number of layers, we can increase the representational capacity of the model. However, increasing the number of hidden units incurs increasing of time and memory cost. Moreover, in the case of some GNN models, increasing the number of hidden units often leads to the over-smoothing problem. As discussed in [47], a large number of hidden layers in the GCN [44] (a GNN variant) can result in nodes from different classes having almost indistinguishable features after the final smoothing operation.

Extending training nodes agnostically using random walks. Increasing the number of training nodes results in increasing the accuracy (see Fig. 1). Nevertheless, simply increasing the number of training nodes defeats the purpose of semi-supervised learning. Instead, a recent method [47] has focused on agnostically increasing the number of training nodes by applying random walks from small set of initial training nodes of known labels. This method extends the training set using (a) the label information of a few seed nodes and (b) the structure of the network. No additional label information is needed and the methods are analogous to an unsupervised set expansion technique [74, 86].

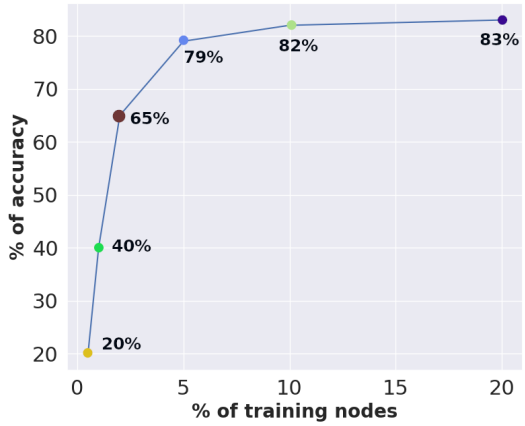


Fig. 1. Accuracy increases with percentage of training nodes used. This experiment was conducted on the Cora Network [71] using the GCN as implemented in Kipf *et. al.* [44].

Issues with extending training nodes. Methods for increasing the training set using random walks suffer from *two main issues* as follows.

First, these methods tend to produce training nodes that are localized in the neighborhood of the seed nodes. The initial set of training nodes are selected to represent each class. However, in cases, where the same class is distributed

across non-contiguous subgraphs, some of these subgraphs may not be represented in the training set. If the nodes in the extended training set come from the same subgraph, then other subgraphs will remain unrepresented and their nodes will be prone to misclassification. It is important to apply pre-processing to ensure that the training nodes are spread out across the network and not localized at certain regions. We illustrate this idea through a hypothetical example in Fig. 2.

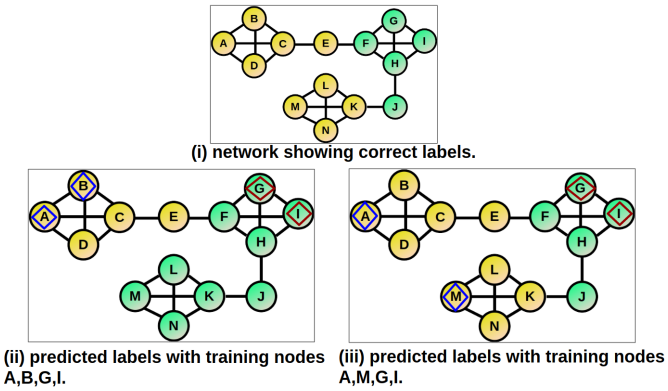


Fig. 2. Collection of training nodes of same class but non-contiguous sub-graphs improves accuracy. Top (i), the correct labels of the nodes. Bottom left (ii), training nodes are {A, B} and {I, G} from the two different classes. The subgraph {K,L,M,N} is erroneously subsumed into the green class. Bottom right (iii), training nodes are {A, M} and {G, I}. The yellow class has representative from two of its subgraphs, and correct classification is obtained.

Second, the nodes visited through a random walk from a set of seed nodes are all assigned the label of the respective seed nodes. However, in practice, several nodes can come from different classes. These misclassified nodes, when used as training nodes, can reduce the accuracy. In particular, we have observed that the output consists of localized regions of wrongly classified nodes. It is important to apply post processing to identify potentially misclassified nodes and correct their labels.

Our main contribution in this paper is to develop novel input and output intervention methods to address these two issues by (i) selecting a set of training nodes distributed along various non-contiguous subgraphs of same class and (ii) identifying misclassified nodes and correcting their labels.

Input intervention: For this, we leverage the structure of the graph using variations of random walks, as well as the feature vector associated with each node.

Output intervention: For this, we leverage the *confidence vector* associated with each node at the output to identify nodes with low confidence in classification, and then relabel the low confidence nodes with the labels of their nearest high confidence nodes.

To summarize, our main contributions are –

- Develop input intervention schemes based on a combination of random walk [90]/node embedding technique [63] and clustering/grouping techniques like K-means [30] and K-NN [28] to create a large set of training nodes collected from various non-contiguous sub-graphs of a graph using structural and feature vector information.
- Develop output intervention schemes to correctly relabel nodes predicted with *low confidence*, based on nearest neighbors labeled with *high confidence*.

Our results show that our input and output intervention techniques together can significantly improve the accuracy in comparison to baseline GNN methods, as well as GNN where the training set is extended using variations of random walk. Our performance gains over the most competing baseline on six benchmark networks of sizes varying between 2.7K–2.5M nodes range between **2%–141%**[1]. Finally, our intervention methods are modular and any advanced GNN method can be fitted seamlessly into the pipeline to obtain improved accuracy.

2 SOME PRELIMINARIES

2.1 Formal definitions

Consider a graph $G(V, E, F)$ where V is the set of nodes, E is the set of edges, $F = [f_1, f_2, ..., f_{|V|}]^T \in \mathbb{R}^{|V| \times m}$ is the feature matrix and f_v is the m dimensional feature vector corresponding to node v. We can write $V = V_l \cup V_u$ where V_l is the set of nodes of known class label and V_u is the set of nodes of unknown class label. Also let $L = \{l_1, l_2, ..., l_k\}$ be the set of k class labels, i.e., there are $|L| = k$ classes and class c has label l_c and each node belongs to a certain class. We also denote d_i as the degree of node i and $d_{avg} = \sum_{i=1}^{|V|} d_i/|V|$ as the average degree of the graph G.

2.2 Training set expansion

In [47], the authors showed that adding more hidden layers in GCN (a GNN variant) can lead to over-smoothing, which in turn makes the features indistinguishable and hence reduces classification accuracy. The authors further improved the accuracy of the GCN by first taking small sets of training nodes for each class and then extending these small sets using PaRWalk [90], a variant of random walk, starting with the training nodes in each set as seeds. In this section, we briefly discuss the workings of PaRWalk. In addition, we have observed that extending training nodes using a popular node embedding technique (DeepWalk [63]) in the graph also results in increasing the accuracy. Thus, in addition to PaRWalk, we also discuss the DeepWalk method.

PaRWalk. Although basic random walks can globally explore the graph, they may fail to capture non-local graph structure [53]. In order to tackle this Wu *et. al.* [90] suggest partially absorbing random walks (PaRWalk) - a variant of random walk. PaRWalk is a second-order Markov chain and in each state, it has partial absorption. If the absorption is properly set, then the absorption probabilities can capture the non-local graph structure. An absorption probability a_{ij} defines the probability of a random walk starting from node i, being absorbed in node j in any finite number of steps. All pair of absorption probabilities can be represented as a matrix $\mathcal{A}$.

DeepWalk. DeepWalk [63], is an unsupervised feature learning technique that leverages the latent information of nodes in a network. DeepWalk uses basic random walks to gather local information to learn latent representations. Similar to a language modeling tool, it requires a corpus and a vocabulary as its input, where the corpus is a set short of truncated simple random walks and the vocabulary is the set of nodes V. The algorithm iterates a few numbers of times ($\mathcal{W}_P$), and in each iteration, for every node v, it produces a short random walk W_v of a given length $\mathcal{W}_L$. This W_v is used to update the node representations. Finally it produces a matrix of node representations $\Phi \in \mathbb{R}^{|V| \times d}$, where $|V|$ denotes the number of nodes, $\mathbb{R}$ is the set of real numbers and d is the size of embedding.

3 PROPOSED FRAMEWORK

Our goal is starting from a very few nodes in V_l as seed nodes, and with the help of input level and output level interventions, to improve the accuracy of the prediction of the label of the nodes in V_u.

[1]We compute % gain as [(acc. for new method - acc. for baseline)/(acc. for baseline)]*100

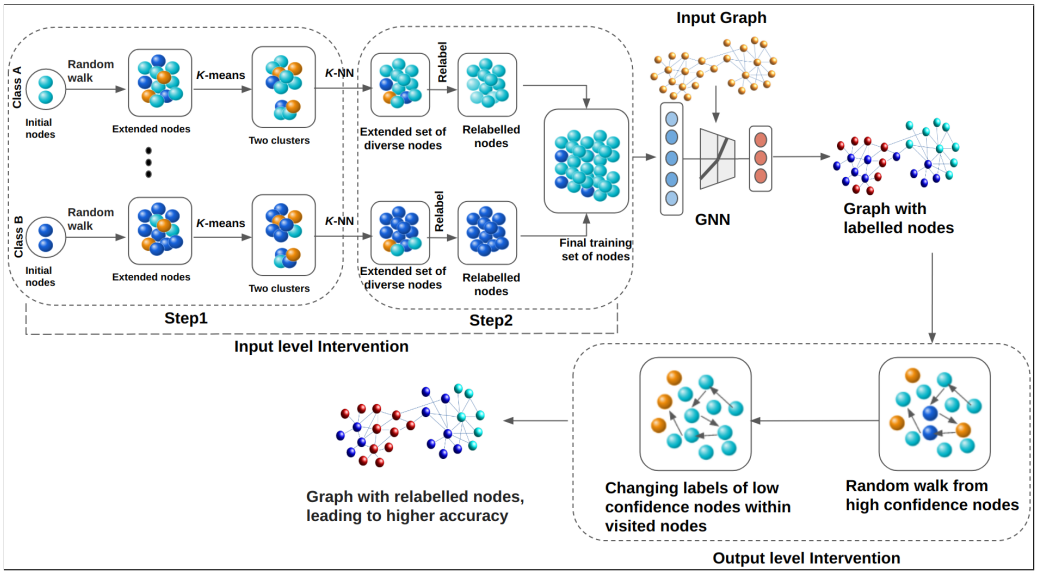


Fig. 3. Our proposed input and output interventions to GNN-based node classification.

We now present our main contributions, namely the **(i)** input level intervention to increase the spatial diversity of the training set and the **(ii)** output level intervention to identify and relabel the misclassified nodes. Figure 3 gives a pictorial representation of these steps.

3.1 Input level intervention

We consider the two methods ParWalk and DeepWalk, both of which extend the set of training nodes through different versions of random walk. However, merely extending training nodes can miss capturing the instances of the same class that are part of different non-contiguous sub-graphs in a graph. We leverage the well known clustering (K-means) and classification (K-NN) methods to track down the training nodes with same class label from different non-contiguous sub-graphs. Below we describe the steps of our algorithms (see Algorithm 1 for PaRWalk implementation and Algorithm 2 for DeepWalk implementation). We follow the same steps as in [47] for the selection of seed nodes. Our modification begins from Step 1 of the algorithm.

Seed nodes selection: The initial input is a set of seed nodes (I_c) selected from each class $c \in \{1, 2, ..., k\}$. We choose two seeds per class. In Figure 3 we show this example for two classes, class A and class B.

Step 1: In this step, first we extend the set of training nodes using either a variant of basic random walk (PaRWalk) or a node embedding method (DeepWalk). Then we aim to identify the nodes that contain the same label as the seed nodes. Lines $4-9$ in Algorithm 1, Lines $5-13$ in Algorithm 2 and the Step 1 in Figure 3 demonstrate this step.

In Algorithm 1, random walk using ParWalk is defined by the transition probability matrix $\mathcal{A}$. Given, a graph Laplacian Γ, the transition probability matrix $\mathcal{A}$ can be calculated as $\mathcal{A} = (\Gamma + \alpha\Lambda)^{-1}$, where $\Gamma = D - A$, D is the degree matrix $D = diag(d_1, d_2, ..., d_{|V|})$, d_i is the degree of node v_i and A is the adjacency matrix, $\alpha > 0$ is a scalar value, $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_{|V|})$ is known as the regularizer, and $\lambda_i \geq 0$ is some arbitrary value. Line 2 in Algorithm 1 shows the necessary step. More details about the absorption probability are in section A.2.

In Algorithm 2 graph embedding using DeepWalk can be achieved by, implementing the DeepWalk algorithm stated in [63]. We have written the DeepWalk algorithm using two simple steps - first, we build the model using the required parameters, as given in the code associated with [63], then we obtain the node embeddings by fitting the graph G inside the model. Lines $2-3$ in Algorithm 2 show the necessary steps.

For each seed node $i \in I_c$, using PaRWalk or DeepWalk, we agnostically capture a set X of at most t most similar nodes that are therefore likely to belong to the set I_c. The value of t is obtained as per [47] where, $t = \frac{|I_c| \times b \times \eta}{\sum_{c=1}^{k} |I_c|}$, b is a constant (we set b to 3), $\eta = |V|/(d_{avg})^{\tau}$, d_{avg} is the average degree of the graph G and τ is the number of layers of the GNN.

For ParWalk, corresponding to node i, node j_1 is more similar than node j_2 if $a_{ij_1} > a_{ij_2}$, where a_{ij_1} and a_{ij_2} are the absorption probabilities corresponding to node pairs (i, j_1) and (i, j_2) respectively. For the set I_c, to extract the t most similar nodes, first the column vector $\mathcal{A}_{:,i} = [a_{1i}, a_{2i}, ..., a_{|V|i}]^T$ is taken from the transition probability matrix of PaRWalk $\mathcal{A}$ for all nodes $i \in I_c$. Then we add them and make a final vector $\mathcal{A}_S = \sum_{i \in I_c} \mathcal{A}_{:,i}$ and select the t highest valued nodes.

For DeepWalk, for a node i, node j_1 is more similar than node j_2 if $CS(\Phi(i), \Phi(j_1)) > CS(\Phi(i), \Phi(j_2))$. $CS(f_1, f_2)$ is the Cosine Similarity [75] between vector f_1 and vector f_2, $\Phi(n) \in \mathbb{R}^d$ represents the embedded vector of a node n of size d (we set d to 100). For the set I_c, to capture at most t most similar nodes, $t/|I_c|$ closest nodes are extracted for all $i \in I_c$. Then the union of the newly extracted nodes are taken to produce the final set of at most t most similar nodes.

Given this extended set of training nodes X, we then perform an unsupervised classification using K-means algorithm to divide the set X into two clusters $C1$ and $C2$, as per the feature vector of the nodes. We assume that the larger cluster will contain more nodes of the same class as the seed while the smaller cluster will aggregate the nodes from other classes. This assumption is based on the intuition that most of the nodes at short distances from the seed will be of the same class, and therefore will be part of the larger cluster ($C1$). Indeed, as shown in the results, this heuristic significantly improves the classification results.

