Graph-Based Semi-Supervised Learning: A Comprehensive Review

Zixing Song, Xiangli Yang, Zenglin Xu, Senior Member, IEEE, and Irwin King, Fellow, IEEE

Abstract—Semi-supervised learning (SSL) has tremendous value in practice due to the utilization of both labeled and unlabeled data. An essential class of SSL methods, referred to as graph-based semi-supervised learning (GSSL) methods in the literature, is to first represent each sample as a node in an affinity graph, and then, the label information of unlabeled samples can be inferred based on the structure of the constructed graph. GSSL methods have demonstrated their advantages in various domains due to their uniqueness of structure, the universality of applications, and their scalability to large-scale data. Focusing on GSSL methods only, this work aims to provide both researchers and practitioners with a solid and systematic understanding of relevant advances as well as the underlying connections among them. The concentration on one class of SSL makes this article distinct from recent surveys that cover a more general and broader picture of SSL methods yet often neglect the fundamental understanding of GSSL methods. In particular, a significant contribution of this article lies in a newly generalized taxonomy for GSSL under the unified framework, with the most up-to-date references and valuable resources such as codes, datasets, and applications. Furthermore, we present several potential research directions as future work with our insights into this rapidly growing field.

Index Terms—Graph embedding, graph representation learning, graph-based semi-supervised learning (GSSL), semi-supervised learning (SSL).

NOMENCLATURE

\( G \) Graph.
\( V \) Set of nodes (vertices) in a graph.
\( E \) Set of edges in a graph.

I. INTRODUCTION

SEMI-SUPERVISED learning (SSL) has achieved great success in various real-world applications where only a few expensive labeled samples are available and abundant unlabeled samples are easily obtained. Moreover, as a typical class of SSL solutions, graph-based SSL (GSSL) is very promising because the graph structure can be naturally used as a reflection for the significant manifold assumption in SSL. More specifically, GSSL methods start with constructing a graph where the nodes represent all the samples, and the weighted edges are usually employed to reflect the similarity between a pair of nodes. This way of graph construction implies that nodes connected by edge associated with large weights tend to have the same label, which corresponds to the manifold assumption that samples locating near each other on a low-dimensional manifold should share similar labels.
Consequently, the expressive power of graph structure under the manifold assumption contributes to the success of GSSL methods. The main two-step procedure of GSSL is to first create a suitable graph along which the given labels can be easily propagated afterward, as shown in Fig. 1.

Step 1 (Graph Construction): A similarity graph is constructed based on the given data, including both the labeled and unlabeled samples. The biggest challenge is how to make the relationship between original samples well represented.

Step 2 (Label Inference): The label inference is performed so that the label information can be “propagated” from the labeled nodes to the unlabeled ones by incorporating the topological information from the constructed similarity graph.

Compared with other SSL methods, which are not involved with the graph structure, GSSL has following advantages.

1) Universality: Many common datasets of current trends are represented by graphs, such as the World Wide Web (WWW), citation networks, and social networks.
2) Convexity: Since an undirected graph is usually involved in the graph construction step, the symmetric feature of the undirected graph makes it easier to formulate the learning problem into a convex optimization problem, which can be solved with various exciting techniques [1].
3) Scalability: Many of the GSSL methods are meticulously designed so that the time complexity is linear to the total number of samples. As a result, they are often easily parallelized to handle large-scale datasets with ease.

A. Related Work

Several SSL survey papers [2], [3] often fail to cover enough methods of GSSL, neglecting its significant role in SSL. Zhu et al. [2] conducted a comprehensive review of classic methods involved in SSL, and GSSL is not explored in detail. Similar earlier work such as [3] also tries to present a whole picture of SSL methods without covering enough work in GSSL. Recent literature review work, Prakash and Nithya [4], van Engelen and Hoos [5], and Ouali et al. [6] followed the footsteps of work [2] and [3] by adding more research output. However, they do not cover the recent development in GSSL. Instead, our work solely focuses on GSSL and combines both earlier studies with recent advances.

The most relevant work to ours is [7], and it is considered as the most up-to-date survey paper on GSSL. However, there are several noticeable drawbacks of this work that are worth mentioning here. First, Chong et al. [7] reviews work from the perspective of transductive, inductive, and scalability learning. This taxonomy fails to show the context of development and thus does not reveal the relationship of different methods or models. As a result, we provide a novel taxonomy from the perspective of the two main steps in GSSL: graph construction and label inference. Second, some of the reviewed methods in [7] are not graph-based models, but rather are convolutional neural network (CNN) models as shown in Section III-D in the original paper [7]. Most importantly, Chong et al. [7] failed to develop a unified framework to generalize the methods or models reviewed. However, this article fills all these gaps with several noticeable contributions.

B. Contributions

To sum up, this article presents an extensive review of GSSL with the following contributions.

