GraphHop: An Enhanced Label Propagation Method for Node Classification

Tian Xie, Graduate Student Member, IEEE, Bin Wang, Member, IEEE, and C.-C. Jay Kuo, Fellow, IEEE

Abstract—A scalable semisupervised node classification method on graph-structured data, called GraphHop, is proposed in this work. The graph contains all nodes’ attributes and link connections but labels of only a subset of nodes. Graph convolutional networks (GCNs) have provided superior performance in node label classification over the traditional label propagation (LP) methods for this problem. Nevertheless, current GCN algorithms suffer from a considerable amount of labels for training because of high model complexity or cannot be easily generalized to large-scale graphs due to the expensive cost of loading the entire graph and node embeddings. Besides, nonlinearity makes the optimization process a mystery. To this end, an enhanced LP method, called GraphHop, is proposed to tackle these problems. GraphHop can be viewed as a smoothing LP algorithm, in which each propagation alternates between two steps: label aggregation and label update. In the label aggregation step, multihop neighbor embeddings are aggregated to the center node. In the label update step, new embeddings are learned and predicted for each node based on aggregated results from the previous step. The two-step iteration improves the graph signal smoothening capacity. Furthermore, to encode attributes, links, and labels on graphs effectively under one framework, we adopt a two-stage training process, i.e., the initialization stage and the iteration stage. Thus, the smooth attribute information extracted from the initialization stage is consistently imposed in the propagation process in the iteration stage. Experimental results show that GraphHop outperforms state-of-the-art graph learning methods on a wide range of tasks in graphs of various sizes (e.g., multilabel and multiclass classification on citation networks, social graphs, and commodity consumption graphs).

Index Terms—Graph convolutional networks (GCNs), graph learning, label propagation (LP), large-scale graphs, semisupervised learning.

I. INTRODUCTION

The success of deep learning and neural networks [1] often comes at the price of a large number of labeled data. Semisupervised learning is an important paradigm that leverages a large number of unlabeled data to address this limitation. The need for semisupervised learning has arisen in many machine learning problems and found wide applications in computer vision, natural language processing, and graph-based modeling, where getting labeled data are expensive, and there exists a large amount of unlabeled data.

Many efforts have been made in applying neural networks to the semisupervised node classification problem for graph-structured data in recent years. For example, the pioneering graph convolutional network (GCN) proposed in [2] achieves state-of-the-art node classification performance to citation networks. GCNs conduct layer embeddings propagation, starting from the node attributes, through a graph with its node labels as supervision. Although GCNs [3]–[6] offer impressive results, they still have several drawbacks that limit the capacity. First, the large number of parameters and nonlinearity in the GCN model demand a considerable number of labeled samples in model training. Li et al. [7] attempted to alleviate this problem by introducing co-training or self-training. However, it is still restricted to the GCN framework without significant change. Second, GCNs fail to exploit the label dependence concerning the graph structure in model learning. Some methods [8]–[10] tried to encode label dependence through a generative model. However, a particular inductive bias is assumed, and complicated optimization algorithms are needed. Third, the end-to-end training through gradient backpropagation makes GCNs difficult to scale for large graphs [3], [11]. Node embeddings of the entire graph have to be stored at all intermediate layers. Besides, the rapid expansion of the neighborhood size in deeper layers makes the minibatch training challenging [12]. There is a tradeoff between training efficiency and a larger receptive field size [13]. Finally, nonlinear activation hinders the understanding of GCNs, and the superior performance of GCNs remains to be a mystery. Some researchers [7], [13], [14] attempted to interpret GCNs by dropping the nonlinear terms, and as a result, a low-pass filter becomes the means in propagating embeddings. It is still not sufficient to explain the whole GCN algorithm.

Prior to the development of GCNs, another line of research based on label propagation (LP) [15]–[17] demonstrated great adaptability [18]–[20], scalability [21], [22], and efficiency [23], [24] for the semisupervised node classification problem. In particular, LP exploits the geometry of data entities induced by labeled and unlabeled samples. With fewer labeled nodes, LP iteratively aggregates label embeddings from neighbors and propagates them throughout the graph to provide labels for all nodes. This iterative process can be viewed as low-pass filtering, where smooth graph signals are extracted given the noisy input [14], [25], [26]. Although LP methods can address the pitfalls in GCNs, their performance is inferior to that of GCN methods against several common benchmarking datasets (e.g., citation datasets).

There could be three reasons to explain the poorer performance of LP methods. First, both attribute smoothening and label smoothening should be considered jointly in model

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
learning. Typically, the propagation (i.e., smoothening operation) is only conducted on label embeddings, while node attributes are ignored. Although some work [18], [19], [27] incorporated the attribute information by introducing them as an extra regularization term under the regularization framework, the effect through optimization is somehow indirect. Still, smoothed attribute information w.r.t. the graph structure is failed to be examined. Second, the propagation of labels in LP methods is typically implemented by a simple lookup table for unlabeled nodes [15], [16], [28]. Since label embedding parameters are not shared between nodes along iterations, it tends to have a slow convergence rate with inferior performance due to the limited regularization effect between label embeddings. Finally, only the information of one-hop neighbors is propagated at each iteration. Embeddings from multihop neighbors are not exploited [29].

In this article, we propose an enhanced LP scheme, called GraphHop, to tackle those weaknesses in LP. In particular, we treat attributes and labels as two distinct but correlated signals, where correct labels can be predicted from the smoothed attributes. They are assumed to be locally smooth on graphs. GraphHop integrates these two signal types with a two-stage training framework. That is, it conducts attributes smoothening in the initialization stage and label smoothening in the iteration stage as sketched in the following.

In the initialization stage, smoothed node attributes and labeled samples are used to train a regression classifier that predicts each node’s label embedding. The dimension of the label embedding vector is the same as the class number. Its element indicates the probability of belonging to a class. The label embeddings serve as the starting point of LP in the iteration stage.

In the iteration stage, one iteration consists of the following two steps.

1) **Label Aggregation:** For each node, a simple average is first taken for the iterative label embeddings of its neighborhood nodes with the same hop number. Then, it concatenates these averages derived from multihop neighbors and itself, which serves as the input to the subsequent label update step.

2) **Label Update:** Regression classifiers are used to smoothen label embeddings. Different classifiers are assigned and trained independently for multihop of aggregated representations so that different hops of information can be processed independently. The new label embeddings are calculated by averaging predictions from multiple classifiers so that the embedding parameters are shared between nodes. Since this iterative procedure exploits multihop neighbor information together, we name the proposed method “GraphHop.”

We conduct extensive experiments on various scales of benchmarking datasets with particular small label rate settings. The contributions of this work can be summarized in the following.

1) We propose an enhanced LP-based method, called GraphHop, which conducts joint attribute and label smoothening on graphs through regression classifiers.

2) We provide theoretical justification with some approximation to the fact that GraphHop converges faster than the traditional LP.

3) We empirically verify that the collaborative model design can address the three weaknesses of the traditional LP.

4) We conduct extensive experiments to validate GraphHop’s effectiveness concerning small label rates and large-scale graphs against state-of-the-art GCN algorithms. To the best of our knowledge, we are the first to show that an enhanced LP-based method can outperform GCN baselines on several well-known benchmarking datasets. Our code is publicly available at https://github.com/TianXieUSC/GraphHop.