Step 2: In this step we replace the potentially correct locally clustered nodes (set $C1$) with spatially diverse ones. Line 10 in Algorithm 1, Line 14 in Algorithm 2, and Step 2 in Figure 3 illustrate this step.

We apply the K-Nearest Neighbors (K-NN) method to the feature vector space of the nodes and find $|C1|$ nodes that are nearest to the centroid of the nodes in set $C1$. These new nodes are selected from the set of unvisited nodes $V - X$. This new set of nodes form the set $C1'$, and is the set of extended nodes obtained from the initial seed from which the random walk was initiated.

Including set C2: In practice, we can apply only the $C1'$ set of training nodes. However, our goal is to compare the benefit of using spatially diverse nodes with those obtained through random walk methods. Therefore we combine the set $C1'$ with the set $C2$ (potentially mislabeled nodes), to create our final training set X' (Line 11 and Line 15 in Algorithm 1 and Algorithm 2 respectively). Note that the potentially mislabeled nodes are spatially local and the potentially correctly labeled ones are spatially diverse.

Relabeling the nodes X: As a final sub-step, we relabel the nodes in the extended set X to the label of the starting seed nodes as described in Lines $12-14$ in Algorithm 1 and Lines $16-18$ in Algorithm 2.

Building the final training set: Finally we merge the relabelled extended nodes obtained corresponding to all classes to make the final training set (Line 15 in Algorithm 1 and Line 19 in Algorithm 2).

Algorithm 1: Algorithm for input level intervention with PaRWalk

Input : The network $G(V, E)$, Graph Laplacian Γ, scalar value α, regularizer Λ, set size t, set of initial seed nodes' set $\mathcal{I} = \{I_1, I_2, .., I_k\}$

Output : Extended training set $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$;
$\mathcal{A} \leftarrow (\Gamma + \alpha\Lambda)^{-1}$;
for *each class* $c \in \{1, 2, ..., k\}$ **do**
  $\mathcal{A}_S \leftarrow \sum_{i \in I_c} \mathcal{A}_{:,i}$;
  $X \leftarrow$ select the t highest valued indices from $\mathcal{A}_S$;
  $Clus1, Clus2 \leftarrow K$-means$(X, clusterSize = 2)$;
  $C1 \leftarrow \max(Clus1, Clus2)$;
  $C2 \leftarrow \min(Clus1, Clus2)$;
  $C1_{cent} \leftarrow$ centroid of $C1$;
  $C1' \leftarrow$ using K-NN, select $|C1|$ nearest nodes to $C1_{cent}$ from $(V - X)$;
  $X' \leftarrow C1' \cup C2$;
  for *each node* $x \in X'$ **do**
    $Label[x] \leftarrow l_c$;
  end
  $\mathcal{S} \leftarrow \mathcal{S} \cup X'$;
end

3.2 Output level intervention

Our second contribution is to identify misclassified nodes and relabel them to their potentially correct labels in an agnostic manner. The steps are shown in Algorithm 3 (for PaRWalk) and Algorithm 4 (for DeepWalk).

The final layer in the GNN is usually passed through a softmax layer to obtain the vector of probabilities $CV_i = [p_{i1}, p_{i2}, ..., p_{ik}]$ of a node i belonging to each class, where, p_{ij} is the probability that node i has class label $l_j \in L$. We term this vector as *confidence vector* and define the *confidence* of a node i as: $Conf(i) = \max(CV_i)$. We then calculate the mean (μ) and standard deviation (σ) of the distribution of the confidence score of all the nodes ($Conf$). A node i is defined as a *high confidence* node if $Conf(i) \geq \mu + \alpha * \sigma$ and a *low confidence* node if $Conf(i) < \mu + \alpha * \sigma$.

We create a set of sets HCN of high confidence nodes for each class. Thus, $HCN = \{HCN_1, HCN_2, ..., HCN_k\}$, where HCN_c represents set of high confidence nodes of class c. Then, for each class c, we find the set of high confidence nodes (HCN_c) from HCN. Starting from HCN_c, i.e., the set of *high confidence* nodes, we perform PaRWalk or DeepWalk to fetch a set of nodes (RWN) of size atmost t. Here the size of t is set to the ratio of the number of nodes in G to the number of classes in G (Lines $3 - 5$ in Algorithm 3 and Lines $4 - 9$ in Algorithm 4).

For each node in RWN, if the node is a *low confidence* node, we change its label using the following rule. We create a set formed of the union of the nodes in (HCN_c) and the neighbors (Nbd) of the current node, we choose the label (l_c) of the node whose *confidence* is maximum, and relabel the *low confidence* node with that label (Lines $6 - 19$ in Algorithm 3 and Lines $10 - 23$ in Algorithm 4).

Note that our methods are completely agnostic. Other than the class of the initial seed nodes, we do not have any external knowledge of the class per node. We simply estimate the class based on random walks and clustering techniques such as K-means and K-NN.

Algorithm 2: Algorithm for input level intervention with DeepWalk

Input : The network $G(V, E)$, walks per node $\mathcal{W}_P$, walk length $\mathcal{W}_L$, set size t, set of initial seed nodes' set $\mathcal{I} = \{I_1, I_2, .., I_k\}$

Output : Extended training set $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$;
$Model \leftarrow DeepWalk(\mathcal{W}_P, \mathcal{W}_L)$;
$Emb \leftarrow$ Fit G in the Model to get the node embedding;
for *each class* $c \in \{1, 2, ..., k\}$ **do**
  $X \leftarrow \emptyset$;
  for *each node* $n \in I_c$ **do**
    $X_n \leftarrow$ get nearest $\frac{t}{|I_c|}$ nodes of n using Emb;
    $X \leftarrow X \cup X_n$;
  end
  $Clus1, Clus2 \leftarrow K$-means($X$, $clusterSize = 2$);
  $C1 \leftarrow \max(Clus1, Clus2)$;
  $C2 \leftarrow \min(Clus1, Clus2)$;
  $C1_{cent} \leftarrow$ centroid of $C1$;
  $C1' \leftarrow$ using K-NN, select $|C1|$ nearest nodes to $C1_{cent}$ from $(V - X)$;
  $X' \leftarrow C1' \cup C2$;
  for *each node* $x \in X'$ **do**
    $Label[x] \leftarrow l_c$;
  end
  $\mathcal{S} \leftarrow \mathcal{S} \cup X'$;
end

3.3 Rationale and Discussion

We now provide a rationale of why our intervention methods improves the classification. For both algorithms, the random walks aim to identify the local structures based on the assumption that nodes that are structurally local belong to the same class. However, the structural properties and the distribution of class in nodes need not always correspond.

Our input intervention aims to ameliorate the discrepancies between the structure of the graph and labeling of the nodes by first selecting the set of nodes that are most likely to be in the correct class and then extending this set via comparing similar feature vectors, rather than locally close nodes. Thus we combine both structural as well as feature properties. Further in our output intervention, we check if the nodes in the neighborhood of high-confidence nodes have low confidence and alter the label of its high-confidence neighbor(s). This output intervention allows us to further refine the classification.

3.3.1 Number of nodes in the expanded training set. We now discuss how large should we make the extended training set. We set the lower bound of the size of the extended training set as η;

$$\eta = |V|/(d_{avg})^{\tau}$$

where d_{avg} is the average degree of the graph G and τ is the number of layers of the GNN. We now estimate the number of training nodes required for a GNN with τ layers to propagate their labels to cover the entire graph. Thus, for a

Algorithm 3: Algorithm for output level intervention using PaRWalk

Input : The network $G(V, E)$, Predicted label of all nodes $PredLabel$, set size t, Confidence of each node $Conf$, Hyper parameter α, mean (μ) and standard deviation (σ) of the distribution of confidence of all the nodes, graph laplacian Γ, scalar value α, regularizer Λ,.

Output : Modified $PredLabel$.

$\mathcal{A} \leftarrow (\Gamma + \alpha\Lambda)^{-1}$;
$HCN \leftarrow$ select a set of high confidence nodes for every class using $Conf$, μ and $\alpha * \sigma$;
for *each class* $c \in \{1, 2, ..., k\}$ **do**
  $\mathcal{A}_S \leftarrow \sum_{i \in HCN_c} \mathcal{A}_{:,i}$;
  $RWN \leftarrow$ select the t highest valued indices from $\mathcal{A}_S$;
  /* change the label of the low confidence nodes */
  for *each node* $u \in RWN$ **do**
    $\mathcal{S} \leftarrow HCN_c \cup Nbd(u)$;
    $f \leftarrow$ select a node in $\mathcal{S}$;
    $maxConf \leftarrow Conf(f)$;
    if $Conf(u) \leq \mu$ - $\alpha * \sigma$ **then**
      for *each node* $v \in \mathcal{S}$ **do**
        if $Conf(v) > maxConf$ **then**
          $maxConf \leftarrow Conf(v)$;
          $l \leftarrow \arg\max_i(CV_v)$;
        end
      end
      $PredLabel[u] \leftarrow l$
    end
  end
end

particular class c, the size of the extended set would become,

$$t = \frac{|I_c| \times \eta}{\sum_{x=1}^{k} |I_x|}$$

where k is the number of classes and I_x is the set of initial seed nodes of a class x. Now, using the random walk (here, the transition probability matrix $\mathcal{A}$), one can obtain the t most similar nodes of a set of seed nodes I_c of a class c.

3.3.2 Error calculation on the expanded training set. Now we calculate the probability of error while extending the training set. Let the final set of extended training nodes be $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{S}_i$, where $\mathcal{S}_i$ be the set of extended nodes for class i. Note that the label of all the nodes in an extended set $\mathcal{S}_c$ is changed to l_c. Thus, the label of the nodes from other classes will also be relabelled to the class label l_c. This results in an error in the extended set.

Now assume that V_c be the set of nodes with class label l_c. Thus it is easy to verify that, $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \Phi$ for $i \neq j$ and $i, j \in \{1, 2, ..k\}$. Therefore the number of nodes having the correct class label l_c within set $\mathcal{S}_c$ will be

$$\mathcal{N}_c = |\mathcal{S}_c \cap V_c|$$

Thus, the number of unwanted nodes (nodes from another class) within $\mathcal{S}_c$ or the error associated with $\mathcal{S}_c$ will be:

$$\mathcal{E}_c = |\mathcal{S}_c| - \mathcal{N}_c = |\mathcal{S}_c| - |\mathcal{S}_c \cap V_c|$$

Algorithm 4: Algorithm for output level intervention using DeepWalk

Input : The network $G(V, E)$, Predicted label of all nodes $PredLabel$, set size t, Confidence of each node $Conf$, Hyper parameter α, mean (μ) and standard deviation (σ) of the distribution of confidence of all the nodes, walks per node $\mathcal{W}_P$, walk length $\mathcal{W}_L$

Output : Modified $PredLabel$.

$Model \leftarrow DeepWalk(\mathcal{W}_P, \mathcal{W}_L)$;
$Emb \leftarrow$ Fit G in the Model to get the node embedding;
$HCN \leftarrow$ select a set of high confidence nodes for every class using $Conf$, μ and $\alpha * \sigma$;
for *each class* $c \in \{1, 2, ..., k\}$ **do**
  $RWN \leftarrow \emptyset$;
  for *each node* $n \in HCN$ **do**
    $X \leftarrow$ get nearest $\frac{t}{|HCN_c|}$ nodes of n using Emb;
    $RWN \leftarrow RWN \cup X$;
  end
  /* change the label of the low confidence nodes */
  for *each node* $u \in RWN$ **do**
    $\mathcal{S} \leftarrow HCN_c \cup Nbd(u)$;
    $f \leftarrow$ select a node in $\mathcal{S}$;
    $maxConf \leftarrow Conf(f)$;
    if $Conf(u) \leq \mu$ - $\alpha * \sigma$ **then**
      for *each node* $v \in \mathcal{S}$ **do**
        if $Conf(v) > maxConf$ **then**
          $maxConf \leftarrow Conf(v)$;
          $l \leftarrow \arg\max_i(CV_v)$;
        end
      end
      $PredLabel[u] \leftarrow l$
    end
  end
end

Therefore, the probability of error associated with $\mathcal{S}_c$ will be:

$$\mathcal{P}_c = \frac{\mathcal{E}_c}{|\mathcal{S}_c|} = \frac{|\mathcal{S}_c| - |\mathcal{S}_c \cap V_c|}{|\mathcal{S}_c|} = \frac{t - |\mathcal{S}_c \cap V_c|}{t} = 1 - \frac{|\mathcal{S}_c \cap V_c|}{t} \quad \text{, as } |\mathcal{S}_c| = t, \forall c \in \{1, 2, ..., k\}$$

Also, the probability of error associated with $\mathcal{S}$ will be:

$$\mathcal{P} = \frac{\sum_{c=1}^{k} \mathcal{E}_c}{\sum_{c=1}^{k} |\mathcal{S}_c|} = \frac{\sum_{c=1}^{k} (|\mathcal{S}_c| - |\mathcal{S}_c \cap V_c|)}{\sum_{c=1}^{k} |\mathcal{S}_c|} = \frac{kt - \sum_{c=1}^{k} |\mathcal{S}_c \cap V_c|}{kt} = 1 - \frac{\sum_{c=1}^{k} |\mathcal{S}_c \cap V_c|}{kt} \quad \text{, as } |\mathcal{S}_c| = t, \forall c \in \{1, 2, ..., k\}$$

The value of the probability depends on how accurately the input level intervention technique collects the nodes having the same label as that of the initial seed nodes. As an example, if we assume that the classes are uniformly distributed and for each class, the input intervention method captures $\frac{3}{4}^{th}$ correct nodes (nodes having same class label as with c) then, $|\mathcal{S}_c \cap V_c| = \frac{3t}{4}$ implying $\mathcal{P}_c = 1 - \frac{3t}{4t} = 1 - \frac{3}{4} = \frac{1}{4}$. In our study, we have taken t as, $t = b\frac{|I_c| \times \eta}{\sum_{x=1}^{k} |I_x|}$, where b is an integer. Tuning b results in increasing or decreasing the set size t. When the value of t is large, then too many nodes from other classes come into $\mathcal{S}_c$ thereby increasing the error and thus reducing the test accuracy of the GNN.