1) Comprehensive Review: We provide the most comprehensive and the most up-to-date overview of GSSL methods. For every approach reviewed in this article, we present the detailed descriptions with key equations, clarify the context of development beneath the algorithms, make the necessary comparison, and summarize their respective strengths or limitations.
2) New Taxonomy: We propose a new taxonomy for GSSL with a more generalized framework, as shown in Fig. 2. We divide the GSSL process into two steps. The first one is to construct a similarity graph and the second step is to do label inference based on this graph. The latter step is more challenging and receives more attention from the community.
3) Unified Frameworks: Label inference methods are the main focus of this article, and they can be categorized into two main groups: graph regularization methods and graph embedding methods. For the former group, a generalized framework of regularizers from the perspective of the loss function is presented. For the latter group, we provide a new unified representation for graph embedding methods in SSL with the help of the encoder–decoder framework.
4) Abundant Resources: We collect abundant resources related to GSSL and build a useful, relevant code base, including the open-source codes for all the reviewed methods or models, some popular benchmark datasets, and pointers to representative practical applications in different areas. This survey can be regarded as a hands-on guide for researchers interested in understanding existing GSSL approaches, using the codes for experiments, and even developing new ideas for GSSL.
5) Future Directions: We propose some open problems and point out some directions for future research with our insights based on the latest representative works in this active area.

C. Organization of This Article

The rest of this survey is organized as follows. In Section II, we introduce the background knowledge related to GSSL with the overview of our proposed taxonomy. In Section III, we provide a detailed review of graph construction, the first step of GSSL. Sections IV and V discuss label inference and the second step of GSSL. Following the proposed taxonomy shown in Fig. 2, graph regularization methods are reviewed in Section IV, while graph embedding methods in Section V are further split into shallow graph embedding and deep graph embedding. Moreover, applications of GSSL are extensively explored in Section VI, along with a list of common

---

1 A list of references can be found at https://github.com/AnthonySong98/awesome-graph-based-semi-supervised-learning.
datasets and a code base for some popular models. Finally, in Section VII, three open problems are briefly reviewed as future research directions.

II. BACKGROUND AND OVERVIEW

A. Related Topics

1) Supervised Learning and Unsupervised Learning: Supervised learning and unsupervised learning can be viewed as two extremes of SSL because all the training samples are well labeled in supervised learning settings, while unsupervised learning can only have access to unlabeled data. SSL aims to introduce cheap unlabeled samples to enhance the model's performance with only a few costly labeled samples. Therefore, the problem setting of SSL is a natural match for many real-world applications in which labels are hard to obtain.

2) Other SSL Methods: Throughout the development of SSL, a great number of successful algorithms or models have emerged in roughly three phases. The first phase is the early stage of SSL before 2000, where classic machine learning algorithms are investigated and improved with unlabeled data, such as S3VM and cotraining. The second phase is the mature stage of SSL between 2000 and 2015, in which many methods flourished, such as mixture model, pseudo label, self-training, manifold learning, and GSSL. The third phase is after 2015, with the advance of deep learning and especially graph neural networks (GNNs). Since GSSL methods witness all these three stages, reviewing its development and recent progress is necessary.

B. Learning Settings

Like other SSL methods, GSSL algorithms can be divided into two categories based on whether to predict data samples' labels out of training data. We list several common notations used in this article in the Nomenclature with detailed definitions in Appendix A of the supplementary material.

Definition 1 (Transductive Setting): Given a training set consisting of labeled and unlabeled data \( D = \{(x_i, y_i)\}_{i=1}^{n_l}, \{(x_i, y_i)\}_{i=1}^{n_u}\} \), the goal of a transductive algorithm is to learn a function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) so that \( f \) is only able to predict the labels for the unlabeled data \( \{x_i\}_{i=1}^{n_u}\).

Definition 2 (Inductive Setting): Given a training set consisting of labeled and unlabeled data \( D = \{(x_i, y_i)\}_{i=1}^{n_l}, \{(x_i)\}_{i=1}^{n_u}\} \), the goal of an inductive algorithm is to learn a function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) so that \( f \) is able to predict the output \( y \) of any input \( x \in \mathcal{X} \).

While most of the GSSL approaches are transductive, there are several inductive GSSL approaches. In most scenarios, transductive SSL methods outperform inductive ones in terms of prediction accuracy, while they often suffer from higher computational cost compared to inductive ones, especially in the context of large-scale incremental learning. Fig. 3 shows their difference in GSSL.

C. Problem Formulation

The goal of graph construction is to discover a graph \( G = (V, E, W) \) where \( V \) is the set of nodes, \( E \subseteq V \times V \) are the edges, and \( W \in \mathbb{R}^{n \times n} \) are the associated weights on the edges. Each node in the graph represents an input sample, and thus, the number of nodes in the graph is \( |V| = n \). As the nodes are fixed (assuming that \( D \) is fixed, which is often the case), the task of graph construction involves estimating \( E \) (unweighted graph) and \( W \) (weighted graph) with the help of a proximity function or distance metric \( d(x_i, x_j) \) that can quantify the resemblance or disparity between every node pair in the training data.