The rest of this article is organized as follows. Some preliminaries are introduced in Section II. The GraphHop method is presented in Section III. The theoretical analysis is conducted to estimate the required iteration number of the iterative algorithm in Section IV. Extensive experiments are conducted to show the state-of-the-art performance with low memory usage in Section V. Comments on related work, applications and further improvements are made in Sections VI and VII, respectively. Finally, concluding remarks are given, and future research directions are pointed out in Section VIII.

II. PRELIMINARIES

In this section, we first formulate the transductive semisupervised learning problem on graphs in Section II-A. Then, we present the smoothness assumption leading to our model design, which holds for most graph learning methods and provides empirical evidence on the benchmarking datasets in Section II-B. Finally, we review three commonly used graph signal processing tools in Section II-C.

A. Problem Statement

An undirected graph can be represented by a triple
\[ \mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{X}) \]
where \( \mathcal{V} \) denotes a set of \( n \) nodes, \( \mathcal{A} \in \mathbb{R}^{n \times n} \) is the adjacency matrix between nodes, and \( \mathcal{X} \in \mathbb{R}^{n \times d} \) is the attribute matrix whose \( d \)-dimensional row vector is an attribute vector associated with each node. In the setting of semisupervised classification, each labeled node belongs to one class
\[ y \in \mathcal{C}, \quad \text{where } \mathcal{C} = \{1, \ldots, c\}. \]
Nodes are divided into labeled and unlabeled sets with their indices denoted by
\[ \mathcal{L} = \{1, \ldots, l\} \quad \text{and } \mathcal{U} = \{l + 1, \ldots, n\} \]
respectively.

Let \( \mathcal{H} \) be the set of \( n \times c \) matrices with nonnegative entries. Matrix
\[ \mathbf{H} \in \mathcal{H} = (h^1, \ldots, h^l, h^{l+1}, \ldots, h^n)^T \]
is the global label embedding matrix for each node. It can be divided into
\[ \mathbf{H} = (\mathbf{H}^l, \mathbf{H}^u)^T \]
where \( H_l \) and \( H_u \) represent labeled and unlabeled samples, respectively. For the \( i \)th row of \( H \), vector \( h \) is a probability vector satisfying \( \sum_j h_{ij} = 1 \), whose element \( h_{ij} \) is the probability of belonging to class \( j \). To classify unlabeled node \( i \) in the convergence stage, we assign it the label of the most likely class

\[
y_i = \arg\max_{j \in c} h_{ij}. \tag{1}
\]

The goal of transductive graph learning is to infer labels of unlabeled nodes (i.e., the matrix \( H_u \)), given attribute matrix \( X \), adjacency matrix \( A \), and labeled samples. In the extremely small label rate case, the number of samples in the labeled set is much smaller than that in the unlabeled set, namely,

\[|c| \ll |u|.
\]

### B. Assumption

The attribute matrix and the label embedding matrix can be treated as signals on graphs. For example, the \( d \)-dimensional column vectors of attribute matrix \( X \) can be viewed as a \( d \)-channel graph signal. Semi-supervised learning assumes that data points close in a high-density region should have similar attributes and produce similar outputs. This property can be stated qualitatively in the following.

**Assumption 1:** The node attribute and label signals are smooth functions on graphs, where correct labels can be predicted from the smoothoothed attributes.

The smoothness of attributes and labels can be verified by evaluating their means and variances as a function of the length of the shortest path between every two nodes, which is usually measured in terms of the hop count. Here, we propose an alternative approach to verify the smooth label assumption (the study of smooth attribute assumption on the benchmarking datasets can be found in [13], [14], and [30]) and conduct experiments on the Cora, CiteSeer, and PubMed datasets (see Section V for more details) using the following procedure.

**Step 1:** Compute the graph Fourier basis

\[
Q = [q_1, q_2, \ldots, q_n]
\]

and its corresponding frequency matrix

\[
\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)
\]

from graph Laplacian \( L = D - A \), where \( D \) is a diagonal matrix of vertex degrees defined as \( D_{ii} = \sum_{j=1}^n A_{ij} \). Specifically, the frequencies in \( \Lambda \) are arranged in an ascending order

\[\lambda_1 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n.\]

Similarly, we can define the reverse-ordered frequency matrix

\[\Lambda_r = \text{diag}(\lambda_n, \lambda_{n-1}, \ldots, \lambda_1)\]

and find the corresponding basis matrix

\[
Q_r = [q_n, q_{n-1}, \ldots, q_1].
\]

It is well known that the basis signals of the graph Laplacian associated with lower frequencies (i.e., smaller eigenvalues) are smoother on the graph [31].

**Step 2:** Compute the top-\( k \) lowest or highest frequency components of labels via

\[
\tilde{Y} = Q[:, k]^T Y \quad \text{or} \quad \tilde{Y}_r = Q_r[:, k]^T Y
\]

where each row of \( Y \in \mathbb{R}^{n \times c} \) denotes the one-hot embedding of a node label. Then, we reconstruct labels for all nodes via

\[
\hat{Y} = Q[:, k] \tilde{Y} \quad \text{or} \quad \hat{Y}_r = Q_r[:, k] \tilde{Y}_r
\]

and make a prediction based on the reconstructed label embedding matrix, where classes of node \( i \) are decided by (1). The above computation yields two classification results for each node: one using smooth components (\( \hat{Y} \)) while the other using highly fluctuating components (\( \hat{Y}_r \)).

We incrementally add the number of Laplacian frequency components, reconstruct label embeddings for classification, and show two accuracy curves for each dataset in Fig. 1. The one using low-frequency components is in blue, and the one using high-frequency components is in red. The blue curve rises quickly, while the red curve increases slowly in all three cases, supporting our assumption that the graphs’ label signals are smooth. Interestingly, the performance does drop when adding more high-frequency components for PubMed.

### C. Smoothening Operations

Based on Assumption 1, we expect that a smoothening operation on node attributes and labels helps achieve better classification results. There are several common ways to achieve the smoothening effect, which can be categorized into the following three types.

1) **Average Attributes or Label Embeddings of Neighboring Nodes:** It is well known that the average operation behaves like a low-pass filter [13]. Another reason for
taking the average is that nodes have different numbers of \( m \)-hop neighbors, denoted by \( \Omega_m \), where \( m = 1, 2, \ldots \). By averaging the attributes/labels of nodes with the same hop distance, we can consider the impact of \( m \)-hop neighbors for all nodes uniformly.

2) Use Regression for Label Prediction: We need to predict labels for a great majority of nodes based on attributes of all nodes and labels of a few nodes. One way for prediction is to train a regressor. Regression (regardless of a linear or a logistic regressor) is fundamentally a smoothing regularization operation since it adopts a fixed but smaller model size to fit a large number of observations.

3) Incorporate a Smooth Regularization Term in an Objective Function: One can introduce a quadratic penalty term and, at the same time, remain consistent with the initial labeling. The results can be equivalently derived from iterations in LP [15], [17] as elaborated in the following.