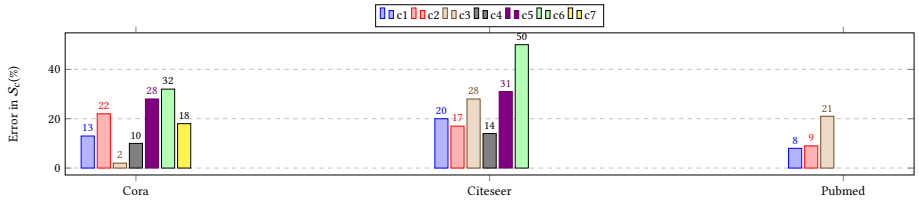
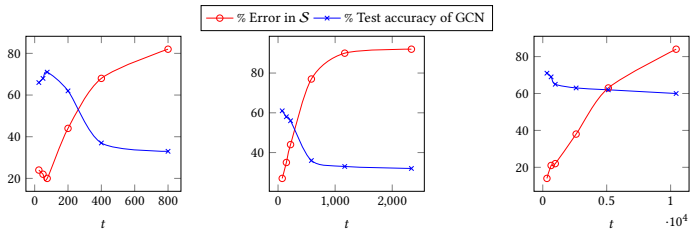


Fig. 4. Percentage of error for each class while extending the training set using input level intervention technique. In the legend, c1, c2, ... c7 are the different classes (b = 3)

.



(a) Cora (b) Citeseer (c) Pubmed

Fig. 5. Percentage of error in the extended set $\mathcal{S}$ and test accuracy of GCN when tuning the set size t

.

Conversely, when t is small, too few nodes from the actual class c come into $\mathcal{S}_c$, resulting in low test accuracy of the GNN. Experimental results capturing the probability of error for each class are provided in Figure 4. We also present the plot of error associated with the fully extended set $\mathcal{S}$ and the test accuracy of GCN while tuning the set size t in Figure 5. In our study, we set b to 3.

4 EXPERIMENTAL SETUP

4.1 Datasets

Our experiments are conducted on the datasets in Table 1. Cora [71], CiteSeer [71], PubMed [71] and Cora-ml [10] are citation networks. On each network, the documents are represented by nodes and the citation links are represented by edges. The node feature (a 0/1-valued vector) represents the absence/presence of a certain word in the corresponding document, and, classes represent subjects, labelled starting from 0 to (number of subjects - 1). Amazon Photo [72] and Amazon Computers [72] are subsets of the Amazon co-purchase network [57], where the goods are represented by nodes. If two goods are often purchased together then there is a link (edge) between them. The features of the nodes represent absence/presence of a certain word in the corresponding product reviews. In these two datasets above, the nodes (goods) are divided into several classes (product categories); for example, the goods in the Amazon photo can be classified as: 'Digital Concepts', 'File Photography', 'Flashes', 'Lenses', 'Video' 'Surveillance', 'Binoculars & Scopes', 'Tripods & Monopods', and 'Lighting & Studio'.

In the category of the larger networks, ogbn-arxiv [85] is a citation network among all computer science arXiv papers. A node represents an arXiv computer science paper and a link between a pair of nodes represents the citation. Each node is associated with a feature vector obtained by taking the embeddings of words in the corresponding paper's title

and abstract. Each node in this dataset is associated with a label representing the subject area such as cs.LG, cs.AI, cs.OS etc. There are 40 such subject areas available in the dataset. The ogbn-products [85] dataset is a graph that represents an Amazon product co-purchasing network and is undirected and unweighted. Products are represented by nodes on Amazon, and edges between two products show that they were bought together. Labels associated with each node represent the product categories. A portion of the Microsoft Academic Graph makes up the heterogeneous network named ogbn-mag [85] (MAG). Papers (736,389 nodes), authors (1,134,649 nodes), institutions (8,740 nodes), and fields of study (59,965 nodes) are among the four types of entities present in the graph. There are also four types of directed relations connecting the four types of entities: an author is "affiliated with" an institution, an author "writes" a paper, a paper "cites" a paper, and a paper "has a topic of" the field of study. Similar to ogbn-arxiv, each publication has a 128-dimensional word2vec feature vector connected with it, although input node features are not linked to any of the other types of entities. The labels associated with each data represent the venue.

Network	nodes	Edges	Features	Classes	Distribution of Nodes in Each Class
Citeseer [71]	3327	4732	3703	6	0: 7%, 1: 18%, 2: 20%, 3: 21%, 4: 18%, 5: 16%
Cora [71]	2708	5429	1433	7	0: 13%, 1: 8%, 2: 15%, 3: 30%, 4: 16%, 5: 11%, 6: 6%
Cora-ml [10]	2995	8416	2879	7	0: 12%, 1: 13%, 2: 15%, 3: 15%, 4: 29%, 5: 6%, 6: 9%
Pubmed [71]	19717	44338	500	3	0: 20%, 1: 40%, 2: 40%
Amazon Photo [72]	7487	119043	745	8	0: 5%, 1: 23%, 2: 9%, 3: 12%, 4: 12%, 5: 10%, 6: 26%, 7: 4%
Amazon Computers [72]	13381	245778	767	10	0: 3%, 1: 16%, 2: 11%, 3: 4%, 4: 39%, 5: 2%, 6: 4%, 7: 6%, 8: 16%, 9: 2%
ogbn-arxiv [85]	169,343	1,166,243	128	40	Can be found in Appendix A.1
ogbn-products [85]	2,449,029	61,859,140	100	47	Can be found in Appendix A.1
ogbn-mag [85]	1,939,743	21,111,007	128	349	Can be found in Appendix A.1

Table 1. The test suite of real-world networks.

4.2 Baseline methods

We use the following baseline GNN methods for comparison: (i) **GCN**, the popular GCN method as implemented in [44] which is a linear approximation of spectral graph convolution, (ii) **GAT** [84], where each neighbourhood of a node gets a different attention (weights) during the aggregation step. (iv) **ChebyNet** [25], ChebyNet is based on Chebyshev polynomials to implement the approximation of spectral convolution. It directly uses Laplacian as a filter. (v) **APPNP** [45], that combines the graph convolutional networks (GCN) and personalized PageRank to derive an improved message propagation technique. (vi) **GPR-GNN** [21] which is just like APPNP; however instead of personalized PageRank, GPR-GNN uses generalized PageRank. (vii) **JK-Nets** [94], In Jumping Knowledge Networks (JK-Nets), layer-wise jump connections and selective aggregations are implemented. (viii) **GraphSAGE** [36], It mainly works in an inductive setting, but can be tuned to work in a transductive setting also.

In addition, we also compared our intervention methods with the following seven methods for training set expansion [47, 96]. These techniques were used in conjunction with GCN – (a) **Co-training**, in which the training set is extended using PaRWalk [90]. The normalized absorption probability matrix A is computed first, then, the *confidence* of a node belonging to class c is measured. The top m nodes with the highest *confidence* measure are added to the training set with label l_c; (b) **Self-training** where after training **GCN** with initial training nodes, for each class, the top m nodes with with highest *confidence*, as determined by the softmax scores are added to the training nodes; and **combination of co-training and self-training** where the combination of the above two methods are tested by taking (c) the union (***Co-Self-union***) and (d) intersection (***Co-Self-intersection***) of the extended training sets returned by the *co-training* and *self-training* methods. (e) ***Training Node Augmentation (TNA)*** [96], in which intersection of nodes obtained from several already trained GNNs having confidence greater than a particular threshold are collected to expand the training

set. (f) ***Random sampling***, in which for each class, rather than using the random walk, the training sets are expanded using random sampling of nodes. (g) ***Hi-deg-path***, in which for each class, a path of high-degree nodes is picked (like meta-path in heterogeneous graph [49]). Starting from each of the training nodes, we search for its neighbor(s) having the highest degree, then we pick the highest degree neighbor, relabel it with the label of the starting node, put it into the set, and continue the process from this highest degree neighbor until the set of high-degree nodes stops extending.

4.3 Hyperparameter settings

For all the models we have kept the following hyperparameters fixed: In our input level intervention we have used two training nodes from each class. A random set of 1000 nodes (other than the training nodes) are taken for testing purposes for all the models. We are not using any validation set. The number of hidden layers used is two. We have set the learning rate to 0.01 and epochs to 200. We have used 16 hidden units and $L2$ regularization whose weight has been set to 5×10^{-4}. For the smaller graphs (Cora, Citeseer, Pubmed, Amazon Computers, Amazon Photo, and Cora-ml), we set walks per node $\mathcal{W}_P$ to 30 and walk length $\mathcal{W}_L$ to 50. For the larger graphs (ogbn-arxiv, ogbn-product and ogbn-mag), we set walks per node $\mathcal{W}_P$ to 10 and walk length $\mathcal{W}_L$ to 50.

4.4 Time complexity analysis

In this section, we shall provide the time complexity of our algorithms. In both the input and output intervention methods, we have used random walks. So, before calculating the exact time complexity of both algorithms, we calculate the time complexity of the random walk first.

Time requirement using PaRWalk: PaRWalk algorithm requires the computation of absorption probability matrix $\mathcal{A} = (\Gamma + \alpha\Lambda)^{-1}$, where, Γ is the graph laplacian defined as $\Gamma = D - A$, D is the degree matrix $D = diag(d_1, d_2, ..., d_{|V|})$, d_i is the degree of node v_i and A is the adjacency matrix, $\alpha > 0$ is a scalar value, $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_{|V|})$ is the known regularizer and $\lambda_i \geq 0$ is some arbitrary value. The computation of Γ is linear. Computation of $(\Gamma + \alpha\Lambda)$ is also linear. $\mathcal{A}$ can be computed using Williams algorithm [2] in $O(|V|^{2.373})$ time.

Time requirement using DeepWalk: In the case of DeepWalk, as per the algorithm given in [63], the time complexity can be calculated as $O(\mathcal{W}_P|V|(\mathcal{W}_L + log(|V|)))$, where, $\mathcal{W}_P$ is the walk per node, $\mathcal{W}_L$ is the walk length, $|V|$ is the number of nodes. The term $log(|V|)$ is due to the time complexity of SkipGram [63] method.

4.4.1 Time complexity of input level intervention. It is evident from Algorithm 1 and 2 that the major computations are contributed by (i) the random walk, (ii) K-means and (iii) K-NN algorithms. Therefore, we shall discuss the time complexity one by one and then aggregate the total time complexity.

(i) Time requirement using random walk has been discussed in the previous section.

(ii) Time requirement using K-means [58]: For t samples of dimension m each and δ clusters the time taken by K-means algorithm is $O(t\delta mj)$, where j is the number of iterations until convergence of the K-means algorithm.

(iii) Time requirement using K-NN: For a cluster of average size $|C_{avg}|$, K-NN [28] requires $O(|C_{avg}|m)$, where m is the dimension of each sample in the cluster C_{avg}.

Time requirement of Algorithm 1: Upon aggregating these above three, total time taken is the time taken by (i) + for each class, time taken using (ii) and (iii). Therefore, for $|L|$ number of class labels, total time taken

$$T_{ip\_par} = O(|V|^{2.373}) + |L|(O(t\delta mj) + O(|C_{avg}|m)).$$

The above expression can be further written as:

$$T_{ip\_par} = O(|V|^{2.373}) + (O(\frac{|V|mj}{(d_{avg})^{\tau}}) + O(|L||C_{avg}|m))$$

Details of the derivation can be found in Appendix A.3.

Time requirement of Algorithm 2: In the case of DeepWalk, applying a similar approach, the total time complexity for the input level intervention is:

$$T_{ip\_deep} = O(\mathcal{W}_P|V|(\mathcal{W}_L + log(|V|))) + (O(\frac{|V|mj}{(d_{avg})^{\tau}}) + O(|L||C_{avg}|m)).$$

4.4.2 Time complexity of Output level intervention: From Algorithm 3 and 4, the time complexity can be calculated as: time taken by (i) the random walk, and (ii) rest of the operations.

(i) Time requirement using random walk has already been discussed.

(ii) Time requirement to relabel the selected low confidence nodes, as computed in the respective algorithms:

Time requirement of Algorithm 3: Line 2 requires O($|V|$) time. For each class, Line 4 and 5 both need O(t) time + Lines 6 − 19 requires O($t(|H_{avg}| + d_{avg})$) time (considering average degree as d_{avg} and the average number of high confidence nodes as $|H_{avg}|$). So, the total time required is (using PaRWalk):

$$T_{op\_par} = O(|V|^{2.373}) + O(|V| + |L|(t + t(|H_{avg}| + d_{avg})))$$

This can be further written as:

$$T_{op\_par} = O(|V|^{2.373}) + O(|V| + |V|(1 + |H_{avg}| + d_{avg}))$$

The detailed derivation can be found in Appendix A.4

Time requirement of Algorithm 4: Similarly, the total time required is (using DeepWalk):

$$T_{op\_deep} = O(\mathcal{W}_P|V|(\mathcal{W}_L + log(|V|))) + O(|V| + |V|(1 + |H_{avg}| + d_{avg}))$$

Time requirement of both the input and output intervention: When we use both the input and output intervention techniques, the random walk is computed only once, which is then used in both interventions. Therefore, the computation time in the case of PaRWalk for both interventions can be written as:

$$T_{ip\_op\_par} = O(|V|^{2.373}) + (O\left(\frac{|V|mj}{(d_{avg})^{\tau}}\right) + O(|L||C_{avg}|m)) + O(|V| + |V|(1 + |H_{avg}| + d_{avg}))$$

and in the case of DeepWalk:

$$T_{ip\_op\_deep} = O(\mathcal{W}_P|V|(\mathcal{W}_L + log(|V|))) + \left(O(\frac{|V|mj}{(d_{avg})^{\tau}}\right) + O(|L||C_{avg}|m)) + O(|V| + |V|(1 + |H_{avg}| + d_{avg}))$$

Approximation for large graphs: In the case of PaRWalk, for the larger graph (ogbn-arxiv, ogbn-mag, and ogbn-products), computation of the absorption probability matrix is infeasible. Therefore for the larger graphs, we approximate (linear) the absorption probability matrix calculation. In our study, we set all $\lambda_q = 1$, for $q \in \{1, 2, ..|V|\}$, or in other words, we set $\Lambda = I$ = Identity matrix. Thus the absorption probability becomes $\mathcal{A} = (L + \alpha I)^{-1}$. Applying the concept of binomial theorem: $\mathcal{A} = (L + \alpha I)^{-1} = \alpha^{-1}(\alpha^{-1}L + I)^{-1} = \alpha^{-1}(I + \alpha^{-1}L)^{-1} = \alpha^{-1}(I - \alpha^{-1}L + (\alpha^{-1}L)^2 - ...)$ inspired by the following formula, $(1 + x)^{-1} = 1 - x + x^2 + ...$ We further approximate the absorption probability matrix $\mathcal{A}$ as $\alpha^{-1}(I - \alpha^{-1}L)$ by removing the higher order terms. Thus, with this linear approximation, the time required for

obtaining the absorption probability becomes $O(|V|)$. In the case of DeepWalk, we have reduced the hyper-parameters (number of walks per node and walk length) as discussed in section 4.3 to reduce the time.