Then, the label inference step seeks to find out a predictive function \( f \) based on the constructed affinity graph to satisfy the manifold assumption. If two nodes \( i \) (\( x_i \)) and \( j \) (\( x_j \)) are adjacent (undirected graphs) or are connected by a edge with a large weight \( W_{ij} \) (directed graphs), their predicted labels, \( f(x_i) \) and \( f(x_j) \), should be similar and not vary too much. For the transductive setting, \( f \) is only suitable for the unlabeled samples \( \{x_i\}_{i=1}^{n_u}\) within the training set, while for the inductive setting, \( f \) can apply to any new sample outside the training set.

D. Chronological Overview

Here, a chronological overview of the mentioned representative methods in this survey is shown in Fig. 4. Most of
in the early years. After the resurgence of deep learning in
centered around graph regularization and matrix factorization
first step. From 2000 to 2012, the mainstream algorithm was
methods focus on the second label inference step with a
methods, while, in the meantime, random-walk-based methods
2015, the field witnessed the emergence of AutoEncoder-based
methods are further split into two groups: graph regularization
between nodes
and
Since no label information is used in the regularizer loss
term. Therefore, the loss function for GSSL can be generalized
as shown in the following equation:
\[ \mathcal{L}(f) = \mathcal{L}_s(f, D_l) + \mu \mathcal{L}_r(f, D) \]  
(2)
According to our proposed framework in Fig. 2, label inference
methods are further split into two groups: graph regularization
and graph embedding. The former develops various forms of
regularization loss \( \mathcal{L}_r(f, D) \) in (2) to reflect the manifold
assumption on graphs, while the latter generates more distin-
guishable embeddings for nodes on a hidden low-dimensional
manifold so that it is easier for the predictor \( f \) to detect the
boundaries between classes. More precisely, the predictor in
graph regularization methods directly takes the original node
features as input and then regularizes the predicted results based
on the graph structure to keep consistent with the homophily
property, while graph embedding methods can simultaneously
take both the feature information and the topological informa-
tion into account by generating node embeddings, which often
enjoys a greater expressive power.

III. Graph Construction

To perform any GSSL methods, a graph must be constructed
first, where nodes represent data samples, some of which are
labeled while others are not, and edges are associated
with a certain weight to reflect each node pair’s similarity.
In some domains, such as citation networks, there is already
an implicit underlying graph. Graph-based methods are thus
a natural fit for SSL problems in these domains. For most of
the other machine learning tasks, however, it is believed that
the data instances are not conveniently represented as a graph
structure, and as a result, a graph has to be built to make these
problems appropriate for GSSL approaches. It is worth noting
that this step can be further divided into two substeps: edge
generation and weight assignment [12]. The edge generation
will decide which pairs of nodes are linked and thus produce
an unweighted graph first. The exact associated weights with
these generated edges are then determined in the next step
of weight assignment to construct a final weighted affinity
graph. Although the less informative unweighted graph in
the first step can also be considered in GSSL investigations,
we can actually view the unweighted graph as a special case
of the weighted graph, where each edge is associated
with the same weight. Therefore, the methods reviewed in
this section are mostly based on the more general weighted
diagrams unless otherwise specified. Moreover, the following
three assumptions often hold during the graph construction.

Assumption 1: The graph is undirected, so \( W \) is symmetric.
Also, all edge weights are nonnegative, \( W_{ij} \geq 0, \forall i \neq j \).

Assumption 2: \( W_{ij} = 0 \) implies the absence of an edge
between nodes \( i \) and \( j \).

Assumption 3: There are no self-loops, \( W_{ii} = 0 \).

These three assumptions simplify the problem by adding
these constraints and lay the foundations for the following
unsupervised and supervised methods. Meanwhile, we often
need a proximity function or distance metric \( d(x_i, x_j) \) such as
\( l_2 \) distance [15] and cosine distance [18] that can quantify
the resemblance or disparity between every node pair in the
training data.

A. Unsupervised Methods

Unsupervised graph construction techniques ignore all the
given label information of the training data during the
construction process. Among all the unsupervised methods for graph construction, the K-nearest neighbor (KNN) graph is the most classic choice in the literature.

1) **KNN-Based Methods:** For KNN-based graph construction approaches [57], every node is associated based on a preconfigured distance metric with its KNNs in the resulting graph. Moreover, KNN-based methods link the KNNs greedily to generate graphs whose nodes’ degree is larger than k, which leads to irregular graphs. Note that a graph is said to be regular if every node has the same degree. In the $\varepsilon$-neighborhood-based graph construction method [57], if the distance between a node pair is smaller than $\varepsilon$, where $\varepsilon \geq 0$ is a predefined constant, a connected edge is formed between them. KNN methods enjoy certain favorable properties when compared with $\varepsilon$-neighborhood-based graphs. Specifically, in $\varepsilon$-neighborhood-based graphs, a misleading choice of the parameter $\varepsilon$ could lead to generating disconnected graphs, while KNN graphs are adaptive to scale and density [58].