The smoothing regularization framework can be formally defined as a minimization of the objective function

\[
J(H) = \frac{||H - Y||_F^2}{\text{Least square penalty}} + \frac{\mu \text{Tr}(H^\top \tilde{L}H)}{\text{Laplacian regularization}}
\]

where \( Y = (Y_1^T, \ldots, Y_n^T)^T \) consists of the one-hot label embedding matrix of labeled nodes \( Y_i \) and zeros of unlabeled nodes \( Y_u \), \( || \cdot ||_F \) is the Frobenius norm, \( \mu \) is a parameter controlling the degree of the regularization, and \( \tilde{L} \) is the symmetric normalized graph Laplacian matrix. The optimization solution of embedding matrix \( H \) in (2) can be derived equivalently by applying an embedding propagation process. The update at the \( t \)th iteration is derived as

\[
H^{(t+1)} = a\tilde{A}H^{(t)} + (1 - a)H^{(0)}
\]

where \( \tilde{A} = D^{-(1/2)}AD^{-(1/2)} \) is the symmetric normalized adjacency matrix, and \( H^{(0)} = Y_f \) for labeled samples and zeros for unlabeled samples. The update rule can be described by replacing the current label embeddings with their averaged neighbors’ through a lookup table plus the initially labeled samples. As shown in (2) and (3), we see that the LP method exploits smoothing operations of the first and third types, respectively. A closed-form solution to (2) and (3) can be equivalently cast in form of

\[
H = (1 - a)(I - a\tilde{A})^{-1}H^{(0)}.
\]

Based on the propagation process in (3), we observe several limitations that restrict the learning capability for embeddings propagation and update.

1) The node attributes, especially the smoothed node attributes, are not exploited.

2) Label embeddings are not shared between nodes so that their correlations are not considered.

3) Only label embeddings of one-hop neighbors are propagated at each iteration. Therefore, the multihop relationships are omitted in model learning.

III. GRAPHHOP METHOD

The GraphHop method is introduced in this section to target the deficiencies mentioned above. It is an iterative algorithm that consists of an initialization stage and an iteration stage, as shown in Fig. 2. The initialization stage predicts the initial label embeddings of all nodes based on the smoothed node attributes and labeled nodes via regression. The predicted label embeddings then serve as the starting point for the subsequent iterations. The iteration stage conducts label aggregation and updates to smoothen the label signals. The attribute information is not needed in the iteration stage.

GraphHop has the following features to address the limitations of the traditional LP methods, as described in Section II.

1) The smoothed attributes are extracted and used to train regressors in the initialization stage, where the predicted label embeddings are further smoothed in the subsequent iterations.

2) Another set of regression classifiers is trained on entire nodes to smoothen the label embeddings, which plays the role of embedding parameter sharing.

3) Mixing multihop neighbors are served as input for each regression classifier independently. The predictions from different hop neighbors are then aggregated to update the label embedding of each node.
A. Initialization Stage

We attempt to smoothen the node attributes input and provide initial label embeddings to all nodes in the initialization stage. Therefore, we design the first smoothening operation and apply it to the attribute signals. Mathematically, for node \( j \) and its \( m \)-th hop neighbors, we have the following averaged attribute representations:

\[
x_{j,m} = \frac{\sum_{i \in \Omega_m(j)} x_i}{|\Omega_m(j)|}, \quad m = 0, 1, \ldots, M
\]

where \( \Omega_m(j) \) denotes the \( m \)-hop neighbors of node \( j \) and \( m = 0 \) indicates the node \( j \) itself. The averaged attribute vectors from different hops (i.e., different \( m \)'s) are concatenated, which yields a higher order neighborhood representation for \( v \) from different hops (i.e., different \( m \)).

\[
X_M = \|0 \leq m \leq M \tilde{A}^m X
\]

where \( \| \) denotes columnwise concatenation, \( \tilde{A}^m \) is the normalized \( m \)-hop adjacency matrix, and \( X_M \in \mathbb{R}^{n \times d(M+1)} \) is the smoothed attribute matrix. A larger \( M \) value generates a stronger signal smoothening results [13], [14]. To derive the initial label embeddings, we leverage a logistic regression (LR) classifier trained on this smoothed attribute matrix with labeled samples as supervision. The objective function can be written as

\[
L = \frac{1}{|\mathcal{E}|} \sum_{y \in \mathcal{E}, x_M \in X_M} H(y, p_{\text{model}}(y|x_M; \theta))
\]

where \( H(p, q) \) is the cross-entropy loss, \( p_{\text{model}} \) is the LR classifier, and \( \theta \) is the set of model parameters. The combination of attributes from various distances (i.e., different \( M \)) possess distinct relationships and should be considered independently [32], [33]. To encode multihop correlations, a unique classifier is adopted for different hops of aggregations (see Fig. 2) to effectively embed multihop attribute information. Afterward, the label embeddings for all nodes are updated by an average from the predictions of all the converged LR classifiers. Formally, this can be written as

\[
H^{(0)} = \frac{1}{K} \sum_{M=1}^{K} p_{\text{model}}(Y|X_M; \theta)
\]

where \( K \) is the number of classifiers trained on different hops of aggregation and \( Y \) is the distribution over class labels. The intuition to leverage the classifier predictions as the initial label embeddings is justified from Assumption 1. That is, nodes of spatial proximity should have similar attributes and produce similar labels.

B. Iteration Stage

The label embeddings predicted in the initialization stage are further processed in the iteration stage. Each iteration contains two steps: label aggregation and label update to smoothen the initial label embeddings. They are introduced as follows.

1) Label Aggregation: The label aggregation step is the same as the attribute aggregation mechanism, as stated in Section III-A, except that it is performed on the iterative label embeddings rather than node attributes. By modifying (6) slightly, we have the following label aggregation formula:

\[
H^{(t-1)}_M = \|0 \leq m \leq M \tilde{A}^m H^{(t-1)}_M, \quad t = 1, 2, \ldots
\]

where \( t \) is the iteration index and \( H^{(t-1)}_M \in \mathbb{R}^{n \times c(M+1)} \) is the smoothed label embedding matrix. The hop number \( M \) controls the size of neighborhoods. A larger \( M \) value results in larger memory and computational complexity. However, even with one-hop connections (i.e., \( M = 1 \)), strong smoothening results can be achieved due to the iteration procedure. This is not available in the model of [13] [see (16)]. In the experiments, we focus on the case with \( M = 2 \), as shown in Fig. 2. So far, the label embedding parameters are not shared between nodes, where each node embedding is unique and derived independently of their neighborhoods. To incorporate the correlations between nodes, we propose to adopt regression classifiers as regularization between nodes and their aggregated results in the label update step so that new label embeddings are generated globally based on entire nodes.

2) Label Update: We perform smoothening and regularization in the label update step on the aggregated label embeddings. An individual LR classifier is adopted for different hops of aggregations (see Fig. 2). Thus, multihop label representations are processed independently. However, being different from the initialization stage where classifiers are only trained by labeled samples, each unlabeled node is now associated with a label embedding (i.e., \( H^{(t-1)}_M \)) from the previous iteration. This embedding encodes the confidence of the current predictions. It can serve as a pseudolabel and contribute to classifier training. The implementation details of the LR classifier in the iteration step are elaborated in the Appendix. In short, the label embeddings of unlabeled nodes are also used as supervision and added to the objective function [see (21)]. Once all classifiers converge, new label embeddings are averaged from the predictions and used as the input to the next iteration. Formally, this can be expressed as

\[
H^{(t)} = \frac{1}{K} \sum_{M=1}^{K} p_{\text{model}}(\mathbf{Y}|H^{(t-1)}_M; \theta)
\]
where \( K \) is the number of classifiers applied for \( M \) hop aggregations. Note that there are other possible choices for the predictor in (8) and (10), as discussed in [34]–[36]. The current LR classifier is chosen due to its efficiency, where the minibatch training can be easily conducted (see Algorithm 1), and its small model size with the probability output.

Finally, the GraphHop method is summarized by pseudocodes in Algorithm 1.