5 EMPIRICAL RESULTS

As stated earlier, our main goals are to increase the accuracy of GNNs by (a) applying input level intervention and (b) applying output level intervention. In the input level intervention, we use two methods for training set expansion – ParWalk and DeepWalk. These two methods are also used in the output level intervention for capturing the nodes that are most similar to the high confidence nodes for each class. For this reason, we grouped our results into two categories: one corresponding to ParWalk and the other corresponding to DeepWalk. For each group, we have shown results for applying (i) only input intervention (ip + GNN variant), (ii) only output intervention (GNN variant + op), and (iii) both input and output interventions (ip + GNN variant + op).

5.1 Results using PaRWalk

Table 2 shows the results of all the three combinations of interventions along with the baselines using PaRWalk. We see that with only input level intervention, the accuracy increased in every case compared to the baselines (highest gain = (68-31)/31% = 119%, for cora network and ChebyNet model) For only output intervention, the accuracy increased a little compared to the baselines (highest gain = (44-37)/37% = 19%, for cora network and SAGE model). When we plug-in both the input and output interventions, as expected, the combination gives maximum gains compared over the baselines (highest gain = (70-31)/31% = 126%, for cora network and ChebyNet model). For each intervention, including baselines, we have boldfaced the maximum accuracies (mean and standard deviation) in the table.

5.2 Results using DeepWalk

Table 3 presents the results of only input intervention, only output intervention, and both input and output interventions, respectively, using DeepWalk as the training node expansion method. As with the previous results (PaRWalk), we can see that plugging the intervention methods improves the accuracies compared to the baseline method. For input intervention, highest gain = (67-31)/31% = 116% (ChebyNet model and cora network), for output intervention, highest gain = (29-23)/23% = 26% (JKNet model and cora-ml network).

5.3 Comparison with training set expansion baselines

In Fig. 6, we compare the proposed input intervention method with other baselines that uses training set expansion, viz., (i) co-training, (ii) self-training, (iii) co-self-union, (iv) co-self-intersection, (v) Training Node Augmentation (TNA), (vi) Random Sampling and (vii) High-deg-path. For each network, the leftmost bar represents the accuracy for GCN (base model). Next seven bars represents seven baselines (i)–(vii) and the last two bars represent the input intervention methods with ParWalk and DeepWalk respectively. It is evident from Fig. 6 that in almost all cases (except Amazon Photo) our methods outperform the other training set expansion baselines. This result is representative and holds true for all other GCN variants noted in Tables 2 and 3.

5.4 Significance test on output level intervention

In order to verify if the results of the output level interventions are statistically significant we report in Tables 4 and 5 the p-values for the Mann-Whitney significance test [55]. For all cases $p < 0.05$ which indicates that the output intervention indeed produces results that are significantly different from the results produced by the base GNN variant.

	GNN model	Networks								
		Cora	Citeseer	Pubmed	Photo	Computer	Cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
Baselines	GCN	0.45±0.05	0.33±0.06	0.52±0.05	0.32±0.09	0.21±0.09	0.36±0.04	0.12±0.04	0.15±0.03	0.02±0.04
	GAT	0.44±0.04	0.35±0.04	0.54±0.05	0.47±0.10	0.41±0.10	0.41±0.08	0.09±0.02	0.17±0.02	0.02±0.03
	ChebyNet	0.31±0.05	0.26±0.03	0.47±0.04	0.35±0.11	0.27±0.09	0.24±0.09	0.09±0.02	0.11±0.05	0.03±0.06
	GPR-GNN	0.48±0.06	0.38±0.05	0.62±0.08	0.54±0.13	0.44±0.17	0.52±0.09	0.14±0.02	0.21±0.02	0.05±0.03
	APPNP	0.50±0.05	0.36±0.04	0.59±0.04	0.23±0.11	0.10±0.10	0.36±0.10	0.14±0.03	0.20±0.04	0.05±0.03
	JKNet	0.35±0.04	0.27±0.05	0.57±0.04	0.17±0.10	0.32±0.10	0.23±0.11	0.08±0.03	0.12±0.02	0.02±0.03
	SAGE	0.37±0.05	0.30±0.05	0.56±0.05	0.42±0.11	0.32±0.10	0.31±0.11	0.04±0.03	0.11±0.06	0.02±0.07
Input intervention	ip+GCN	0.71±0.03	0.56±0.05	0.65±0.00	0.39±0.01	0.62±0.02	0.64±0.03	0.22±0.03	0.27±0.04	0.09±0.04
	ip+GAT	0.63±0.02	0.57±0.03	0.75±0.01	0.52±0.02	0.59±0.02	0.64±0.03	0.19±0.03	0.28±0.03	0.09±0.04
	ip+ChebyNet	0.68±0.01	0.62±0.03	0.85±0.00	0.50±0.03	0.51±0.01	0.60±0.03	0.20±0.02	0.20±0.07	0.11±0.05
	ip+GPR-GNN	0.68±0.01	0.60±0.03	0.83±0.01	0.57±0.03	0.50±0.08	0.67±0.03	0.24±0.01	0.24±0.07	0.08±0.04
	ip+APPNP	0.69±0.01	0.56±0.03	0.75±0.00	0.34±0.03	0.54±0.04	0.66±0.04	0.23±0.03	0.23±0.07	0.08±0.08
	ip+JKNet	0.52±0.01	0.43±0.03	0.75±0.01	0.27±0.01	0.48±0.03	0.45±0.02	0.19±0.07	0.18±0.08	0.04±0.07
	ip+SAGE	0.63±0.04	0.60±0.03	0.78±0.01	0.55±0.02	0.46±0.03	0.57±0.01	0.12±0.05	0.14±0.07	0.06±0.04
Output intervention	GCN+op	0.50±0.05	0.36±0.05	0.54±0.05	0.33±0.09	0.25±0.08	0.39±0.04	0.15±0.03	0.16±0.03	0.02±0.03
	GAT+op	0.46±0.04	0.38±0.05	0.53±0.05	0.48±0.07	0.49±0.08	0.45±0.07	0.12±0.04	0.17±0.04	0.03±0.03
	ChebyNet+op	0.35±0.04	0.27±0.03	0.48±0.04	0.36±0.10	0.28±0.09	0.27±0.08	0.11±0.04	0.12±0.04	0.03±0.03
	GPR-GNN+op	0.50±0.05	0.39±0.04	0.62±0.05	0.55±0.09	0.47±0.11	0.54±0.11	0.15±0.05	0.22±0.03	0.05±0.03
	APPNP+op	0.51±0.15	0.39±0.15	0.60±0.05	0.24±0.04	0.15±0.04	0.39±0.10	0.14±0.04	0.20±0.03	0.06±0.07
	JKNet+op	0.37±0.10	0.29±0.09	0.59±0.04	0.18±0.04	0.32±0.04	0.29±0.09	0.11±0.04	0.13±0.03	0.02±0.03
	SAGE+op	0.44±0.05	0.32±0.05	0.58±0.05	0.44±0.10	0.33±0.09	0.35±0.08	0.06±0.03	0.12±0.04	0.03±0.03
Both intervention	ip+GCN+op	0.74±0.01	0.63±0.02	0.67±0.01	0.40±0.04	0.64±0.05	0.68±0.02	0.24±0.04	0.28±0.03	0.09±0.04
	ip+GAT+op	0.65±0.01	0.63±0.03	0.77±0.02	0.54±0.06	0.63±0.08	0.67±0.01	0.21±0.05	0.28±0.03	0.09±0.04
	ip+ChebyNet+op	0.70±0.02	0.65±0.03	0.86±0.01	0.51±0.01	0.52±0.08	0.63±0.01	0.21±0.04	0.21±0.03	0.12±0.04
	ip+GPR-GNN+op	0.69±0.02	0.63±0.01	0.83±0.01	0.58±0.07	0.51±0.09	0.70±0.01	0.27±0.03	0.24±0.03	0.09±0.04
	ip+APPNP+op	0.71±0.01	0.59±0.01	0.78±0.01	0.35±0.06	0.53±0.08	0.69±0.01	0.24±0.04	0.24±0.03	0.09±0.04
	ip+JKNet+op	0.54±0.02	0.44±0.01	0.76±0.01	0.27±0.06	0.57±0.09	0.47±0.01	0.21±0.04	0.19±0.03	0.04±0.04
	ip+SAGE+op	0.65±0.01	0.65±0.01	0.78±0.01	0.43±0.06	0.56±0.10	0.60±0.01	0.13±0.04	0.16±0.03	0.09±0.06

Table 2. Accuracy of different baseline methods with and without applying interventions. ip: input intervention, op: output intervention. Here we use PaRWalk in the intervention technique. Photo: Amazon Photo and Computer: Amazon Computer.

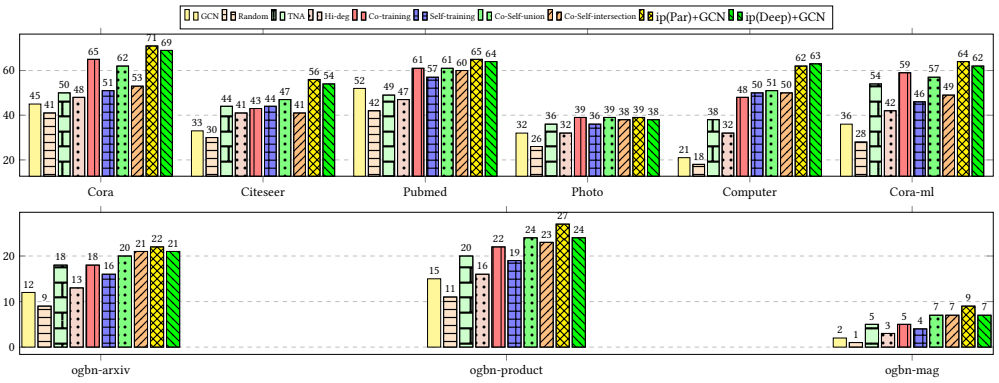


Fig. 6. Comparison of % accuracy (Y-axis) for different training set expansion methods: Training Node Augmentation (TNA), random sampling (Random), High-deg-path (Hi-deg), Co-training, Self-training, Co-Self-union, Co-Self-intersection and our proposed input interventions. ip (Par): input intervention with PaRWalk, ip (Deep): input intervention with DeepWalk, Photo: Amazon Photo and Computer: Amazon Computer.

5.5 Compute time on standard datasets

Tables 6 and 7 compares the time to execute all the different baseline methods while using PaRWalk and DeepWalk in the intervention pipeline respectively. The time to classify using the intervention methods will naturally be higher since more steps are required. In addition with various GNN baselines, it is evident from the tables that if the network

	GNN model	Networks								
		Cora	Citeseer	Pubmed	Photo	Computer	Cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
Baselines	GCN	0.45±0.05	0.33±0.06	0.52±0.05	0.32±0.09	0.21±0.09	0.36±0.04	0.12±0.04	0.15±0.03	0.02±0.04
	GAT	0.44±0.04	0.35±0.04	0.54±0.05	0.47±0.10	0.41±0.10	0.41±0.08	0.09±0.02	0.17±0.02	0.02±0.03
	ChebyNet	0.31±0.05	0.26±0.03	0.47±0.04	0.35±0.11	0.27±0.09	0.24±0.09	0.09±0.02	0.11±0.05	0.03±0.06
	GPR-GNN	0.48±0.06	0.38±0.05	0.62±0.08	0.54±0.13	0.44±0.17	0.52±0.09	0.14±0.02	0.21±0.02	0.05±0.03
	APPNP	0.50±0.05	0.36±0.04	0.59±0.04	0.23±0.11	0.10±0.10	0.36±0.10	0.14±0.03	0.20±0.04	0.05±0.03
	JKNet	0.35±0.04	0.27±0.05	0.57±0.04	0.17±0.10	0.32±0.10	0.23±0.11	0.08±0.03	0.12±0.02	0.02±0.03
	SAGE	0.37±0.05	0.30±0.05	0.56±0.05	0.42±0.11	0.32±0.10	0.31±0.11	0.04±0.03	0.11±0.06	0.02±0.07
Input intervention	ip+GCN	0.69±0.02	0.54±0.05	0.64±0.00	0.38±0.01	0.63±0.02	0.62±0.03	0.21±0.04	0.24±0.03	0.07±0.05
	ip+GAT	0.62±0.03	0.53±0.03	0.74±0.02	0.53±0.03	0.61±0.02	0.61±0.03	0.20±0.03	0.23±0.03	0.07±0.08
	ip+ChebyNet	0.67±0.01	0.61±0.03	0.85±0.00	0.50±0.03	0.51±0.01	0.58±0.03	0.19±0.03	0.18±0.05	0.09±0.04
	ip+GPR-GNN	0.68±0.01	0.60±0.03	0.81±0.01	0.60±0.03	0.50±0.08	0.66±0.03	0.23±0.03	0.24±0.05	0.07±0.04
	ip+APPNP	0.69±0.01	0.57±0.03	0.74±0.02	0.32±0.03	0.55±0.04	0.65±0.04	0.23±0.03	0.24±0.07	0.08±0.07
	ip+JKNet	0.53±0.01	0.41±0.03	0.75±0.01	0.28±0.01	0.47±0.03	0.43±0.02	0.17±0.03	0.18±0.06	0.06±0.06
	ip+SAGE	0.62±0.04	0.58±0.02	0.78±0.01	0.54±0.05	0.46±0.09	0.58±0.04	0.12±0.02	0.15±0.06	0.06±0.04
Output intervention	GCN+op	0.49±0.05	0.36±0.05	0.55±0.06	0.33±0.10	0.23±0.09	0.38±0.05	0.13±0.03	0.16±0.03	0.02±0.03
	GAT+op	0.45±0.03	0.39±0.04	0.54±0.06	0.48±0.10	0.43±0.12	0.45±0.07	0.11±0.04	0.17±0.05	0.03±0.08
	ChebyNet+op	0.35±0.04	0.27±0.03	0.48±0.04	0.38±0.10	0.28±0.09	0.27±0.08	0.10±0.06	0.13±0.03	0.04±0.05
	GPR-GNN+op	0.50±0.05	0.40±0.04	0.62±0.05	0.55±0.09	0.47±0.11	0.53±0.11	0.16±0.04	0.22±0.02	0.05±0.03
	APPNP+op	0.54±0.15	0.37±0.15	0.60±0.05	0.24±0.04	0.13±0.04	0.39±0.10	0.15±0.03	0.22±0.06	0.05±0.03
	JKNet+op	0.37±0.10	0.29±0.09	0.59±0.04	0.18±0.04	0.32±0.04	0.29±0.09	0.12±0.05	0.13±0.06	0.03±0.04
	SAGE+op	0.42±0.05	0.32±0.05	0.58±0.05	0.44±0.10	0.33±0.09	0.35±0.08	0.05±0.07	0.13±0.06	0.03±0.04
Both intervention	ip+GCN+op	0.71±0.03	0.57±0.02	0.66±0.03	0.40±0.04	0.63±0.05	0.66±0.05	0.23±0.03	0.25±0.05	0.07±0.04
	ip+GAT+op	0.64±0.01	0.56±0.03	0.77±0.03	0.54±0.07	0.62±0.10	0.64±0.02	0.22±0.05	0.23±0.04	0.08±0.03
	ip+ChebyNet+op	0.68±0.03	0.63±0.04	0.85±0.01	0.51±0.01	0.52±0.08	0.62±0.01	0.22±0.04	0.18±0.06	0.09±0.03
	ip+GPR-GNN+op	0.70±0.03	0.62±0.03	0.83±0.01	0.61±0.06	0.52±0.06	0.69±0.03	0.25±0.03	0.25±0.05	0.09±0.04
	ip+APPNP+op	0.70±0.02	0.60±0.03	0.74±0.01	0.33±0.06	0.56±0.07	0.68±0.01	0.25±0.03	0.24±0.06	0.08±0.04
	ip+JKNet+op	0.56±0.04	0.42±0.03	0.76±0.01	0.29±0.06	0.57±0.09	0.44±0.01	0.19±0.07	0.19±0.06	0.07±0.05
	ip+SAGE+op	0.64±0.02	0.62±0.02	0.78±0.01	0.55±0.07	0.47±0.09	0.61±0.02	0.14±0.03	0.16±0.03	0.08±0.05

Table 3. Accuracy of different baseline methods with and without applying interventions. ip: input intervention, op: output intervention. Here we use DeepWalk in our intervention techniques. Photo: Amazon Photo and Computer: Amazon Computer.