Despite the benefits of the KNN graph, it is contended that a hub or a center situated in the sample space can result in a corresponding hub in the classic KNN graphs [8]. This may downgrade the prediction performance on several classification tasks. In order to handle this issue, M-KNN is proposed a new way of constructing a graph by using mutual KNN in combination with a maximum span tree [8]. In parallel with this work, sequential KNN (S-KNN) [27] is also introduced to produce graphs under the new relaxed condition in which the resulting graph contains no hubs but is not necessarily regular.

2) **b-Matching:** As discussed above, KNN-based methods often lead to producing graphs where different nodes have different degrees, while b-Matching [9] guarantees that every node in the resulting graph has exactly b neighbors. b-Matching provides a solution by formulating an optimization problem with the objective

$$\min_{P \in \{0,1\}^{n \times n}} \sum_{i,j} P_{ij} \Delta_{ij}$$

s.t. $P_{ij} = 1$, $P_{ii} = 0$, $\forall 1 \leq i, j \leq n$. (3)

Here, $\Delta \in \mathbb{R}^{n \times n}$ is a symmetric distance matrix based on $D$. For the entry in $D$, $P_{ij} = 1$ signifies an existing edge between a node pair in the generated graph, while $P_{ij} = 0$ suggests a lack of an edge. In summary, the b-Matching method restricts the constructed similarity graph to be regular so that the given label can be propagated in a more balanced way during the following label inference step.

3) **Kernel-Based Methods:** Both KNN-based methods and b-Matching achieve the goal of edge generation and they can be easily extended to the weighted graph generation with the aid of kernel-based methods. The most common choice is the Gaussian kernel defined as $W_{ij} = P_{ij} \exp(-((d(x_i, x_j))^2)/(2\sigma^2)))$, where $\sigma$ is the bandwidth hyperparameter to set beforehand. Therefore, the performance of the Gaussian kernel is sensitive to $\sigma$. To alleviate this issue, the HM similarity function [59] can be utilized with an adaptive kernel size.

4) **Locally Linear Reconstruction-Based Methods:** Locally linear reconstruction (LLR) is derived from the locally linear embedding (LLE) technique [55]. The goal is to reconstruct the node feature vector $x_i$ from its neighborhood and use the derived weights as the similarity measurement. It can be formulated to solve the following optimization problem:

$$\min_{W} \sum_{i} \left\| x_i - \sum_{j} W_{ij} x_j \right\|^2$$

s.t. $\sum_{j} W_{ij} = 1$, $W_{ij} \geq 0$, $i = 1, \ldots, n$. (4)

Instead of reconstructing each node with all the other nodes and thus generating the fully connected graph in LLR [55], linear neighborhood propagation (LNP) [21] only makes use of one-hop neighborhood to fit the target node so that it can be easily generalized to out-of-sample nodes. SIS [10] decomposes each node as a sparse linear combination of the rest of the data points by moving the objective in (4) into the constraint and minimizing the $l_1$ norm of the weight matrix instead, avoiding the predefined distance metrics. NNLR Graph [11] follows a similar idea but minimizes the rank of the weight matrix in order to capture both the global and local structures. AnchorGraph [56] views each node as a locally weighted average of the selected anchor nodes so as to reduce the time complexity down to the linear running time. AEW [12] is specifically designed for LP-based label inference models [15] and directly aims to find the optimal width parameters in Gaussian kernel.

B. **Supervised Methods**

The existing prevalent strategies of graph construction are unsupervised, i.e., they fail to use any given label information during the construction phase. However, labeled samples can be used as a kind of prior knowledge that can be used to refine the generated graph [60] for the downstream learning tasks. Driven by previous studies [8], a new method, graph based on informativeness of labeled instances (GBILI) [26], also utilizing the label information, is introduced. GBILI not only results in a decent accuracy on classification tasks but also stands out with a quadratic time complexity [61]. Moreover, built on GBILI [26], Berton et al. [38] have upgraded the method for producing more robust graphs by solving an optimization problem with the specific algorithm called the robust graph that considers labeled instances (RGCLI). More recently, a new SSL learning method referred to as a low-rank semi-supervised representation is proposed [62], which incorporates labeled data into the low-rank representation (LRR). A follow-up work is by Taherkhani et al. [63]. By taking additional supervised information, the generated similarity graph can significantly facilitate the following label inference process.

C. **Summary**

For ease of reading, we summarize the main differences among the reviewed representative graph construction methods in Table I. Although the constructed graph is at the heart of the GSSL problem, there is still no universal way to construct a “best” affinity graph [2]. However, empirical results [64] show that the M-KNN graph could be the best option for the unweighted graph, while the Gaussian Kernel may be the best fit for the weighted graph. Meanwhile, a visualization system [65] is built to interactively construct a high-quality graph for better label inference. More recent investigations on graph construction techniques shift from the classical static setting where each node is fixed to the dynamic setting...
where nodes or data distribution may vary over time [66]–[68]. Another increasingly popular research topic is to provide a holistic way of conducting graph construction and label inference simultaneously so that the influence of the similarity graph can be optimized based on the performance of label inference [69], [70].