**Algorithm 1 GraphHop**

1. **Input**: Graph \( A \), attributes \( X \), label vectors \( Y_i \)
2. **Output**: Label vectors \( H_u \)
3. **Initialization**:
   4. \( X_M \leftarrow \text{calculate Eq. (6)} \)
   5. while not converged do
      6. for each minibatch do
         7. Compute \( g \leftarrow \nabla L(X_M, Y; \theta) \) in Eq. (7)
         8. Conduct Adam update using gradient estimator \( g \)
      9. end for
   10. end while
11. \( H^{(0)} \leftarrow \text{calculate Eq. (8)} \)
12. **Iteration**:
   13. for iteration \( t \in [1, \ldots, \text{max}_\text{iter}] \) do
      14. \( H^{(t-1)}_M \leftarrow \text{calculate Eq. (9)} \) label aggregation
      15. while not converged do
         16. for each minibatch do
            17. Compute \( g \leftarrow \nabla L(H^{(t-1)}_M, Y, \theta) \) in Eq. (21)
            18. Conduct Adam update using gradient estimator \( g \)
         19. end for
      20. end while
   21. \( H^{(t)} \leftarrow \text{calculate Eq. (10)} \) label update
22. end for

**IV. ANALYSIS**

**A. Convergence Analysis**

In the iteration stage, we analyze the relationship between the lower bound of iteration number, label rate, and the hops number. The higher order (larger hops) feature of nodes can be analyzed in its lower bound of iteration number, label rate, and the hops number. The higher order feature of nodes can be analyzed in its lower bound of iteration number, label rate, and the hops number.

**Lemma 1**: Given the maximum hop \( k = \max[M] \) covered in each LP step, the node predictions in \( V_t \) will be sufficient until the propagation of iteration \( i/k \) is finished.

**Proof**: Since \( |L| \ll |U| \) in semisupervised learning, we ignore the difference between the number of unlabeled nodes (i.e., \(|U|\)) and all nodes in the graph (i.e., \(|L| + |U|\)). According to Lemma 1, after \( t \) iterations, the nodes in subsets \( V_1, V_2, \ldots, V_t \) are sufficient. Then, we have

\[
|V_1 \cup V_2 \cup \cdots \cup V_t| = |V_1| + |V_2| + \cdots + |V_t| 
\leq jd + jd^2 + \cdots + jd^kt 
= jd^{kt+1} - 1 
d - 1. 
\]  

The first equality is because \( V_1, V_2, \ldots, V_t \) are mutually disjoint. Each node has only one unique minimum distance to labeled set \( V_0 \) so that they can only be assigned to one specific subset. The inequality is due to the use of the maximum degree \( d \) for every node. Apparently, \(|V_1 \cup V_2 \cup \cdots \cup V_t| \leq n \). Thus, we get

\[
|V_1 \cup V_2 \cup \cdots \cup V_t| \leq \min\left(n, jd^kt+1 - 1 \right) 
\]  

and the theorem is proven.

It is easy to get the following corollary.

**Corollary 1**: The predictions of all unlabeled nodes on graph \( G \) will be sufficient with \( t \) iterations, where

\[
t \in \Omega\left(\frac{1}{d} \log_d \left(1 + \frac{n(d-1)}{jd}\right)\right). 
\]  

**Proof**: According to Theorem 1, at most \( \min(n, jd((d^kt+1 - 1)/(d-1))) \) nodes are sufficient after \( t \) iterations. To ensure that all nodes on graph \( G \) are sufficient, we let

\[
jd^kt+1 - 1 \geq n 
d - 1 \]

so that

\[
t \geq \frac{1}{d} \log_d \left(1 + \frac{n(d-1)}{jd}\right). 
\]  

**\Box**

Corollary 1 shows that the relationship between sufficient iterations and the maximum hop number \( k \) is in inverse ratio. Increasing \( k \) will decrease the required number of iterations. The initial label rate \( j \) and the graph density \( d \) also influence the iteration number. Nevertheless, the effects are minor due to the logarithmic function. Notably, in a large-scale graph where \( j \ll n \), changing the label rate has negligible influence on the iteration number. The same behavior has been shown in Section V. In practice, we observe that GraphHop converges efficiently.
in a few iterations (usually 10) since few iterations are required to achieve sufficiency.

B. Complexity Analysis

The time and memory complexities of GraphHop are significantly lower than those of GCNs for the following reasons. First, only one set of node embeddings in GraphHop, while there are multiple layer embeddings in GCNs. Second, the minibatch training is straightforward in GraphHop, which is difficult for GCNs due to the neighbor expansion problem [12]. Note that LR classifiers can be directly trained using the aggregation in (9) (resp. (6)). Thus, minibatches can be easily applied to matrix $H$ aggregation in (9) (resp. (6)). Thus, minibatches can be easily applied to matrix $H^{(r-1)}$ (resp. $X_M$) (see lines 6 and 16 in Algorithm 1).

Suppose that, during training, the number of minibatches is $N$, the number of nodes is $n$, the number of iterations is $t$, the size of one minibatch is $b$, and the number of classes is $c$. Then, the time complexity of one minibatch propagation can be computed as

$$O\left(n\left\|\tilde{A}^M\right\|_0 + t\left\|\tilde{A}^M\right\|_0c + tbc^2\right)$$

where the first term is from the computation of multihop neighbors, the second term comes from label aggregation, and the third comes from label update. Note that we can eliminate the first term by considering the one-hop neighbors (i.e., $M = 1$) only. The memory usage complexity is

$$O(bc + c^2)$$

which represents embeddings of one minibatch and parameters for the classifiers. We ignore the storage of the adjacency matrix since it is the same for all algorithms. Note that the memory cost is fixed and independent of iterations $t$ and scales linearly in terms of the minibatch size $b$.

V. EXPERIMENTS

We conduct experiments to evaluate the performance of GraphHop with multiple datasets and tasks. Datasets used in the experiments are described in Section V-A. Experimental settings are discussed in Section V-B. Then, the performance of GraphHop is compared with state-of-the-art methods in small- and large-scale graphs in Section V-C. Finally, ablation studies are given in Section V-E.

A. Datasets

We evaluate the performance of GraphHop on six representative graph datasets, as shown in Table I. Cora, CiteSeer, and PubMed [2] are three citation networks. Nodes are papers, while edges are citation links in these graphs. The task is to predict the category of each paper. PPI and Reddit [3] are two datasets of large-scale networks. PPI is a multilabel dataset, where each node denotes one protein with multiple labels in the gene ontology sets (121 in total). Amazon2M [12] is by far the largest graph dataset that is publicly available with over two million nodes and 61 million edges obtained from Amazon copurchasing networks. The raw node features are bag-of-words extracted from product descriptions. We use the principal component analysis (PCA) [38] to reduce their dimension to 100. Also, we use the top-level category as the class label for each node.

| Dataset | Vertices | Edges | Classes | Features Dims |
|---------|----------|-------|---------|---------------|
| Cora    | 2,708    | 5,429 | 7       | 1,433         |
| CiteSeer| 3,327    | 4,732 | 6       | 3,703         |
| PubMed  | 19,717   | 44,338| 3       | 500           |
| PPI     | 56,944   | 1,612,348 | 121  | 50            |
| Reddit  | 231,443  | 11,606,919 | 41   | 602           |
| Amazon2M| 2,449,029| 61,859,140 | 47   | 100           |

| Dataset | Data splits | Train | Validation |
|---------|-------------|-------|------------|
| PPI     | 563(1%)    | 1138(2%)| 2847(5%) | 5634(10%) | 5000    |
| Reddit  | 2296(1%)   | 4592(2%)| 11562(5%)| 22888(10%)| 5000    |
| Amazon2M| 19606(1%) | 37259(2%)| 77700(4%)| 184856(10%)| 50000   |