GNN vs GNN+op	Networks								
	cora	citeseer	pubmed	photo	computer	cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
GCN vs GCN+op	0.002	0.001	0.002	0.011	0.012	0.004	0.002	0.01	0.02
GAT vs GAT+op	0.001	0.005	0.0001	0.001	0.011	0.004	0.004	0.02	0.04
ChebyNet vs ChebyNet+op	0.002	0.004	0.003	0.002	0.007	0.002	0.003	0.01	0.03
GPRGNN vs GPRGNN+op	0.001	0.03	0.002	0.005	0.02	0.004	0.002	0.01	0.04
APPNP vs APPNP+op	0.002	0.005	0.009	0.003	0.01	0.01	0.004	0.02	0.02
JKNet vs JKNet+op	0.01	0.002	0.0003	0.010	0.034	0.003	0.002	0.01	0.04
SAGE vs SAGE+op	0.0006	0.0007	0.001	0.0001	0.008	0.003	0.001	0.01	0.04

Table 4. Significance tests with output interventions plugged in. ParWalk is used in the intervention methods.

GNN vs GNN+op	Networks								
	cora	citeseer	pubmed	photo	computer	cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
GCN vs GCN+op	0.003	0.001	0.001	0.013	0.02	0.001	0.002	0.02	0.02
GAT vs GAT+op	0.003	0.002	0.001	0.002	0.01	0.005	0.004	0.01	0.02
ChebyNet vs ChebyNet+op	0.001	0.002	0.003	0.01	0.02	0.012	0.002	0.02	0.01
GPRGNN vs GPRGNN+op	0.004	0.002	0.002	0.001	0.02	0.002	0.001	0.02	0.03
APPNP vs APPNP+op	0.001	0.005	0.003	0.002	0.02	0.012	0.006	0.01	0.03
JKNet vs JKNet+op	0.001	0.004	0.004	0.002	0.03	0.001	0.002	0.02	0.03
SAGE vs SAGE+op	0.002	0.0008	0.004	0.001	0.08	0.001	0.002	0.02	0.02

Table 5. Significance tests with output interventions plugged in. DeepWalk is used in the intervention methods.

is large (such as computer), then more time is needed for our intervention method. In case of PaRWalk, the actual time requirement is due to the calculation of the absorption probability and in case of DeepWalk, the actual time requirement is mainly due to two parameters: random walk length and number of times the random walk is required.

	GNN model	Networks								
		cora	citeseer	pubmed	photo	computer	cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
Baselines	GCN	3s	5s	24s	36s	73s	5s	88s	32015s	1608s
	GAT	35s	72s	207s	300s	601s	57s	626s	35393s	2551s
	ChebyNet	15s	44s	51s	117s	248s	38s	261s	33265s	1987s
	GPRGNN	9s	18s	38s	67s	147s	17s	166s	32965s	1843s
	APPNP	9s	17s	37s	66s	147s	16s	163s	32958s	1839s
	JKNet	8s	11s	64s	40s	85s	10s	93s	32329s	1679s
	SAGE	9s	26s	46s	98s	187s	23s	253s	32768s	1983s
	Random-sample	4s	6s	26s	38s	78s	6s	90s	32037s	1629s
	TNA	8s	14s	50s	62s	156s	11s	95s	32048s	1641s
	Hi-deg-Path	4s	8s	31s	46s	92s	9s	93s	32037s	1631s
	Co-training	10s	13s	241s	131s	756s	11s	345s	33135s	2369s
	Self-training	9s	18s	161s	86s	547s	10s	324s	33124s	2213s
	Co-Self-union	15s	27s	347s	185s	922s	18s	548s	51567s	3516s
	Co-Self-intersection	16s	27s	347s	185s	922s	18s	537s	51769s	3532s
Input intervention	ip+GCN	11s	15s	248s	160s	880s	13s	940s	47763s	37593s
	ip+GAT	44s	75s	432s	424s	1416s	71s	1788s	51194s	38566s
	ip+ChebyNet	22s	56s	278s	241s	967s	47s	1164s	49094s	37989s
	ip+GPRGNN	16s	27s	261s	193s	957s	30s	1025s	48794s	37856s
	ip+APPNP	18s	29s	259s	191s	956s	27s	1022s	48781s	37849s
	ip+JKNet	17s	18s	283s	167s	898s	19s	947s	48091s	37687s
	ip+SAGE	19s	37s	269s	225s	1003s	33s	1116s	48534s	37973s
Output intervention	GCN+op	5s	9s	240s	144s	792s	9s	731s	37266s	5127s
	GAT+op	36s	74s	422s	408s	1320s	66s	1582s	40649s	6079s
	ChebyNet+op	18s	45s	270s	227s	1060s	43s	983s	38526s	5506s
	GPRGNN+op	10s	20s	252s	188s	864s	26s	834s	38253s	5373s
	APPNP+op	10s	18s	250s	186s	863s	25s	830s	38244s	5368s
	JKNet+op	9s	12s	274s	161s	804s	15s	757s	37562s	5207s
	SAGE+op	12s	28s	263s	209s	919s	32s	996s	39037s	5723s
Both intervention	ip+GCN+op	12s	19s	468s	270s	1604s	18s	1586s	52989s	41126s
	ip+GAT+op	46s	77s	651s	534s	2135s	81s	2748s	56486s	42122s
	ip+ChebyNet+op	23s	59s	497s	351s	1785s	52s	1889s	54361s	41566s
	ip+GPRGNN+op	19s	33s	477s	314s	1676s	40s	1695s	54082s	41427s
	ip+APPNP+op	19s	30s	474s	312s	1673s	39s	1693s	54072s	41405s
	ip+JKNet+op	20s	23s	495s	288s	1620s	24s	1613s	53378s	41233s
	ip+SAGE+op	22s	40s	481s	343s	1749s	43s	1868s	54785s	41798s

Table 6. Time requirement for different methods. PaRWalk is used in the intervention methods.

5.6 Effects on increasing the number of hidden layers

In this section, we describe how the number of layers affects the accuracy of the GNN in the input intervention. Figs. 7 and 8 shows the results of the GNNs equipped with only input intervention on cora network (the trends for the other networks are similar). Except for GPRGNN and APPNP, we can see that the accuracy does not get affected much as we increase the number of layers. For GPRGNN and APPNP, transition from layer 3 to layer 4 results in a sudden drop in the accuracy.

5.7 Sensitivity analysis

Both the PaRWalk and DeepWalk have certain parameters that might affect the accuracy of our results. In this section we study the influence of such parameters by varying a certain parameter while keeping others fixed. We did this study for three networks: cora, citeseer and pubmed (results for other networks not shown for brevity). As a specific

	GNN model	Networks								
		cora	citeseer	pubmed	photo	computer	cora-ml	ogbn-arxiv	ogbn-product	ogbn-mag
Baselines	GCN	3s	5s	24s	36s	73s	5s	88s	32015s	1608s
	GAT	35s	72s	207s	300s	601s	57s	626s	35393s	2551s
	ChebyNet	15s	44s	51s	117s	248s	38s	261s	33265s	1987s
	GPRGNN	9s	18s	38s	67s	147s	17s	166s	32965s	1843s
	APPNP	9s	17s	37s	66s	147s	16s	163s	32958s	1839s
	JKNet	8s	11s	64s	40s	85s	10s	93s	32329s	1679s
	SAGE	9s	26s	46s	98s	187s	23s	253s	32768s	1983s
	Random-sample	4s	6s	26s	38s	78s	6s	90s	32037s	1629s
	TNA	8s	14s	50s	62s	156s	11s	95s	32048s	1641s
	Hi-deg-Path	4s	8s	31s	46s	92s	9s	93s	32037s	1631s
	Co-training	10s	13s	241s	131s	756s	11s	345s	33135s	2369s
	Self-training	9s	18s	161s	86s	547s	10s	324s	33124s	2213s
	Co-Self-union	15s	27s	347s	185s	922s	18s	548s	51567s	3516s
	Co-Self-intersection	16s	27s	347s	185s	922s	18s	537s	51769s	3532s
Input intervention	ip+GCN	8s	11s	213s	138s	683s	11s	1123s	49567s	38251s
	ip+GAT	41s	78s	398s	401s	1201s	68s	1969s	52963s	31998s
	ip+ChebyNet	21s	51s	246s	192s	854s	42s	1347s	50823s	38639s
	ip+GPRGNN	14s	24s	229s	171s	776s	26s	1208s	50492s	38486s
	ip+APPNP	15s	23s	227s	169s	774s	26s	1205s	50483s	38482s
	ip+JKNet	13s	17s	254s	143s	702s	13s	1128s	49862s	38327s
	ip+SAGE	16s	33s	246s	206s	805s	32s	1293s	50331s	38632s
Output intervention	GCN+op	5s	8s	202s	123s	608s	8s	921s	38127s	5632s
	GAT+op	37s	74s	389s	385s	1133s	54s	1764s	41625s	6591s
	ChebyNet+op	17s	47s	237s	174s	781s	41s	1144s	39395s	6042s
	GPRGNN+op	10s	19s	217s	156s	699s	20s	1004s	39097s	5883s
	APPNP+op	10s	19s	217s	154s	699s	20s	1002s	39085s	5875s
	JKNet+op	9s	13s	245s	127s	625s	12s	922s	38484s	5021s
	SAGE+op	13s	30s	227s	193s	725s	30s	1093s	38880s	6018s
Both intervention	ip+GCN+op	12s	16s	391s	230s	1221s	15s	1956s	55682s	42283s
	ip+GAT+op	45s	80s	583s	486s	1743s	69s	3107s	59203s	36025s
	ip+ChebyNet+op	25s	54s	435s	248s	1395s	52s	2230s	56967s	42698s
	ip+GPRGNN+op	18s	28s	411s	293s	1346s	30s	2048s	56582s	42539s
	ip+APPNP+op	18s	28s	408s	291s	1341s	30s	2044s	56571s	42522s
	ip+JKNet+op	17s	22s	439s	231s	1249s	17s	1959s	56029s	41713s
	ip+SAGE+op	23s	38s	431s	307s	1361s	41s	2153s	56461s	42683s

Table 7. Time requirement for different input intervention methods. DeepWalk is used in the intervention methods.

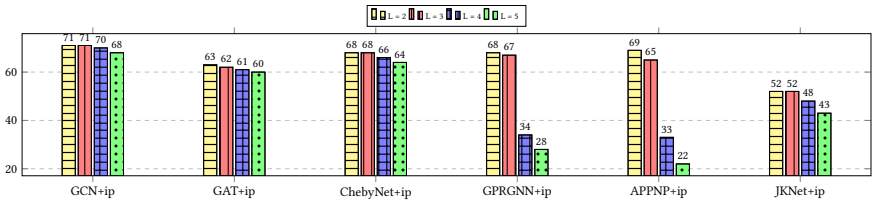


Fig. 7. Effects on the % accuracy (Y-axis) for different GNN models while varying the number of layers in cora network. L: Number of hidden layers in the GNN. PaRWalk is used for input intervention.

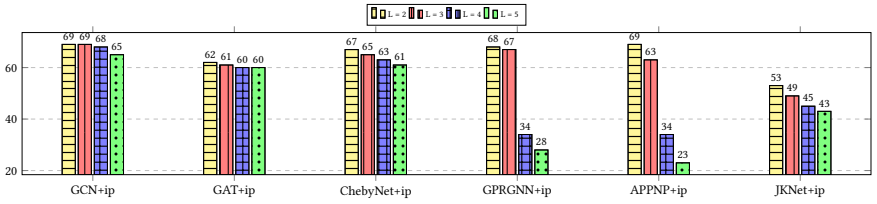


Fig. 8. Effects on the % accuracy (Y-axis) for different GNN models while varying the number of layers in cora network. L: Number of hidden layers in the GNN. DeepWalk is used for input intervention.

GNN model, we picked GCN since it is faster among all and investigate only the input level intervention for both these random walk methods.

Tuning PaRWalk's parameters: The only parameter in ParWalk is the absorption probability α. We tune it to 10 values starting from 10^{-6} to 10^{-1} and plot the results in Figure 9. As can be seen, the change in accuracy is negligible as we increase α from 10^{-6} to 10^{-1}. This observation states that the accuracy of GCN is not much sensitive to the absorption probability.

Tuning DeepWalk's parameters: Set of important parameters in DeepWalk method includes (i) window size $\mathcal{W}_S$, (ii) walks per node $\mathcal{W}_P$, (iii) walk length $\mathcal{W}_L$. Figure 10 shows the accuracy of the GCN while tuning various parameters. In Figure 10(a), we can see that the accuracy slowly increases as we increase the window size while keeping walks per node to 1 and walk length to 2. In Figure 10(b), the accuracy first suddenly increases and then stabilizes as number of walks per node increases from 1, while keeping other two parameters fixed to 2. In Figure 10(c), the trends are similar to Figure 10(b) (we keep the other parameters - window size and walks per node to 2 and 1 respectively). For all the parameters, the accuracy quickly stabilizes showing that the random walk has converged and any change in the node embeddings after this point is extremely rare.