IV. GRAPH REGULARIZATION

All the classic GSSL methods can actually be simplified by searching for a function \( f \) on the graph. \( f \) has to satisfy two criteria simultaneously: 1) it must be as close to the given labels as possible and 2) it must be smooth on the entire constructed graph.

These two conditions can be further expressed in a general regularization framework in which loss function can be decomposed into two main parts. The first term is a supervised loss constrained to the first criterion, and the second term is a graph regularization loss constraint to the second criterion. Formally, we have

\[
L(f) = \sum_{(x_i, y_i) \in D_{\text{supervised}}} L_s(f(x_i), y_i) + \mu \sum_{x_i \in D_{\text{supervised}}} L_r(f(x_i)) \tag{5}
\]

where \( f \) is the prediction function and \( \mu \) is a tradeoff hyperparameter.

According to whether a propagation matrix is involved in diffusing the label information, we review the graph regularization from two subgroups: label-propagation (LP)-based methods and general-regularization-based methods. For ease of reading, we list some pros and cons of the reviewed methods that we will discuss in detail in Table II. This framework of graph regularization can be traced back to [78], and its theoretical analysis from different perspectives has also been provided in [79]–[81].

A. LP-Based Methods

LP [82] is one of the earliest yet still the most popular methods for label inference, along with its various extensions. LP can be formulated as a problem, in which some of the nodes’ labels, also referred to as seeds, “propagate” to unlabeled nodes based on the similarity of each node pair, which is represented by the constructed graph discussed in Section III. Meanwhile, during the propagation process, predicted labels for labeled nodes are fixed as ground-truth labels or are constrained not to deviate from them too much. In this way, labeled nodes serve as guides that lead label information flow through the edges within the graph so that unlabeled nodes can also be tagged with predicted labels. The most obvious feature of LP-based methods is that a propagation matrix is involved to iteratively propagate the label information from labeled nodes to unlabeled nodes in the neighborhood.

Therefore, the basic framework of the LP algorithm is shown in Algorithm 1. We use the classification problem with \( c \) classes as an example task to show its procedure. Here, \( F = \{F_1, F_2, \ldots, F_c\} \in \mathbb{R}^{n \times c} \) with nonnegative entities. \( Y_i \in \mathbb{R}^{n \times c} \) is the given label matrix where \( Y_{ij} = 1 \) if node \( i \) is labeled as \( j \) and \( Y_{ij} = 0 \) otherwise.

1) Gaussian Random Fields: Gaussian random field (GRF) [15] is a typical example of the early yet still popular work in GSSL by using LP algorithms. The strategy is to estimate some prediction function \( f \) based on the graph \( G \) with some constraints to ensure certain necessary properties and afterward attach labels to the unlabeled nodes according to \( f \). In fact, the abovementioned constraint is to take \( f_i(x_i) \equiv Y_i \) on all the labeled nodes as shown in the optional step in Algorithm 1. Intuitively, the clustering unlabeled points with strongly connected edges should share common labels. This is why the quadratic energy function is designed as shown in the following equation:

\[
E(f) = L(f) = \frac{1}{2} \sum_{i,j} W_{ij} (f(x_i) - f(x_j))^2. \tag{6}
\]
It is noteworthy that the minimum value of energy function \( f = \arg \min_{f|l_0=x} E(f) \) is harmonic; namely, it satisfies the constraint \( Lf = 0 \) on the unlabeled nodes and is equal to \( f \) on the labeled nodes \( D_l \), where \( L \) is the graph Laplacian matrix. The property of harmonic function indicates that the value of \( f \) at every unlabeled node is the mean value of \( f \) at its neighboring nodes: \( f(x_j) = (1/d_j) \sum_{i \sim j} W_{ij} f(x_i) \), for \( j = l + 1, \ldots, l + u \). This constraint is actually compatible with the previous smoothness requirement of \( f \) with respect to the graph.

GRF can also be interpreted in an iterative manner as \( f^{(t+1)} \leftarrow P \cdot f^{(t)} \), where \( P = D^{-1}W \). Furthermore, a closed-form solution can be deduced if weight matrix \( W \) is split into four blocks \( W = \begin{bmatrix} W_{ll} & W_{lu} \\ W_{ul} & W_{uu} \end{bmatrix} \). Then,

\[
 f_u = (D_{uu} - W_{uu})^{-1} W_{ul} f_l = (I - P_{uu})^{-1} P_{ul} f_l. 
\]

(7)

2) Local and Global Consistency: Local and global consistency (LGC) [16] is an extended work for multiclass setting based on GRF [15]. The main change is that LGC relaxes the restriction on labeled nodes in GRF by removing the label clamping operation. The iterative formula is shown in (8) by using a symmetric propagation matrix.

\[
 F^{(t)} = \alpha SF^{(t-1)} + (1 - \alpha) F^{(0)} 
\]

(8)

where \( S = D^{-1/2}WD^{-1/2} \), and \( \alpha \) is a hyperparameter. We can also easily derive the closed-form solution for (8) as shown in (9) when omitting the constant coefficient \( 1 - \alpha \) for the classification problem

\[
 F^* = (I - \alpha S)^{-1} F^{(0)}. 
\]