B. Experimental Settings

We evaluate GraphHop and several benchmarking methods on the semisupervised node classification task in a transductive setting at several small-label rates. For citation datasets, we first conduct experiments by following the conventional train/validation/test split (i.e., 20 labels per class) of the training set. Next, we train models at meager label rates (i.e., one, two, four, eight, and 16 labeled samples per class). For the three large-scale networks, PPI, Reddit, and Amazon2M, the original data split target inductive learning scenarios, which do not fit our purpose. To tailor them to the transductive semisupervised setting, we adopt fewer labeled training samples. Specifically, for Reddit and Amazon2M, we randomly pick the same number of samples in each class with multiple label rates for training. For the multilabel PPI dataset, we simply select a small portion of samples randomly in training. The fixed size of the remaining samples is selected as validation, while the rest is used for testing. The complete data splits are summarized in Table II. For simplicity, we use the percentages of training samples to indicate different data split in reporting performance results.

We implement GraphHop in PyTorch [39]. For the LR classifiers in the initialization stage and the iteration stage, we use the same Adam optimizer with a learning rate of 0.01 and $5 \times 10^{-5}$ weight decay. The minibatch size is fixed to 512 for citation networks but adaptive for large-scale graphs since experiments show a tradeoff between efficiency and memory cost using different minibatch sizes for the latter case. The training epochs are set to 1000 with an early stopping criterion, which stops the classifiers from training and goes to the next iteration. We set the maximum iteration to 100 for citation datasets and 200 for large-scale networks. As shown in the experiments, these numbers are large enough for GraphHop to converge. For hyperparameters $T$, $\alpha$, and $\beta$, we perform a grid search based on the validation results in the parameter space. The hyperparameter tuning ranges and
their final values are listed in Tables III and IV, respectively. Note that hyperparameters are tuned for different label rates. We show their values for the largest label rates in Table IV. All experiments were conducted on a machine with an NVIDIA Tesla P100 GPU (16-GB memory), a ten-core Intel Xeon CPU (2.40 GHz), and 100 GB of RAM.

C. Performance Evaluation

We conduct performance benchmarking between GraphHop and several state-of-the-art methods and compare their results for small- and large-scale datasets in the following.

1) Citation Networks: The state-of-the-art methods used for performance benchmarking are grouped into three categories.

- 1) LP-Based Methods: LP [15], LNP [40], Special LP [41], Centered Kernel [42], WNLL [43], and Poisson [44].
- 2) Unsupervised Methods: DeepWalk [45], LINE [46], DGI [47], and Graph2Gauss [48].
- 3) Semisupervised Methods: GCN [2], GAT [4], co-training GCN, and self-training GCN [7].

Results on three citation datasets are summarized in Table V, where label rates are chosen to be one, two, four, eight, 16, and 20 labeled nodes per class. Each column shows the classification accuracy (%) for GraphHop and nine benchmarking methods under a dataset and a given label rate.

Overall, GraphHop performs the best, especially for cases with extremely small label rates. The reason is that its adoption of label aggregation and label update steps in the iteration stage yields a smooth distribution of label embeddings on the graph for prediction, which relies less on label supervision. Similarly, other embedding-based methods (e.g., DGI and Graph2Gauss) also outperform GCN variants for cases with very few labels since their methods are designed to take advantage of the graph structure into embeddings in an unsupervised way. When the label rate goes higher, GCN variants perform better than unsupervised models. Li et al. [7] and Sun et al. [49] showed the limitation of GCN in a few label cases and proposed a co-training or self-training mechanism to handle this problem. Still, GraphHop outperforms its methods in various label rates. The limited performance of LP-based

D. Computational Complexity and Memory Requirement

Since GraphHop is an iterative algorithm, we study test accuracy curves as a function of the iteration number for all six datasets with different label rates in Fig. 4. We have two main observations. First, for Cora, Citeseer, and PubMed datasets, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively. Although fewer labeled nodes tend to demand more iterations to achieve convergence, its impact on convergence behavior is minor, which is consistent with Corollary 1. Second, for Reddit and Amazon2M, test accuracy curves converge in about ten, five, and four iterations, respectively.
TABLE V
CLASSIFICATION ACCURACY (%) FOR THREE CITATION DATASETS WITH DIFFERENT LABEL RATES. THE HIGHEST ACCURACY IN EACH COLUMN IS HIGHLIGHTED IN BOLD AND THE TOP THREE ARE UNDERLINED

| # of labels per class | Cora | CiteSeer | PubMed |
|-----------------------|------|----------|--------|
| 1                     | 2    | 4        | 8      | 16    | 20   | 1    | 2   | 4   | 8   | 16  | 20  | 1    | 2   | 4   | 8   | 16  | 20  |
| LP                    | 51.5 | 56.0     | 61.5   | 63.4  | 65.8  | 67.3  | 30.1 | 33.6 | 38.2 | 40.6 | 43.4 | 44.8 | 55.7 | 58.8 | 62.7 | 64.4 | 65.8 | 66.4 |
| LNP                   | 39.9 | 42.8     | 51.8   | 60.2  | 63.0  | 66.7  | 69.2  | 17.6 | 26.4 | 32.9 | 37.9 | 44.7 | 46.7 | 45.2 | 62.5 | 64.7 | 67.2 | 63.7 | 64.9 |
| Special LP            | 50.5 | 51.1     | 60.8   | 64.4  | 68.7  | 70.2  | 70.2  | 20.4 | 34.3 | 34.1 | 41.0 | 46.2 | 47.6 | 62.3 | 66.5 | 66.5 | 66.5 | 67.5 | 69.6 |
| Centered Kernel       | 25.0 | 26.2     | 38.9   | 51.4  | 50.2  | 53.2  | 45.0  | 28.6 | 31.8 | 29.1 | 36.3 | 42.1 | 38.5 | 41.7 | 52.9 | 46.1 | 49.0 | 49.9 | 51.0 |
| WNNL                  | 12.7 | 38.8     | 60.2   | 69.9  | 70.6  | 70.9  | 70.9  | 7.6  | 7.5  | 37.5 | 47.6 | 52.6 | 47.9 | 68.5 | 70.2 | 69.6 | 70.3 | 70.7 | 71.3 |
| Poison                | 12.7 | 44.2     | 61.2   | 69.5  | 70.4  | 72.3  | 72.3  | 7.6  | 7.5  | 37.5 | 47.6 | 54.6 | 49.2 | 67.2 | 68.7 | 66.4 | 65.7 | 71.3 | 72.2 |