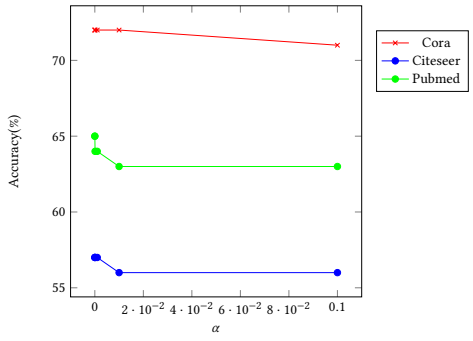


Fig. 9. Sensitivity analysis on ParWalk by tuning various parameters while performing input intervention. The absorption probability α is varied from 0.1 to 10^{-6}

6 RELATED WORK

Random walks. A random walk is a random process of moving from one node to another node and creating a path. According to to [52], every Markov chain can be thought of as a random walk on a directed graph, whereas, a random walk in an undirected graph is a time-reversible Markov chain. [60] investigated random walks on complex networks and derived the exact expression for the mean first-passage time between two nodes. Variants include random walk with restart [62], lazy random walk [73], personalized PageRank [29], PaRWalk [90], etc. In [81] random walk is used to extend the original node2vec node-neighborhood sampling method and generate a second-order random walk sampling for heterogeneous multi-graphs. Multiple applications are built using random walk based algorithms including link prediction [51], recommendation system [33, 34], computer vision [73], semi-supervised learning [110, 111], network embedding [35, 63], complex social network analysis [69] etc. Several surveys on random walk include [11, 69, 92].

Node embedding techniques based on random walks. Node embedding techniques are popular graph representation learning methods [37]. Few popular random walk based node embedding techniques include DeepWalk [63],

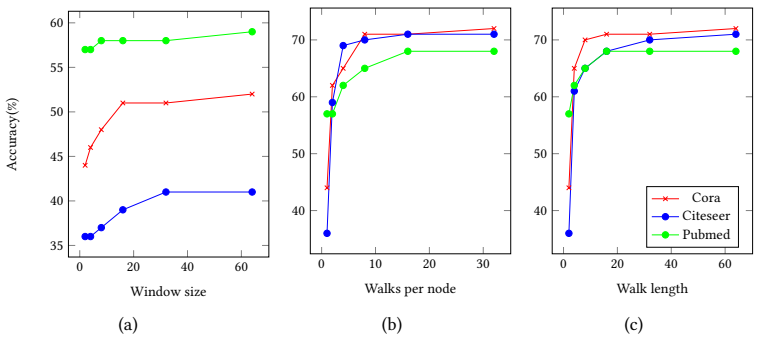


Fig. 10. Sensitivity analysis on DeepWalk by tuning various parameters while performing input intervention. (a) Varying window size, while fixing walks per node = 1 and walk length = 2. (b) Varying walks per node, while fixing window size = 2 and walk length = 2. (c) Varying Walk length, while fixing window size = 2 and walks per nodes = 1

Node2Vec [35], Large-scale Information Network Embedding (LINE) [80], Asymmetric Proximity Preserving (APP) [104] etc. DeepWalk applies standard random-walks, while Node2Vec considers biased random walks and APP adopts rooted PageRank, among others. LINE operates by optimising an objective function that keeps both the global and local network structures intact. *Huang et. al.* [41] broadly categorizes the random walk based node embedding process into two categories: Pointwise Mutual Information (PMI) and Auto-covariance. DeepWalk [63], WalkLets [64], NetMF [66], Node2Vec [35], LINE [80] and NetSMF [65] fall into the PMI group whereas, Multiscale [70] and their (*Huang et. al.*'s) proposed method [41] falls in the auto-covariance group. Further surveys on node embedding techniques based on random walk can be found in the following articles [12, 16, 24, 88].

Graph-based semi-supervised learning. Semi-supervised learning (SSL) [17, 82] and more specifically, graph-based SSL (GSSL) [22, 105, 111] have become popular over the last few years. It deals with spreading labels from a small set of labeled data points to a wider number of unlabeled data. In order to learn from both labeled and unlabelled data, some GSSL [8] exploited the pairwise correlations among the nodes in the min-cut technique. Some approach takes into account the cluster assumption [18], which specifies that the decision boundary should not intersect densely populated areas. Authors in [5, 42] used the spectral method for the semi-supervised learning task. [40] discussed the consistency of optimization-based techniques for the GSSL, with the assumption that the labels that have small noise and the unlabelled data are well clustered. Other works include the inclusion of label propagation method using Gaussian field and harmonic functions [111], random walk [79, 90], transductive SVM [23], combination of label propagation and bipartite graph construction [89] etc.

It is crucial to reduce the burden of computational and storage costs for GSSL approaches given the rapidly rising data volume. By utilizing the convergence of the eigenvectors of the normalised graph Laplacian to the eigen-functions of weighted Laplace-Beltrami operators, *Fergus et. al.* [27] provided a method for constructing numerical approximations to the eigenvectors of normalized graph Laplacian. Few GSSL methods consider the feature vectors associated with the data along with the graph structure that includes regularization tasks. For example, *Zhang et. al.* [101] suggested employing prototype vectors in order to approximate the graph-based regularizer in GSSL with the assumption of low-rank approximation and little information loss. *Belkin et. al.* [4] proposed some algorithms for regularization on graphs. The suggested algorithms are fairly straightforward and solve a single system of linear equations, which is typically sparse. *Zhou et. al.* [107] proposed GSSL by higher order regularization where they used Iterated Laplacian

regularisation, which is comparable to a higher order Sobolev semi-norm. *Li et.al.* [46] used preconditioned conjugate gradient descent and Nystrom subsampling to boost the effectiveness of Laplacian Regularized Least Squares. *Belkin et. al.* [6] presented a methodology for data-dependent regularisation that takes advantage of the probability distribution's geometry. Recent surveys on GSSL can be found in [77, 78].

Graph neural networks for node classification. GNN [91] is a powerful tool for various graph-based semi-supervised tasks such as node classification [25], link prediction [48], graph classification [25, 93], graph embedding [13], and so on. Node classification is one of the major tasks in the network domain since it has a lot of applications in several areas such as text classification [98], neural machine translation [56], molecular fingerprints learning [26] etc. Apart from [25], other tools for node classification tasks include [36, 44, 84], etc. The development of GNN for node categorization is hampered by over-fitting and over-smoothing. Therefore current works on GNN mostly focus on alleviating these issues. For example, in [47], the authors improved the accuracy of the GCN by extending the training set using PaRWalk [90]. Jumping Knowledge Networks (JKNet) [94] is another alternative, where for each node, the neighborhood features are weighted differently during aggregation. Mutual Teaching for Graph Convolutional Networks (MT-GCN) [99] is inspired by knowledge distillation [39] and label smoothing [59]. The model uses both the temperature of the softmax layer and the ground truth labels for training label expansion as well as teaching the peer network.

However, knowledge distillation techniques typically suffer from either subpar distillation brought on by insufficient use of unlabeled data or overconfident and biased pseudo-labels. To circumvent this problem, *Luo et. al.* [54] offers DualGraph, a guiding framework to more efficiently use unlabeled graphs for semi-supervised graph classification, motivated by current developments in contrastive learning [15] and dual learning [38]. Approximate personalized propagation of neural predictions (APPNP) [45] utilizes PageRank [61] along with GCN [44] to derive an improved message passing scheme in the graph. Generalized PageRank Graph Neural Network (GPRGNN) [21] has been inspired by the APPNP model that proposed a generalized PageRank GNN that overcomes the problems of over-fitting and over-smoothing. DropEdge [67] acts as a data augmenter and a message passing reducer by randomly removing a specific number of edges from the input graph during each training epoch. Iterative Deep Graph Learning (IDGL) [20] learns graph structure and graph embedding jointly and iteratively and, thereby, improves the accuracies. Deep Graph Infomax (DGI) [83] is based on maximising mutual information between patch representations and corresponding high-level graph summaries, both of which are derived using well-established graph convolutional network architectures.

The Graph Harmonic Neural Network (GHNN) [43] which consists of two modules: a graph kernel network (GKN) module [19] and a graph convolutional network (GCN) module that examines graph topology data from contrasting angles. By providing emphasis to high-quality unlabeled data during the training of two modules, they created a novel harmonic contrastive loss and a harmonic consistency loss to harmonise the training of two modules and reconcile the consistency of their predictions.

7 DISCUSSIONS AND FUTURE WORK

We present methods for input and output level intervention on GNN to improve its accuracy. In the input intervention, we extend the training set with a set of other training nodes of the same class by carefully selecting nodes from different non-contiguous subgraphs. We use variations of random walks or node embedding techniques to extend the training sets and K-means followed by K-NN to increase the diversity of the nodes. In the output intervention, we use a random walk or node embedding technique in a similar vein to identify the misclassified nodes and relabel them to correct class labels using the *confidence* of the nodes. We now discuss some of the possible extensions of our work.

Application in inductive setting. Our research is focused on transductive scenarios in which we are aware of the testing nodes throughout training. Another situation is referred to as an inductive setting [7], in which we are unaware of which nodes are the training nodes during training. Inductive learning makes the assumption that there are some rules that can be applied by the model and that allow us to categorize a pattern given its attributes. In contrast to the transductive setting, the inductive node embedding problem is particularly challenging because, in order to generalize to unseen nodes, freshly observed subgraphs must be "aligned" to the node embeddings that the algorithm has been already optimized for. A node's neighborhood's structural characteristics that disclose both the node's local function in the graph and its global position must be recognized through an inductive framework.

Both of our intervention methods use random walk which is used to pick up the set of nodes very similar to the original nodes but is agnostic of the node type. Therefore to extend our methods to the inductive framework we have to incorporate the random walk into the inductive functions. Note that the subsequent steps of clustering using K-means and K-NN do not depend on the knowledge of which nodes are the training nodes.

Application to hypergraphs. A hypergraph is a generalization of a graph in which multiple nodes can be connected by an edge (relationship). There are datasets with relationships between the nodes that are not pairwise, such as email, co-citation, co-authorship, correspondence, etc. Hypergraphs enable us to model linkages in such cases. Some popular works based on hypergraph can be found in [76, 100, 106]. Some works based on the application of the GNNs on the hypergraph can be found in [3, 95, 102]. *Bai et. al.* [3] demonstrated mathematically that when a non-pairwise relationship degenerates into a pairwise one, graph convolution is a specific case of hypergraph convolution. Therefore, our suggested intervention methodology, which is based on GNNs for ordinary graph settings, can be applied to hypergraph settings, where the random walks (on the ordinary graph) that we now utilize in our intervention technique need to be converted to random walks on hypergraph [14].

Application to heterophily graphs. A heterophily graph has a very low homophily value, where the homophily of a node v can be defined as the ratio of the number of neighbors having same labels as v to the number of neighbors of v. The homophily value of a graph is the sum of the homophily values of all the nodes normalised by the number of nodes. Typically, in a heterophily graph, a large number of edges exist whose two terminals have different class labels. In this paper, only the homophily graphs are studied. But there exist real world graphs such as Chameleon [68], Squirrel [68], Cornell [31], Texas [31] that are heterophily in nature. Existing works on applying the GNN on heterophily networks include [9, 21, 97, 108]. A recent survey on heterophily network can be found in [103]. As our future work, we plan to integrate some non-local neighborhood extension methods such as high-order neighborhood mixing [1, 109] or potential neighborhood discovery [50, 87] methods with our input intervention technique, so that not only we can extend the training set (with the help of the input intervention), but also the model can correctly choose the neighbors (with the help of these non-local neighbor extension methods) of each node in the network for the feature aggregation and updation steps. In similar lines, the output intervention technique can also be modified so that it can correctly recognise the class labels of the neighbours of a node even in the heterophily environment and relabel them properly.

In this paper, we have concentrated on input and output interventions. In the future, we aim to introduce in-process interventions, making changes to the GNN architecture itself, as well as extend the model for the inductive and hypergraph settings. We also plan to introduce parallelism in the entire pipeline so that the methods can scale to very large networks faster.

Reproducibility. All our datasets are public. The code is available at: https://github.com/anjangit000/inputGCN/tree/master

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29.

[2] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, Dániel Marx (Ed.). SIAM, 522–539. https://doi.org/10.1137/1.9781611976465.32

[3] Song Bai, Feihu Zhang, and Philip H.S. Torr. 2021. Hypergraph convolution and hypergraph attention. *Pattern Recognition* 110 (2021), 107637. https://doi.org/10.1016/j.patcog.2020.107637

[4] Mikhail Belkin, Irina Matveeva, and Partha Niyogi. 2004. Regularization and semi-supervised learning on large graphs. In *International Conference on Computational Learning Theory*. Springer, 624–638.

[5] Mikhail Belkin and Partha Niyogi. 2004. Semi-supervised learning on Riemannian manifolds. *Machine learning* 56, 1 (2004), 209–239.

[6] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *Journal of Machine Learning Research* 7, 85 (2006), 2399–2434. http://jmlr.org/papers/v7/belkin06a.html

[7] Monica Bianchini, Anas Belahcen, and Franco Scarselli. 2016. A comparative study of inductive and transductive learning with feedforward neural networks. In *Conference of the Italian Association for Artificial Intelligence*. Springer, 283–293.

[8] Avrim Blum and Shuchi Chawla. 2001. Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 19–26. https://doi.org/10.1184/R1/6606860.v1

[9] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 3950–3957.

[10] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1ZdKJ-0W

[11] Raffaella Burioni and Davide Cassi. 2005. Random walks on graphs: ideas, techniques and results. *Journal of Physics A: Mathematical and General* 38, 8 (2005), R45.

[12] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[13] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations (access date: 07-Aug-2023). *AAAI* (2016), 1145–1152. https://doi.org/10.1609/aaai.v30i1.10179

[14] Timoteo Carletti, Federico Battiston, Giulia Cencetti, and Duccio Fanelli. 2020. Random walks on hypergraphs. *Physical review E* 101, 2 (2020), 022308.

[15] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. *Advances in Neural Information Processing Systems* 33 (2020), 9912–9924. https://proceedings.neurips.cc/paper_files/paper/2020/file/70feb62b69f16e0238f741fab228fec2-Paper.pdf

[16] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2022. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research* 23, 89 (2022), 1–64.

[17] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien (Eds.). 2006. Semi-Supervised Learning (access date: 07-Aug-2023). (2006). http://dblp.uni-trier.de/db/books/collections/CSZ2006.html

[18] Olivier Chapelle and Alexander Zien. 2005. Semi-supervised classification by low density separation. In *International workshop on artificial intelligence and statistics*. PMLR, 57–64.

[19] Dexiong Chen, Laurent Jacob, and Julien Mairal. 2020. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning*. PMLR, 1576–1586.

[20] Yu Chen, Lingfei Wu, and Mohammed Zaki. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.). *Proceedings of the 34th International Conference on Neural Information Processing Systems* 33, Article 1620, 19314–19326 pages. https://proceedings.neurips.cc/paper_files/paper/2020/file/e05c7ba4e087beea9410929698dc41a6-Paper.pdf

[21] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*. https://openreview.net/forum?id=n6jl7fLxrP

[22] Yanwen Chong, Yun Ding, Qing Yan, and Shaoming Pan. 2020. Graph-based semi-supervised learning: A review. *Neurocomputing* 408 (2020), 216–230. https://doi.org/10.1016/j.neucom.2019.12.130

[23] Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. 2006. Large Scale Transductive SVMs. *Journal of Machine Learning Research* 7, 62 (2006), 1687–1712. http://jmlr.org/papers/v7/collobert06a.html

[24] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE transactions on knowledge and data engineering* 31, 5 (2018), 833–852.