(9)

From a perspective of optimization problem, LGC [16] actually tries to minimize the following objective function (10) associated with prediction function \( f \):

\[
 L(f) = \frac{1}{2} \left( \sum_{i,j} W_{ij} \left( \frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j) \right)^2 \right) + \mu \sum_{i=1}^{n_l} (f(x_i) - y_i)^2. 
\]

(10)

Compared to the GRF objective above, LGC has two important differences: 1) the inferred labels for the labeled nodes are no longer required to be exactly consistent with ground-truth labels and this helps with cases where there may be noise in the seed labels and 2) the label for each node is penalized by the degree of that node \( (1/\sqrt{D_{ii}}) \), ensuring that

in the case of irregular graphs, the influence of high degree nodes is regularized.

There exist quite a few variants of LGC method, and a representative one is p-Laplacian regularization [48]. The first term in (10) can be substituted by a more general one \( \sum_{i,j} W_{ij} |(1/\sqrt{D_{ii}}) f(x_i) - (1/\sqrt{D_{jj}}) f(x_j)|^p \), where \( p \) is a positive integer. A comprehensive analysis of its theoretical grounds is also provided [48]. Motivated by earlier work [83], another recent advance is to incorporate label correlation into the basic LGC method with the emphasis on dynamic metric fusion so that it can be adapted to the multilabel classification scheme [23]. Instead of keeping propagation matrix fixed during the iteration, it is now dependant on the time step and is given as \( P_{t+1} = P_0 (P_t + \alpha F^T_t) P_0^T + \lambda_t I \), where \( P_0 \) is the initial propagation matrix and \( \alpha \) and \( \lambda_t \) are balancing factors. TLLT [84] also adopts a dynamic scheme by manipulating the propagation sequence from simple to difficult nodes.

3) Special Label Propagation: Compared with the harmonic function and consistency methods, special label propagation (SLP) [22] can not only meet the target of GSSL but also detect outliers or discover the latent novel class in the data. SLP introduces an additional label \( c + 1 \), which gives the original LP algorithm a chance to discover novel classes. This simple mechanism of novel class detection is useful since the unlabeled nodes may not necessarily belong to all the provided labeled classes. Similar to (9), the closed-form solution is given as \( F^* = (I - I_n P)^{-1} (I - I_n) F^{(0)} \), where \( I_n \) is a diagonal matrix with each entry \( a_i \) for node \( i \). It is worth mentioning that the dimension of \( F \) has been changed to \( n \times (c + 1) \).

There are several noticeable extensions on SLP. For instance, SLP can be utilized to create and enrich the pairwise constraint (PC) sets to obtain more supervised information for enhanced performance under the extreme situation of insufficient labels [85]. Another line of research is to extend SLP to the task of semi-supervised dimension reduction with the aid of a weighted regression model [86] and a deep connection between SLP and GRF is analyzed in [87]. More recent works center around how to merge the graph construction and SLP into one step so that the tricky choice of neighborhood size or kernel width in the first step can be avoided. Adaptive-NP seamlessly integrates sparse coding and neighborhood propagation into a unified framework. Adaptive-NP [88] integrates sparse coding and neighborhood propagation into a unified framework. AELP-WL [89] and ALP-TMR [90] both purpose a robust and inductive version of SLP.

4) Linear Neighborhood Propagation: LNP [21], [71], [72] not only produces a new way of graph construction with the assumption that each node can be linearly reconstructed
by its neighbors but also derives a straightforward way to extend GRF [15] and LGC [16] to the inductive setting. According to the regularization term in (6), the regularization term for the single new node $v$ is given as $\mathcal{L}(f(x_i)) = \sum_{i \in N_v} W_{i v} (f(x_i) - f(x_j))^2$. Since $\mathcal{L}(f(x_i))$ is convex with respect to $f(x_i)$, it is minimized when (11) holds, which is exactly corresponds to the case when the label of node $v$ can be optimally reconstructed from its neighbors as shown in the following equation:

$$f(x_v) = \sum_{i \in N_v} W_{i v} f(x_i). \quad (11)$$

A probabilistic view of LNP is naturally derived in [72]. The outcome through LNP is equivalent to the conditional expectation through a second-order intrinsic Gaussian–Markov random field (IGMRF) [91]. Essentially, it is also equivalent to solving a biharmonic equation $(I - W)^T (I - W) f = 0$ with the Dirichlet boundary condition $f_j (x_i) \equiv y_i$ for $x_i \in \mathcal{V}$, as shown in the following equations:

$$\pi(i) = \sum_{j \rightarrow i} \alpha(j) p(j, i) \quad \forall i \in \mathcal{V} \quad (12)$$

$$p(i, j) = \frac{W_{i j}}{d^+(i)} = \frac{W_{i j}}{\sum_{j' \in i} W_{i j'}}. \quad (13)$$