DeepWalk                | 40.4 | 47.1     | 56.6   | 62.4  | 67.7  | 69.9  | 69.9  | 28.3 | 31.5 | 36.4 | 40.1 | 43.8 | 45.5 | -    | -   | -   | -   | -   | -   |
LINE                   | 49.4 | 56.0     | 63.0   | 67.3  | 72.6  | 74.0  | 74.0  | 28.0 | 31.4 | 36.4 | 40.6 | 45.8 | 48.5 | -    | -   | -   | -   | -   | -   |
DGI                    | 55.3 | 63.1     | 71.8   | 74.5  | 77.2  | 77.9  | 77.9  | 46.1 | 52.7 | 61.7 | 65.6 | 68.2 | 68.2 | 68.3 | 68.4 | 69.9 | 71.2 | 74.8 | 76.7 |
Graph2Gauss            | 54.5 | 61.3     | 69.5   | 72.4  | 74.8  | 75.8  | 75.8  | 47.1 | 50.8 | 58.4 | 61.7 | 64.4 | 65.7 | 67.2 | 66.0 | 68.7 | 67.6 | 69.4 | 69.6 |
GCN                    | 42.4 | 52.0     | 65.0   | 72.5  | 78.4  | 80.2  | 80.2  | 36.4 | 43.4 | 53.9 | 60.4 | 67.5 | 68.8 | 41.3 | 48.1 | 59.3 | 67.4 | 74.5 | 77.8 |
GAT                    | 41.8 | 51.8     | 66.4   | 73.6  | 77.8  | 79.6  | 79.6  | 32.8 | 40.7 | 51.8 | 57.9 | 64.5 | 68.2 | 57.6 | 70.0 | 69.8 | 71.7 | 75.6 | 76.5 |
Co-training GCN        | 53.1 | 59.4     | 68.0   | 73.5  | 78.9  | 78.7  | 78.7  | 36.7 | 42.9 | 52.0 | 57.9 | 62.5 | 65.9 | 55.1 | 59.9 | 66.9 | 71.3 | 75.7 | 77.9 |
Self-training GCN      | 40.6 | 52.3     | 67.5   | 73.8  | 77.3  | 79.1  | 79.1  | 34.6 | 42.3 | 54.4 | 63.1 | 68.3 | 69.1 | 49.7 | 56.2 | 65.0 | 68.9 | 73.6 | 76.5 |
GraphHop               | 59.8 | 58.2     | 69.3   | 76.3  | 79.7  | 81.0  | 81.0  | 48.4 | 55.0 | 55.1 | 60.4 | 66.7 | 70.3 | 69.3 | 70.9 | 71.1 | 71.9 | 75.0 | 77.2 |

TABLE VI
CLASSIFICATION ACCURACY (%) FOR THREE LARGE-SCALE GRAPH DATASETS, WHERE THE COLUMN OF LABELED SAMPLES IS MEASURED IN TERMS OF PERCENTAGES OF THE ENTIRE DATASET AND OOM MEANS “OUT OF MEMORY”

| % of labeled samples | Reddit | Amazon2M | PPI |
|----------------------|--------|----------|-----|
|                      |        |          |     |
| 1                    | 2      | 5        | 10  |
| FastGCN              | 78.6   | 80.2     | 86.7 |
| Cluster-GCN          | 92.0   | 92.7     | 93.7 |
| L-GCN                | 89.2   | 90.7     | 92.0 |
| GraphHop             | 93.4   | 94.1     | 94.7 |

To demonstrate the efficiency and scalability of GraphHop, we compare training time and memory usage of several methods in Table VII. Here, we focus on benchmarking models that can handle large-scale graphs, such as GraphSAGE [3], Cluster-GCN [12], L-GCN [6], and FastGCN [11]. For citation networks of smaller sizes, we adopt their original codes and implement them supervised. For large-scale graphs (i.e., Reddit, PPI, and Amazon2M), we follow the process discussed earlier and set the label rate to the largest. We measure the averaged running time per epoch (or per iteration for GraphHop) and the total training time in seconds. Early stopping is adopted, where we record the time when the performance on the validation set drops continuously for five iterations. For memory usage, we only consider the GPU memory.1

Generally speaking, GraphHop can achieve fast training with low memory usage. Although L-GCN has the lowest memory usage, all parameters are fixed without validation applied (validation data are counted in memory consumption for ours and other baselines), so the comparison may not be fair. To shed light on training complexity and memory usage,

1It is measured by torch.cuda.memory_allocated() for PyTorch and tf.contrib.memory_stats.MaxBytesInUse() for TensorFlow.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
we plot the time complexity versus memory usage of different methods on Reddit in Fig. 6(a) based on the data in Table VII. The lower left corner of this figure indicates the desired region with low training complexity and low GPU memory consumption. Furthermore, we can balance the training time and memory usage by changing the minibatch size, as shown in Fig. 6(b) for GraphHop. By increasing the minibatch size, the memory consumption increases in exchange for a lower training time.

E. Additional Observations

1) Ablation Study: To illustrate the effectiveness of GraphHop against the three weaknesses of traditional LP (i.e., failing to encode smoothened attribute information, without embedding parameter sharing, and one-hop neighbors propagation), we explore three variants of GraphHop in the following.

1) Variant I: GraphHop with the initialization stage only. It utilizes center’s and neighbors’ node features for label prediction without any label embeddings propagation.

2) Variant II: GraphHop without the initialization stage and LR classifiers between iterations, which is the same as vanilla LP in (3). It propagates label embeddings without leveraging any attribute information.

3) Variant III: GraphHop with the initialization stage and vanilla LP, where the label embeddings in LP are initialized from the classifiers’ predictions. It encodes smoothened attribute information with continuous label smoothening but without LR classifiers for parameter sharing.

We compare the test accuracy of all three variants against GraphHop under the same label rate in Table VIII. It is clear...
that the collective model design of GraphHop outperforms
the other three. Note that Variant III is the most similar to
GraphHop, and it yields the closest accuracy performance.
Still, the LR classifiers that serve as smoothening regulariza-
tion at each iteration can boost the performance. In addition,
we show test accuracy curves as a function of the iteration
number for Variant II and GraphHop in Fig. 7. We see that
GraphHop can achieve higher accuracy and converge faster
than Variant II. These indicate that a good initialization
of label embeddings and multihop neighbor aggregation results
in faster convergence.

2) Oversmoothening Problem: We study the oversmoothen-
ing problem by examining the convergence behavior of Graph-
Hop on Reddit in Fig. 8. The left subfigure shows the averaged
convergence curves of GraphHop on Reddit with multiple label
rates. The curves drop after five iterations and converge at
around 50 iterations. We argue that this phenomenon is due to
the oversmoothening of label embeddings. Generally speaking,
correlations of embeddings are valid only locally. These are
especially true for large-scale graphs. Adding uncorrelated
information from long-distance hops tends to have a negative
impact. To verify this claim, we conduct experiments on a
variant of GraphHop, whose label embeddings are updated in
the form of

\[
H^{(t)} = (1 - \tau) \frac{1}{K} \sum_{M=1}^{K} P_{\text{model}} \left( Y | H_{M}^{(t-1)} ; \theta \right) + \tau H^{(t-1)} \tag{15}
\]

where parameter \( \tau \in (0, 1) \) is used to control the update
speed. Equation (15) is also known as the residual connec-
tion [50]. A large \( \tau \) value enables the model to preserve more
information from the previous iteration and slows down the
smoothening speed. We report the results with \( \tau = 0.9 \) in the
right subfigure while keeping the other settings the same as the
left. We observe more stable curves with slower performance
degradation.

3) Fast Convergence: We explain why only a few iterations
can achieve convergence. The reason is that a small number
of iterations reaches sufficiency in Corollary 1. Without loss
of generality, we choose Cora and CiteSeer datasets for
illustration. We calculate the number of nodes in each subset,
\( \mathcal{V}_i \), in (11) up to \( k \) hops. The cumulative results are shown in
Fig. 9. Both subfigures indicate that all nodes in the graph can
be reached from labeled samples within less than ten hops. Fast
propagation of embeddings yields efficient label smoothening
and fast convergence. Also, we see from Fig. 9 that higher
label rates will reach sufficient iterations faster than lower
label rates. Besides, the label update step also contributes
to faster convergence due to the further smoothening of the
embeddings.