[25] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Proceedings of the 30th International Conference on Neural Information Processing Systems* 29, 3844–3852. https://doi.org/10.5281/zenodo.

1318406

[26] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2224–2232.

[27] Rob Fergus, Yair Weiss, and Antonio Torralba. 2009. Semi-Supervised Learning in Gigantic Image Collections. In *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (Eds.), Vol. 22. Curran Associates, Inc., 522–530. https://proceedings.neurips.cc/paper_files/paper/2009/file/1651cf0d2f737d7adeab84d339dbabd3-Paper.pdf

[28] Fix, Evelyn, and J. L. Hodges. 1989. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review* 57, 3 (1989), 238–247. https://doi.org/10.2307/1403797

[29] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments. *Internet Math.* 2, 3 (2005), 333–358. http://www.ams.org/mathscinet-getitem?mr=2212369

[30] E. Forgy. 1965. Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classification. *Biometrics* 21, 3 (1965), 768–769.

[31] Rayid Ghani. 2001. CMU World Wide Knowledge Base (WebKB) project. https://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/. (2001).

[32] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning (access date: 07-Aug-2023)*. MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org.

[33] Marco Gori and Augusto Pucci. 2006. Research paper recommender systems: A random-walk based approach. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*. IEEE, 778–781.

[34] Marco Gori, Augusto Pucci, V Roma, and I Siena. 2007. Itemrank: A random-walk based scoring algorithm for recommender engines.. In *IJCAI*, Vol. 7. 2766–2771.

[35] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., 1024–1034. https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[37] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74. http://sites.computer.org/debull/A17sept/p52.pdf

[38] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual Learning for Machine Translation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 820–828.

[39] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*. http://arxiv.org/abs/1503.02531

[40] Franca Hoffmann, Bamdad Hosseini, Zhi Ren, and Andrew M. Stuart. 2020. Consistency of Semi-Supervised Learning Algorithms on Graphs: Probit and One-Hot Methods. *J. Mach. Learn. Res.* 21, 1, Article 186 (jan 2020), 55 pages.

[41] Zexi Huang, Arlei Silva, and Ambuj Singh. 2021. A broader picture of random-walk based graph embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 685–695.

[42] Thorsten Joachims. 2003. Transductive learning via spectral graph partitioning. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 290–297.

[43] Wei Ju, Xiao Luo, Zeyu Ma, Junwei Yang, Minghua Deng, and Ming Zhang. 2022. GHNN: Graph Harmonic Neural Networks for semi-supervised graph-level classification. *Neural Networks* 151, C (2022), 70–79. https://doi.org/10.1016/j.neunet.2022.03.018

[44] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (Palais des Congrès Neptune, Toulon, France) *(ICLR '17)*. https://openreview.net/forum?id=SJU4ayYgl

[45] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. *7th International Conference on Learning Representations, ICLR (Poster) 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=H1gL-2A9Ym

[46] Jian Li, Yong Liu, Rong Yin, and Weiping Wang. 2019. Approximate Manifold Regularization: Scalable Algorithm and Generalization Analysis.. In *IJCAI*. 2887–2893.

[47] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, Article 433, 8 pages. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098

[48] Zhao Li, Zhanlin Liu, Jiaming Huang, Geyu Tang, Yucong Duan, Zhiqiang Zhang, and Yifan Yang. 2019. MV-GCN: Multi-View Graph Convolutional Networks for Link Prediction. *IEEE Access* 7 (2019), 176317–176328. https://doi.org/10.1109/ACCESS.2019.2957306

[49] Xingxing Liang, Yang Ma, Guangquan Cheng, Changjun Fan, Yuling Yang, and Zhong Liu. 2022. Meta-path-based heterogeneous graph neural networks in academic network. *International Journal of Machine Learning and Cybernetics* 13, 6 (2022), 1553–1569.

[50] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2021. Non-local graph neural networks. *IEEE transactions on pattern analysis and machine intelligence* 44, 12 (2021), 10270–10276.

[51] Weiping Liu and Linyuan Lü. 2010. Link prediction based on local random walk. *EPL (europhysics Letters)* 89, 5 (2010), 58007.

[52] László Lovász. 1993. Random walks on graphs. *Combinatorics, Paul erdos is eighty* 2, 1-46 (1993), 4.

[53] L. Lovasz and M. Simonovits. 1990. The mixing rate of Markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 346–354 vol. 1. https://doi.org/10.1109/FSCS.1990.89553

[54] Xiao Luo, Wei Ju, Meng Qu, Chong Chen, Minghua Deng, Xian-Sheng Hua, and Ming Zhang. 2022. DualGraph: Improving Semi-supervised Graph Classification via Dual Contrastive Learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 699–712. https://doi.org/10.1109/ICDE53745.2022.00057

[55] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50 – 60. https://doi.org/10.1214/aoms/1177730491

[56] Diego Marcheggiani, Jasmijn Bastings, and Ivan Titov. 2018. Exploiting Semantics in Neural Machine Translation with Graph Convolutional Networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 486–492. https://doi.org/10.18653/v1/N18-2078

[57] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52. https://doi.org/10.1145/2766462.2767755

[58] James B McQueen. 1967. Some methods of classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.* 281–297.

[59] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. 2019. *When Does Label Smoothing Help?* Curran Associates Inc., Red Hook, NY, USA.

[60] Jae Dong Noh and Heiko Rieger. 2004. Random Walks on Complex Networks. *Physical Review Letters* 92, 11 (Mar 2004), 1–4. https://doi.org/10.1103/physrevlett.92.118701

[61] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[62] Jia-Yu Pan, Hyung-jeong Yang, Christos Faloutsos, and Pinar Duygulu. 2004. Automatic Multimedia Cross-modal Correlation Discovery. *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (07 2004). https://doi.org/10.1145/1014052.1014135

[63] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (Aug 2014). https://doi.org/10.1145/2623330.2623732

[64] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't walk, skip! online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. 258–265.

[65] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. Netsmf: Large-scale network embedding as sparse matrix factorization. In *The World Wide Web Conference*. 1509–1520.

[66] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 459–467.

[67] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. https://openreview.net/forum?id=Hkx1qkrKPr

[68] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* 9, 2 (2021), cnab014.

[69] Purnamrita Sarkar and Andrew W Moore. 2011. Random walks in social networks and their applications: a survey. In *Social Network Data Analytics*. Springer, 43–77.

[70] Michael T Schaub, Jean-Charles Delvenne, Renaud Lambiotte, and Mauricio Barahona. 2019. Multiscale dynamical embeddings of complex networks. *Physical Review E* 99, 6 (2019), 062308.

[71] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008), 93–106. http://www.cs.iit.edu/~ml/pdfs/sen-aimag08.pdf

[72] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *In Relational Representation Learning Workshop, Advances in Neural Information Processing Systems (NeurIPS)* (2018).

[73] Jianbing Shen, Yunfan Du, Wenguan Wang, and Xuelong Li. 2014. Lazy Random Walks for Superpixel Segmentation. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 23, 4 (04 2014), 1451–1462. https://doi.org/10.1109/TIP.2014.2302892

[74] Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. 2017. SetExpan: Corpus-Based Set Expansion via Context Feature Selection and Rank Ensemble. In *Machine Learning and Knowledge Discovery in Databases*, Michelangelo Ceci, Jaakko Hollmén, Ljupčo Todorovski, Celine Vens, and Sašo Džeroski (Eds.). Springer International Publishing, Cham, 288–304.

[75] Amit Singhal. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 24, 4 (2001), 35–43. http://dblp.uni-trier.de/db/journals/debu/debu24.html#Singhal01

[76] Tasuku Soma and Yuichi Yoshida. 2019. Spectral sparsification of hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2570–2581.

[77] Yunsheng Song, Jing Zhang, and Chao Zhang. 2022. A survey of large-scale graph-based semi-supervised classification algorithms. *International Journal of Cognitive Computing in Engineering* 3 (2022), 188–198. https://doi.org/10.1016/j.ijcce.2022.10.002

[78] Zixing Song, Xiangli Yang, Zenglin Xu, and Irwin King. 2022. Graph-Based Semi-Supervised Learning: A Comprehensive Review. *IEEE Transactions on Neural Networks and Learning Systems* PP (2022), 1–21. https://doi.org/10.1109/TNNLS.2022.3155478

[79] Martin Szummer and Tommi Jaakkola. 2001. Partially labeled classification with Markov random walks, T. Dietterich, S. Becker, and Z. Ghahramani (Eds.). *Advances in Neural Information Processing Systems* 14, 945–952. https://proceedings.neurips.cc/paper_files/paper/2001/file/a82d922b133be19c1171534e6594f754-Paper.pdf

[80] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-Scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1067–1077. https://doi.org/10.1145/2736277.2741093

[81] Giorgio Valentini, Elena Casiraghi, Luca Cappelletti, Vida Ravanmehr, Tommaso Fontana, Justin T. Reese, and Peter N. Robinson. 2021. Het-node2vec: second order random walk sampling for heterogeneous multigraphs embedding. *CoRR* abs/2101.01425 (2021). arXiv:2101.01425 https://arxiv.org/abs/2101.01425

[82] Jesper E Van Engelen and Holger H Hoos. 2020. A survey on semi-supervised learning. *Machine Learning* 109, 2 (2020), 373–440.

[83] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.

[84] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *6th International Conference on Learning Representations* (2017).

[85] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[86] Richard C. Wang and William W. Cohen. 2008. Iterative Set Expansion of Named Entities Using the Web. In *2008 Eighth IEEE International Conference on Data Mining*. 1091–1096. https://doi.org/10.1109/ICDM.2008.145

[87] Tao Wang, Di Jin, Rui Wang, Dongxiao He, and Yuxiao Huang. 2022. Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4210–4218.

[88] Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2018. A brief review of network embedding. *Big Data Mining and Analytics* 2, 1 (2018), 35–47.

[89] Zhen Wang, Long Zhang, Rong Wang, Feiping Nie, and Xuelong Li. 2023. Semi-Supervised Learning via Bipartite Graph Construction With Adaptive Neighbors. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2023), 5257–5268. https://doi.org/10.1109/TKDE.2022.3151315

[90] Xiao-Ming Wu, Zhenguo Li, Anthony Man-Cho So, John Wright, and Shih-Fu Chang. 2012. Learning with Partially Absorbing Random Walks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, 3077–3085.

[91] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (Jan. 2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[92] Feng Xia, Jiaying Liu, Hansong Nie, Yonghao Fu, Liangtian Wan, and Xiangjie Kong. 2019. Random walks: A review of algorithms and applications. *IEEE Transactions on Emerging Topics in Computational Intelligence* 4, 2 (2019), 95–107.

[93] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=ryGs6iA5Km

[94] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*. PMLR, 5453–5462.

[95] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Proceedings of the 33rd International Conference on Neural Information Processing Systems* 32, Article 135 (2019), 1509-1520 pages.

[96] Han Yang, Xiao Yan, Xinyan Dai, Yongqiang Chen, and James Cheng. 2021. Self-Enhanced GNN: Improving Graph Neural Networks Using Model Outputs. In *International Joint Conference on Neural Networks*.

[97] Liang Yang, Mengzhe Li, Liyang Liu, bingxin niu, Chuan Wang, Xiaochun Cao, and Yuanfang Guo. 2021. Diverse Message Passing for Attribute with Heterophily, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.). *Advances in Neural Information Processing Systems* 34, 4751–4763. https://proceedings.neurips.cc/paper_files/paper/2021/file/253614bbac999b38b5b60cae531c4969-Paper.pdf

[98] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph Convolutional Networks for Text Classification. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 905, 8 pages. https://doi.org/10.1609/aaai.v33i01.33017370

[99] Kun Zhan and Chaoxi Niu. 2021. Mutual Teaching for Graph Convolutional Networks. *Future Generation Computer Systems* 115, 2 (2021), 837–843. https://doi.org/10.1016/j.future.2020.10.016

[100] Chenzi Zhang, Shuguang Hu, Zhihao Gavin Tang, and TH Hubert Chan. 2017. Re-revisiting learning on hypergraphs: confidence interval and subgradient method. In *International Conference on Machine Learning*. PMLR, 4026–4034.

[101] Kai Zhang, James T Kwok, and Bahram Parvin. 2009. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1233–1240.

[102] Ruochi Zhang, Yuesong Zou, and Jian Ma. 2020. Hyper-SAGNN: a self-attention based graph neural network for hypergraphs. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020.* https://openreview.net/forum?id=ryeHuJBtPH

[103] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S. Yu. 2022. Graph Neural Networks for Graphs with Heterophily: A Survey. *ArXiv* abs/2202.07082 (2022). https://api.semanticscholar.org/CorpusID:246863422

[104] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[105] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2003. Learning with Local and Global Consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems* (Whistler, British Columbia, Canada) *(NIPS'03)*. MIT Press, Cambridge, MA, USA, 321–328.

[106] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems* 19 (2006), 1601–1608. https://doi.org/10.7551/mitpress/7503.003.0205

[107] Xueyuan Zhou and Mikhail Belkin. 2011. Semi-supervised learning by higher order regularization. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 892–900.

[108] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2021. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11168–11176.

[109] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *Proceedings of the 34th International Conference on Neural Information Processing Systems* 33, Article 653 (2020), 12 pages.

[110] Xiaojin Zhu. 2005. *Semi-Supervised Learning Literature Survey (accessed 07-08-2023).* Technical Report 1530. Computer Sciences, University of Wisconsin-Madison.

[111] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. *Proceedings of the Twentieth International Conference on International Conference on Machine Learning* 8 (2003), 912–919.

A APPENDIX

A.1 Dataset table

The details about all the datasets are given in Table 8.