In (12) and (13), $j \rightarrow i$ denotes the set of vertices adjacent to the vertex $i$, while $j \leftarrow i$ denotes the set of vertices adjacent from the vertex $i$. Thus, we can define a regularization term that sums the weighted variation of each edge in the directed graph as shown in the following equation:

$$\mathcal{L}(f) = \frac{1}{2} \sum_{i, j} \pi(i) p(i, j) \left( \frac{1}{\sqrt{D_{i j}}} f(x_i) - \frac{1}{\sqrt{D_{j i}}} f(x_j) \right)^2. \quad (14)$$

It is also worth noting that (10) for undirected graphs can be regarded as a specific case of (14) for directed graphs. The stationary distribution of the random walk in an undirected graph is $\pi(j) = D_{j i} / \sum_{i \in \mathcal{V}} D_{i i}$. By substituting this expression into (14), we can easily derive (10), which is exactly the regularizer of LGC [16].

2) Tikhonov and Interpolated Regularization: Both Tikhonov regularization [24] and interpolated regularization [73] can be, respectively, viewed as the extension of LGC [16] and GRF [15] for the regression task. The main difference is that they normalize the data by subtracting the mean first as $\tilde{y}_i = y_i - \bar{y}$. For Tikhonov regularization, the objective function is given as (15) with the zero-mean constraint on the predicted value $\sum_{i=1}^n f(x_i) = 0$

$$\mathcal{L}(f) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - \tilde{y}_i)^2 + \mu f^TSf \quad (15)$$

where $\mathcal{S}$ is the smoothness matrix and can be set as the Laplacian matrix $L$. The closed-form solution to this problem can be obtained as $f^* = (n \mu S + I_n)^{-1} (\bar{y} + \gamma 1)$, where $I_n$ is a diagonal matrix with $n$ of 1 and gamma is chosen to satisfy the zero-mean constraint.

Similar to the label clamping operation in GRF, interpolated regularization forces the label constituency on labeled nodes by $f(x_i) = \tilde{y}_i$, and thus, the optimization objective is shown as (16) with the same zero-mean constraint

$$\mathcal{L}(f) = f^TSf. \quad (16)$$

By factoring the smoothness matrix $\mathcal{S}$, the optimal solution is now changed to $\mathcal{S}^n_{uu} \mathcal{S}_d^d ((\tilde{y}_1, \ldots, \tilde{y}_n)^T + \gamma 1)$.

3) Manifold Regularization: The manifold regularization [18], [92] is actually a general framework that allows for developing a great number of algorithms ranging from supervised learning to unsupervised learning. However, it is viewed as a natural fit for GSSL since it combines the spectral graph theory with manifold learning to search for a low-dimensional representation with smoothness constraint in the original commonly high-dimensional space.

The manifold regularization framework fully utilizes the geometry property of the unknown probability distribution, which the data samples obey. Therefore, it introduces another term as a regularizer to control the complexity of the prediction function in the intrinsic space, measured by the geometry of the probability distribution.

Formally, for a Mercer kernel $K: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, we denote the associated reproducing kernel Hilbert space (RKHS) of the prediction function $f$. Then, the loss function can be formulated in (17) as

$$\mathcal{L}(f) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2 + \gamma_A \|f\|^2_2 + \gamma_f \|f\|^2_1 \quad (17)$$

where $\gamma_A$ balances the complexity of the prediction function in the ambient space and $\gamma_f$ is the weighting parameter for the smoothness constraint term $\|f\|^2_2$ induced by both labeled and unlabeled samples.

It is noted that the added regularization term $\|f\|^2_2$ usually takes the following form:

$$\|f\|^2_2 = \frac{1}{(n_1 + n_2)^2} \tilde{y}^T L \tilde{y} \quad (18)$$

where $\tilde{y} = [f(x_1), f(x_2), \ldots, f(x_{n_1+n_2})]^T$ and $L$ is the Laplacian matrix of the graph. According to the representer theorem [93], it is well known that (17) has a closed-form solution as (19) when $\|f\|^2_2$ above takes the form as shown in (18)

$$f^*(x_i) = \sum_{j=1}^n \alpha_j K(x_j, x) \quad (19)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where \( K(\cdot, \cdot) \) is the kernel function and \( \alpha^* \) is the set of regression parameters. The significant advantage of manifold regularization is that it can be naturally extended to the inductive setting due to the formation of its closed-form solution by setting \( \gamma_A = 0 \) [18], [74]. However, it suffers from the high computational cost [94], which makes the algorithm unscalable when faced with large graphs. Popular solutions to alleviate this problem would be to accelerate either the construction of the Laplacian graph [95], [96] or the kernel matrix operation [97], [98]. Another influential extension of manifold regularization is to convexly combine several matrix operation [97], [98]. Another influential extension of manifold regularization is to convexly combine several construction of the Laplacian graph [95], [96] or the kernel matrix operation 

\[ L = \sum_{i=1}^{m} z_{ik} a_k \]

Following the graph regularization term used in GRF [15], we can get the objective with respect to \( a_k \) as follows:

\[ \mathcal{L}(f) = \frac{1}{2} \sum_{i,j} W_{ij} \left( f(x_i) - f(x_j) \right)^2 + \mu \sum_{i,j} W_{ij} \left( \sum_{k=1}^{m} z_{ik} a_k - \sum_{k=1}^{m} z_{jk} a_k \right) \]

The optimal solution is given as \( a = (Z^T L Z + \mu Z^T L Z)^{-1} Z^T y \). Further noticeable extensions of AGR include EAGR [76], which exploits the normalized Laplacian for effectiveness, and HARG [77], which utilizes the hierarchical anchor graph for efficiency.