VI. COMMENTS ON RELATED WORK

A. Graph-Based Semisupervised Learning

There is rich literature on semisupervised learning [51],
including generative models [52], the transductive support
vector machine [53], entropy regularization [54], manifold learning [55], and graph-based methods [15], [56], [57]. Our discussion is restricted to graph-related work. Most semisupervised graph-based methods are built on the manifold assumption [51], where nearby nodes are close in the data manifold, and as a result, they tend to have the same labels. Early research penalizes nonsmoothness along edges of a graph with the Markov random field [58], the Laplacian eigenmaps [59], spectral kernels [60], and context-based methods [45], [61]. Their main difference lies in the choice of regularization. The quadratic penalty term applied on nearby nodes to enforce label consistency with the data geometry is the most popular one. The optimization result is shown to be equivalent to LP [17]. Traditional graph-based methods are nonparametric, discriminative, and transductive, making them lightweight with good classification performance. To further improve the performance, methods are developed by combining graph-based regularization with other entities to yield one joint learning framework. Instead of constructing the graph ahead, the adjacency weights can be learned adaptively through the optimization process [18], [19], [24], [27]. Rather than regularizing label embeddings, this idea can be extended to attributes [62] and even to hidden layers or auxiliary embeddings in neural networks. Manifold regularization [55] and Planetoid [63] generalize the Laplacian regularizer with a supervised classifier that imposes stronger constraints on the model learning. The work in [56] tries to generalize neural networks to transductive learning with the help of LP. However, their focus is mainly on image classifications instead of node classifications on graphs.

B. Graph Convolutional Networks

Inspired by the recent success of convolutional neural networks (CNNs) [1], [64]–[66] on images and videos, a series of efforts have been made to generalize convolutional filters from grid-structured domains to non-Euclidean domains [67]–[69] with theoretical support from graph signal processing [25]. The space spanned by the eigenvectors of the graph Laplacian can be regarded as a generalization of the Fourier basis. By following this idea, a deep neural architecture was formulated in [67] and [68] to employ the Fourier transform as a projection onto the eigenbasis of the graph Laplacian. Furthermore, to overcome the expensive eigendecomposition, recurrent Chebyshev polynomials were proposed in [70] as an efficient filter for approximation. GCN [2] further simplified it by only considering the first-order approximation in the Chebyshev polynomials. GCN has inspired quite a lot of follow-up work, e.g., [47], [71], and [72].

With the combination of embedding propagation and nonlinear activation, GCNs offer impressive results on the semisupervised classification problem. Later, it was explained in [7], [13], [14], and [30] that the success of GCNs is due to a low-pass filtering operation performed on node attributes. Specifically, Wu et al. [13], Li et al. [14], and NT and Maehara [30] have shown that the powerful feature extraction ability behind the graph convolutional operation in GCNs is due to a low-pass filter applied on the feature matrix to extract only smooth signals for prediction. This simplified graph convolution can be formulated as an LR classifier on the aggregated features [13]. That is, we have

\[
H = p_{\text{model}}(Y|S^kX; \theta)
\]

(16)

where \(p_{\text{model}}(\cdot)\) is the LR classifier, \(S^k\) is the \(k\)th power of the normalized adjacency matrix \(S\), \(\theta\) is the classifier parameters, and \(H\) is the label embeddings output as defined previously. This shed light on a simple filter design on the graph feature matrix. However, the intensive computation of matrix powers restrains the long-range correlations from each node. Besides, there is no restriction on the smoothness of the predicted label embeddings, limiting the generalization to extremely small label rate cases.

Despite the strength of GCNs, they have still limited in three aspects: 1) effective modeling of node attributes and labels jointly; 2) ease of scalability to large graphs; and 3) requirement of a considerable amount of labels for training. For the first point, unlabeled samples are not integrated into model training but only inference. Several algorithms have been proposed to tackle this deficiency. Zhang et al. [10] employed a Bayesian approach by modeling the graph structure, node attributes, and labels as a joint probability and inferring the unlabeled samples by calculating the posterior distribution. On the other hand, Qu et al. [9] used the conditional random field to embed the correlation between labeled and unlabeled samples with GCNs for feature extraction. In practice, these methods are costly and incurred by the local minimum during optimization. Another line of methods employed self-training techniques to generate pseudolabels for unlabeled samples and used them throughout training [7], [49], [73]. However, they do not utilize the correlation between labeled and unlabeled samples effectively and often suffer from label error feedback [74].

For the second point, the main issue of GCNs is the demand of loading the entire graph and intermediate node embeddings into memory, which makes the generalization to large graphs especially difficult. Unlike images in computer vision or sentences in natural language processing, one graph can be significant, while its nodes are connected without segmentation. The layerwise convolutional operation introduces an exponential expansion of neighborhood sizes [12], which hinders GCNs from minibatch training. Sampling-based strategies (e.g., GraphSAGE [3] and FastGCN [11]) have been proposed to overcome this problem. They attempt to reduce the neighborhood size during aggregation. Alternatively, some methods [5], [12] directly sample one or more subgraphs and perform subgraph-level training. Recently, You et al. [6] proposed a layerwise training algorithm for GCNs, called L-GCN. The idea is that, instead of training multiple GCN layers at once, the gradient update and parameters’ convergence are performed in a layerwise fashion. Nevertheless, L-GCN requires a large amount of training data (i.e., heavily supervised learning), and its performance degrades dramatically when only a few labeled samples are available in training.

The limitations from the first and second points and the nonlinearity result in the last aspect, i.e., the requirement of a considerable number of training labels. The large number of parameters and nonlinear activation terms induces GCNs to rely heavily on label supervision, where the joint
dependence of node labels is often ignored. These weaken GCN’s performance in the extremely small label rates.

VII. APPLICATIONS AND IMPROVEMENTS

Since GraphHop is an enhancement of the classical LP, any application that can be formulated as graph-based semisupervised classification could be considered. For example, the underlying manifold in image classification can be formulated as a graph, where labels are smoothly distributed. Then, GraphHop can be applied with only a few labeled images. GraphHop can also be applied to other vision tasks, e.g., face recognition [18], object recognition [19], video semantic recognition [75], and human activity recognition [76]. The effective encoding of node attributes in model learning makes GraphHop suitable for data with ample or high-quality features. This is confirmed by high accuracy results of networks (e.g., citation networks, copurchasing networks, and social networks) with rich attribute information. Another application is large-scale graph classification. New real-world networks are getting bigger. GraphHop can be directly applied to large-scale graphs simultaneously with low memory cost, fast running time, and high performance.

Although GraphHop achieves excellent performance in the node classification task, it could be further improved in several aspects. First, it adopts the LR classifiers to learn the mapping between the smoothed node attributes and labels, and regularize the aggregated neighborhood label embeddings with the node itself in the initialization stage and the label update, respectively. The LR is a linear classifier that may degrade the performance in nonlinear mappings. We may use a kernel in LR or adopt nonlinear classifiers. Second, GraphHop is applied to transductive semisupervised learning. It is desired to extend it to inductive learning. Third, the two main operations in GraphHop (i.e., aggregation and classifier training) attempt to smooth the attribute and label signals. This is feasible on graphs with smooth signals. However, the smoothness assumption may not hold for networks of low homophily or heterophily [77]. For example, people of the opposite gender are more likely to connect in dating networks, where high-frequency signals (i.e., the difference between nodes) could be more relevant.