A.2 Absorption probability in ParWalk

Given a graph $G(V, E)$, a random walk can be defined as a markov chain over V. The transition probability of a *standard* random walk jumping to a node v from a node u can be given as:

$$p_{uv} = \frac{A(u,v)}{d_u}$$

Thus, the transition probability matrix of a *standard* random walk can be given as:

$$P = D^{-1}A$$

where D is the degree matrix $D = diag(d_1, d_2, ..., d_{|V|})$, d_i be the degree of node v_i and A is the adjacency matrix of graph G. PaRWalk is a variant of the standard random walk. The transition probability of a PaRWalk can be given as [90]:

$$par_{uv} = \begin{cases} \frac{\alpha\lambda_u}{\alpha\lambda_u + d_u} & u = v \\ \frac{A(u,v)}{\alpha\lambda_u + d_u} & u \neq v \end{cases}$$

The absorption probability of a PaRWalk can be given as:

$$a_{uv} = \begin{cases} \frac{\alpha\lambda_u}{\alpha\lambda_u + d_u} + \sum_{x \neq u} \frac{A(u,x)}{\alpha\lambda_u + d_u} a_{xu} & u = v \\ \sum_{k \neq u} \frac{A(u,v)}{\alpha\lambda_u + d_u} a_{kv} & u \neq v \end{cases}$$

Network	nodes	Edges	Features	Classes	Distribution of Nodes in Each Class
Citeseer [71]	3327	4732	3703	6	0: 7%, 1: 18%, 2: 20%, 3: 21%, 4: 18%, 5: 16%
Cora [71]	2708	5429	1433	7	0: 13%, 1: 8%, 2: 15%, 3: 30%, 4: 16%, 5: 11%, 6: 6%
Cora-ml [10]	2995	8416	2879	7	0: 12%, 1: 13%, 2: 15%, 3: 15%, 4: 29%, 5: 6%, 6: 9%
Pubmed [71]	19717	44338	500	3	0: 20%, 1: 40%, 2: 40%
Amazon Photo [72]	7487	119043	745	8	0: 5%, 1: 23%, 2: 9%, 3: 12%, 4: 12%, 5: 10%, 6: 26%, 7: 4%
Amazon Computers [72]	13381	245778	767	10	0: 3%, 1: 16%, 2: 11%, 3: 4%, 4: 39%, 5: 2%, 6: 4%, 7: 6%, 8: 16%, 9: 2%
ogbn-arxiv [85]	169,343	1,166,243	128	40	0: 0.33%, 1: 0.40%, 2: 2.85%, 3: 1.22%, 4: 3.46%, 5: 2.92%, 6: 0.95%, 7: 0.34%, 8: 3.68%, 9: 1.66%, 10: 4.64%, 11: 0.44%, 12: 0.017%, 13: 1.39%, 14: 0.35%, 15: 0.23%, 16: 16%, 17: 0.30%, 18: 0.44%, 19: 1.69%, 20: 1.22%, 21: 0.23%, 22: 1.12%, 23: 1.67%, 24: 13%, 25: 0.74%, 26: 2.71%, 27: 2.83%, 28: 12.64%, 29: 0.24%, 30: 6.97%, 31: 1.67%, 32: 0.24%, 33: 0.75%, 34: 4.64%, 35: 0.07%, 36: 2.08%, 37: 1.39%, 38: 0.88%, 39: 1.19% Can be found in Appendix
ogbn-products [85]	2,449,029	61,859,140	100	47	0: 4.67%, 1: 4.48%, 2: 4.74%, 3: 6.17%, 4: 27.31%, 5: 1.66%, 6: 6.48%, 7: 7.03%, 8: 4.52%, 9: 2.75%, 10: 2.14%, 11: 1.34%, 12: 5.39%, 13: 4.15%, 14: 0.13%, 15: 1.1%, 16: 3.41%, 17: 1.73%, 18: 2.0%, 19: 0.71%, 20: 0.92%, 21: 3.3%, 22: 0.04%, 23: 0.15%, 24: 1.85%, 25: 0.12%, 26: 0.02%, 27: 0.01%, 28: 0.08%, 29: 0.06%, 30: 0.01%, 31: 0.02%, 32: 0.02%, 33: 0.0%, 34: 0.01%, 35: 0.0%, 36: 0.03%, 37: 0.02%, 38: 0.0%, 39: 0.0%, 40: 0.0%, 41: 0.0%, 42: 1.33%, 43: 0.06%, 44: 0.02%, 45: 0.0%, 46: 0.0%
ogbn-mag [85]	1,939,743	21,111,007	128	349	0: 0.3%, 1: 4.2%, 2: 0.03%, 3: 0.06%, 4: 0.03%, 5: 0.96%, 6: 0.15%, 7: 0.6%, 8: 0.03%, 9: 2.02%, 10: 0.04%, 11: 0.1%, 12: 0.19%, 13: 0.3%, 14: 0.07%, 15: 0.03%, 16: 0.12%, 17: 0.12%, 18: 0.52%, 19: 0.11%, 20: 0.18%, 21: 0.12%, 22: 0.34%, 23: 0.16%, 24: 0.04%, 25: 0.22%, 26: 0.03%, 27: 0.05%, 28: 0.3%, 29: 0.17%, 30: 0.03%, 31: 0.42%, 32: 0.12%, 33: 0.67%, 34: 0.35%, 35: 1.32%, 36: 0.03%, 37: 0.29%, 38: 0.62%, 39: 0.03%, 40: 0.04%, 41: 0.2%, 42: 0.11%, 43: 0.17%, 44: 0.16%, 45: 1.04%, 46: 0.07%, 47: 0.04%, 48: 0.76%, 49: 0.09%, 50: 0.11%, 51: 0.56%, 52: 0.88%, 53: 0.06%, 54: 0.12%, 55: 0.05%, 56: 0.1%, 57: 0.1%, 58: 0.24%, 59: 0.08%, 60: 0.03%, 61: 0.14%, 62: 0.06%, 63: 0.07%, 64: 0.1%, 65: 0.05%, 66: 0.11%, 67: 0.04%, 68: 0.26%, 69: 0.07%, 70: 0.2%, 71: 0.06%, 72: 0.7%, 73: 0.03%, 74: 0.04%, 75: 0.36%, 76: 0.07%, 77: 0.67%, 78: 0.2%, 79: 0.09%, 80: 0.16%, 81: 0.15%, 82: 0.19%, 83: 0.45%, 84: 1.36%, 85: 0.3%, 86: 0.33%, 87: 0.05%, 88: 0.03%, 89: 0.26%, 90: 0.09%, 91: 0.12%, 92: 0.2%, 93: 0.1%, 94: 0.06%, 95: 0.2%, 96: 0.42%, 97: 0.16%, 98: 0.04%, 99: 0.29%, 100: 0.22%, 101: 0.09%, 102: 0.05%, 103: 0.11%, 104: 0.05%, 105: 0.16%, 106: 0.64%, 107: 0.2%, 108: 0.13%, 109: 0.21%, 110: 0.36%, 111: 0.04%, 112: 1.24%, 113: 0.24%, 114: 0.28%, 115: 0.74%, 116: 0.46%, 117: 0.06%, 118: 0.09%, 119: 0.07%, 120: 0.05%, 121: 0.05%, 122: 0.26%, 123: 0.36%, 124: 0.03%, 125: 0.03%, 126: 0.44%, 127: 0.12%, 128: 0.27%, 129: 0.09%, 130: 0.24%, 131: 0.15%, 132: 0.63%, 133: 0.04%, 134: 4.17%, 135: 0.08%, 136: 0.21%, 137: 0.05%, 138: 0.36%, 139: 0.41%, 140: 0.33%, 141: 0.46%, 142: 0.06%, 143: 0.03%, 144: 0.78%, 145: 0.38%, 146: 0.38%, 147: 0.07%, 148: 0.12%, 149: 0.15%, 150: 0.03%, 151: 0.03%, 152: 0.08%, 153: 0.04%, 154: 0.51%, 155: 0.09%, 156: 0.12%, 157: 0.1%, 158: 0.03%, 159: 0.12%, 160: 0.37%, 161: 0.23%, 162: 0.08%, 163: 0.13%, 164: 0.09%, 165: 0.66%, 166: 0.39%, 167: 0.29%, 168: 0.09%, 169: 0.2%, 170: 0.74%, 171: 0.22%, 172: 0.37%, 173: 0.04%, 174: 0.03%, 175: 0.03%, 176: 0.03%, 177: 0.08%, 178: 0.1%, 179: 0.06%, 180: 0.32%, 181: 0.37%, 182: 0.23%, 183: 0.28%, 184: 0.19%, 185: 0.23%, 186: 1.05%, 187: 0.07%, 188: 0.28%, 189: 2.11%, 190: 0.09%, 191: 0.21%, 192: 0.36%, 193: 1.7%, 194: 0.05%, 195: 0.03%, 196: 0.16%, 197: 0.11%, 198: 0.11%, 199: 0.15%, 200: 0.2%, 201: 0.2%, 202: 0.08%, 203: 0.04%, 204: 0.23%, 205: 0.13%, 206: 0.03%, 207: 0.24%, 208: 0.05%, 209: 0.07%, 210: 0.05%, 211: 0.11%, 212: 0.03%, 213: 0.05%, 214: 0.89%, 215: 0.03%, 216: 0.03%, 217: 0.06%, 218: 0.18%, 219: 0.88%, 220: 0.06%, 221: 0.82%, 222: 0.06%, 223: 0.24%, 224: 0.12%, 225: 0.04%, 226: 0.26%, 227: 0.05%, 228: 0.2%, 229: 0.2%, 230: 0.04%, 231: 0.03%, 232: 0.64%, 233: 0.6%, 234: 0.1%, 235: 0.23%, 236: 1.17%, 237: 0.08%, 238: 0.47%, 239: 0.05%, 240: 0.22%, 241: 0.09%, 242: 0.06%, 243: 0.16%, 244: 0.07%, 245: 0.27%, 246: 0.05%, 247: 0.5%, 248: 0.27%, 249: 0.51%, 250: 0.13%, 251: 0.13%, 252: 0.24%, 253: 0.28%, 254: 0.06%, 255: 0.09%, 256: 0.37%, 257: 0.09%, 258: 3.37%, 259: 0.12%, 260: 0.05%, 261: 0.17%, 262: 1.23%, 263: 0.03%, 264: 0.04%, 265: 0.37%, 266: 1.53%, 267: 0.22%, 268: 0.09%, 269: 0.03%, 270: 0.04%, 271: 0.05%, 272: 0.06%, 273: 0.08%, 274: 0.04%, 275: 0.04%, 276: 0.28%, 277: 1.02%, 278: 0.19%, 279: 0.08%, 280: 0.09%, 281: 2.53%, 282: 0.03%, 283: 3.32%, 284: 0.07%, 285: 0.05%, 286: 0.04%, 287: 0.04%, 288: 0.04%, 289: 0.79%, 290: 0.11%, 291: 0.56%, 292: 0.07%, 293: 0.24%, 294: 0.21%, 295: 0.13%, 296: 0.3%, 297: 0.1%, 298: 0.03%, 299: 0.15%, 300: 3.74%, 301: 0.04%, 302: 0.45%, 303: 0.05%, 304: 0.86%, 305: 0.15%, 306: 0.09%, 307: 0.09%, 308: 0.13%, 309: 0.11%, 310: 0.06%, 311: 2.29%, 312: 0.43%, 313: 0.07%, 314: 0.21%, 315: 0.13%, 316: 0.03%, 317: 0.09%, 318: 0.14%, 319: 0.05%, 320: 0.04%, 321: 0.03%, 322: 0.26%, 323: 0.08%, 324: 0.27%, 325: 0.04%, 326: 0.04%, 327: 0.03%, 328: 0.03%, 329: 0.03%, 330: 0.04%, 331: 0.03%, 332: 0.04%, 333: 0.09%, 334: 0.04%, 335: 0.03%, 336: 0.05%, 337: 0.04%, 338: 0.06%, 339: 0.04%, 340: 0.05%, 341: 0.03%, 342: 0.06%, 343: 0.04%, 344: 0.03%, 345: 0.04%, 346: 0.04%, 347: 0.06%, 348: 0.04%

Table 8. The test suite of real-world networks.

In matrix form, the transition probability matrix of PaRWalk can be written as:

$$\mathcal{A} = (\Gamma + \alpha\Lambda)^{-1}$$

where, Γ be the graph laplacian defined as $\Gamma = D - A$, $\alpha > 0$ is a scalar value, $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_{|V|})$ is known to be as regularizer, $\lambda_i \geq 0$ be some arbitrary value.

A.3 Detail time complexity analysis of Algorithm 1

Time requirement of Algorithm 1: Total time taken is time taken by using (i) + for each class, time taken using (ii) and (iii). Therefore, for $|L|$ number of class labels, total time taken = $O(|V|^{2.373}) + |L|\ (O(t\delta mj) + O(|C_{avg}|m))$ = $O(|V|^{2.373}) + (O(|L|t\delta mj) + O(|L||C_{avg}|m))$ = $O(|V|^{2.373}) + (O(|L|\frac{|I_c|b\eta}{\sum_{c=1}^{k}|I_c|}\delta mj) + O(|LC_{avg}|m))$, since, $t = \frac{|I_c|b\eta}{\sum_{c=1}^{k}|I_c|}$ (from discussion in section 3.1) = $O(|V|^{2.373}) + (O(|L|\frac{|I_c|b\eta}{2|L|}\delta mj) + O(|L||C_{avg}|m))$ as, $\sum_{c=1}^{k}|I_c| = 2L$, since we choose two nodes per class. = $O(|V|^{2.373}) + (O(\frac{|I_c|b\eta}{2}\delta mj) + O(|L||C_{avg}|m))$ = $O(|V|^{2.373}) + (O(\frac{2b\eta}{2}\delta mj) + O(|L||C_{avg}|m))$ = $O(|V|^{2.373}) + (O(b\eta\delta mj) + O(|L||C_{avg}|m))$ = $O(|V|^{2.373}) + (O(\frac{|V|b\delta mj}{(d_{avg})^{\tau}}) + O(|L||C_{avg}|m))$, d_{avg} is the average degree of the graph G and τ is the number of layers of the GNN. = $O(|V|^{2.373}) + (O(\frac{|V|mj}{(d_{avg})^{\tau}}) + O(|L||C_{avg}|m))$, for $b = 3$, $\delta = 2$ are constant. = $O(|V|^{2.373}) + (O(\frac{|V|mj}{(d_{avg})^{\tau}}) + O(|L||C_{avg}|m))$, for $b = 3$, $\delta = 2$ are constant.

A.4 Detail time complexity analysis of Algorithm 3

Time requirement of Algorithm 3: line 2 needs O($|V|$) time + Looking at line 3 − 20, for each class, line 4 and 5 both needs O(t) time + line 6 − 19 requires O($t(|H_{avg}| + d_{avg})$) time (considering average degree as d_{avg} and the average number of high confidence nodes as $|H_{avg}|$). So, total time required is (using PaRWalk as the random walk): $O(|V|^{2.373}) + O(|V| + |L|(t + t(|H_{avg}| + d_{avg}))) = O(|V|^{2.373}) + O(|V| + |L|t(1 + |H_{avg}| + d_{avg}))$. Since we have taken $t = \frac{|V|}{|L|}$, thus, time complexity of Algorithm 3 will be: $O(|V|^{2.373}) + O(|V| + |L|\frac{|V|}{|L|}(1 + |H_{avg}| + d_{avg})) = O(|V|^{2.373}) + O(|V| + |V|(1 + |H_{avg}| + d_{avg}))$.