VIII. CONCLUSION

A novel iterative LP-like method, called GraphHop, was proposed for transductive semisupervised node classification on graph-structured data. The main ingredients contributing to the success of GraphHop are: 1) jointly modeling the smoothing node attribute and smoothing label signals on graphs; 2) introducing regression classifiers in each iteration to regularize the label embeddings in a smoothing way; and 3) treating multihop neighbors independently and then aggregating them during the iteration process. They collaboratively lead to superior classification accuracy than the state-of-the-art GCN algorithms. GraphHop is scalable well to large-scale graphs and extremely small label rates. Theoretical derivation and extensive experimental results were provided to demonstrate GraphHop’s efficiency and effectiveness. In the future, we will extend GraphHop to more challenging tasks (e.g., inductive learning) and derive some deeper understanding from different angles (e.g., the regularization framework).

APPENDIX

IMPLEMENTATION OF LR CLASSIFIER

We explain the implementation details of the LR classifier in the label update step here. For labeled samples, the supervised loss term can be written as

\[ L_l = \frac{1}{|\mathcal{L}|} \sum_{y \in \mathcal{L}} H \left( \frac{p_{\text{model}}(y \mid \mathbf{h}_{M}^{(t-1)}; \theta)}{p_{\text{model}}(y \mid \mathbf{h}_{M}^{(t-1)}; \theta)} \right) \]  

(17)

where \( H(p, q) \) is the entropy loss for classification and \( \theta \) denotes the parameters of the classifier. The missing labels for unlabeled samples prevent direct supervision. However, we can leverage the label embeddings generated from the last iteration as pseudolabels for supervision. Since they also encode the current confidence to the node label distributions, the direct results of the classifier training are the probability predictions that are consistent between neighborhoods and the node itself, i.e., smoothing regularization.

Similar ideas are employed and viewed as consistency regularization [78]–[80], which enforces model predictions to be consistent under any input transformations. Then, the loss term for unlabeled samples can be expressed as

\[ L_u = \frac{1}{|\mathcal{U}|} \sum_{y \in \mathcal{U}} H \left( \text{Sharpen}(\mathbf{h}^{(t-1)}), \right) \]  

\[ p_{\text{model}} \left( y \mid \mathbf{h}_{M}^{(t-1)}; \theta \right) \]  

(18)

where \( \text{Sharpen}(\cdot) \) is a function to adjust the entropy of the label distributions within each embedding. In particular, a temperature is introduced to alternate the categorical distribution, which is defined as

\[ \text{Sharpen}(p, T)_{j} = p_{j}^{\dagger} / \left( \sum_{j=1}^{C} p_{j}^{\dagger} \right) \]  

(19)

where \( p \) is the categorical input distribution (i.e., a label embedding \( \mathbf{h} \) in GraphHop) and \( T \) is the temperature hyper-parameter. Using a higher value for \( T \) produces a softer probability distribution over classes and vice versa. By adjusting the temperature, models can decide how confident they should believe in the iteration of the current label embeddings. A similar idea is adopted in entropy minimization [54] and knowledge distillation [81].

The sharpening operation may result in uniform label distributions by setting a large temperature. To enforce the classifier outputs low-entropy predictions on unlabeled data, we minimize the entropy [54], [79] of model prediction \( p_{\text{model}}(y \mid \mathbf{h}; \theta) \) (abbr. \( p_{\text{model}} \)) with an additional loss term

\[ L_u' = \frac{1}{|\mathcal{U}|} \sum_{y \in \mathcal{U}} H(p_{\text{model}}, p_{\text{model}}). \]  

(20)

Combining the above loss term with labeled and unlabeled samples in (17) and (18), respectively, the final loss function can be derived as

\[ L = L_l + \alpha L_u + \beta L_u' \]  

(21)
where $\alpha$ and $\beta$ are two hyperparameters to adjust the scales of the unlabeled samples and the entropy, respectively. The LR classifier is trained in multiple epochs until convergence. Finally, new label embeddings for the next iteration are predicted by (10).

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, Feb. 2015.
[2] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Represent., 2016, pp. 1–14.
[3] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 1024–1034.
[4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengi, “Graph attention networks,” in Proc. Int. Conf. Learn. Represent., 2018, pp. 1–12.
[5] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, “GraphSAINT: Graph sampling based inductive learning method,” in Proc. Int. Conf. Learn. Represent., 2020, pp. 1–19.
[6] Y. You, T. Chen, Z. Wang, and Y. Shen, “L-GCN: Layer-wise and learned efficient training of graph convolutional networks,” in Proc. IEEE/ACM Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 2127–2135.
[7] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 1–8.
[8] J. Ma, W. Tang, J. Zhu, and Q. Mei, “A flexible generative framework for graph-based semi-supervised learning,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 3281–3290.
[9] M. Qu, Y. Bengio, and J. Tang, “GMNN: Graph Markov neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 5241–5250.
[10] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, “Bayesian graph convolutional neural networks for semi-supervised classification,” in Proc. AAAI Conf. Artif. Intell., vol. 33, 2019, pp. 5829–5836.
[11] J. Chen, T. Ma, and C. Xiao, “FastGCN: Fast learning with graph convolutional networks via importance sampling,” in Proc. Int. Conf. Learn. Represent., 2018, pp. 1–15.
[12] W.-L. Chang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks,” in Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Jul. 2019, pp. 257–266.
[13] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6861–6871.
[14] Q. Li, X.-M. Wu, H. Liu, X. Zhang, and Z. Guan, “Label efficient semi-supervised learning via graph filtering,” in Proc. IEEE/ACM Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 9582–9591.
[15] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” in Proc. Adv. Neural Inf. Process. Syst., vol. 16, 2004, pp. 321–328.
[16] X. Zhu, Z. Ghahramani, and J. D. Lafferty, “Semi-supervised learning using Gaussian fields and harmonic functions,” in Proc. 20th Int. Conf. Mach. Learn. (ICML), 2003, pp. 912–919.
[17] Y. Bengio, O. Delalleau, and N. Le Roux, Label Propagation and Quadratic Criteri”, Cambridge, MA, USA: MIT Press, 2006.
[18] Z. Zhang, F. Li, L. Jie, J. Qin, L. Zhang, and S. Yan, “Robust adaptive embedded label propagation with weight learning for inductive classification,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 8, pp. 3388–3403, Aug. 2018.
[19] H. Zhang, Z. Zhang, M. Zhao, Q. Ye, M. Zhang, and M. Wang, “Robust triple-matrix-recovery-based auto-weighted label propagation for classification,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 11, pp. 4538–4552, Nov. 2020.
[20] Q. L. Ye, J. Yang, T. M. Yin, and Z. Zhang, “Can the virtual labels obtained by traditional LP approaches be well encoded in WLR?” IEEE Trans. Neural Netw. Learn. Syst., vol. 27, no. 7, pp. 1591–1598, Jul. 2016.
[21] M. Wang, W. Fu, S. Hao, D. Tao, and X. Wu, “Scalable semi-supervised learning by efficient anchor graph regularization,” IEEE Trans. Knowl. Data Eng., vol. 28, no. 7, pp. 1864–1877, Jul. 2016.