5) Label Prediction Algorithm via DGL: The above three methods [15]–[17] all prove to be ineffective for handling ambiguous examples [99]. Gong et al. [29] introduced deformed graph Laplacian (DGL) and provided the corresponding label prediction algorithm via DGL (LPDGL) for SSL. A new smoothness term that considers local information is added to the regularizer. The graph regularizer becomes (22) as

\[ \mathcal{L}(f) = \frac{1}{2} \alpha \sum_{i,j} W_{ij} \left( f(x_i) - f(x_j) \right)^2 + \frac{1}{2} \beta \sum_{i=1}^{n} \left( 1 - \frac{D_{ii}}{\sum_{j=1}^{n} D_{ij}} \right) (f(x_i))^2 \]

where both \( \alpha \) and \( \beta \) are tradeoff parameters. It has been shown by theoretical analysis that LPDGL achieves a globally optimal prediction function. In addition, the performance is robust to the hyperparameters setting, so this model is not difficult to fine-tune.

6) Poisson Learning: The most recent work under the regularization framework is called Poisson learning in [53], which is motivated by the need to address the degeneracy of previous graph regularization methods when the label rate is meager. The new proposed approach replaces the given label values with the assignment of sources and sinks like flow in the graph. Thus, a resulting Poisson equation based on the graph can be nicely solved. The graph regularizer of Poisson learning is shown in the following equation:

\[ \mathcal{L}(f) = \frac{1}{2} \sum_{i,j} W_{ij} \left( f(x_i) - \sum_{j \in N(i)} f(x_j) \right)^2 \] (23)

C. Summary

The LP-based methods are accepted as mainstream methods during the early development stage of GSSL due to their simplicity, efficiency, and theoretical soundness. The basic framework of LP is a simple loop and undoubtedly easy to implement. Furthermore, LP reduces the time complexity from \( O(|V|^3) \) in the closed-form solution down to \( O(|V|^2) \) when the iteration number is fixed, while it also guarantees that the converged result is still equivalent to the closed-form solution. We summarize representative LP-based methods in Tables III and IV.

Meanwhile, the general-regularization-based methods can be interpreted as the extension of previous LP-based methods by discarding its iterative paradigm and focusing on the more expressive smoothness regularization term to satisfy the manifold assumption in GSSL. However, since most of the corresponding closed-form solutions involve the computation of the inverse of a \(|V|\)-dimensional matrix, the total time complexity rises back to \( O(|V|^3) \) in general. We summarize popular general-regularization-based methods in Table V.

V. GRAPH EMBEDDING

Generally speaking, there are two types of graph embedding at two levels commonly seen in the literature. The first one is at the entire graph level, while the second one is at the single node level [100]. Both of them aim to represent the target object in a low-dimensional vector space. For GSSL, we focus on node embeddings since such representations can be easily used for SSL tasks. The main objective of node embedding is to encode the nodes as vectors with lower dimensions, which in the meantime can reflect their positions and the structure of their local neighborhood.

Formally, we have the following definition for node embedding on graphs. Given a graph \( G = (V, E) \), a node embedding on it is a mapping \( f : \nu \rightarrow \mathbb{R}^d, \forall \nu \in \mathbb{V} \) such that \( d \ll |V| \) and the function \( f \) preserves some proximity measure defined on graph \( G \). The generalized form of the loss function for graph embedding methods is shown in the following equation:

\[ \mathcal{L}(f) = \sum_{(x_i, y_i) \in D_i} \mathcal{L}_i(f(f(x_i)), y_i) + \mu \sum_{x_i \in D_D + D_u} \mathcal{L}_i(f(f(x_i))) \] (24)

where \( f \) is the embedding function. It is obvious that (24) is almost the same as (5) for graph regularization except that for
graph embedding methods, classifiers are trained based on the nodes’ embedding results rather than nodes’ attributes directly. Following the generalization methods on graph representation learning by Hamilton et al. [100], all the node embedding methods mentioned in this section can be generalized under an encoder–decoder framework. First, an encoder model $\text{Enc}: \mathcal{V} \rightarrow \mathbb{R}^d$ tries to map every node $v \in \mathcal{V}$ into a low-dimensional vector $z_v \in \mathbb{R}^d$. Second, a decoder model $\text{Dec}: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$ is constructed to take the low-dimensional node embeddings as input and use them to reconstruct the structural information in the original graph, such as an adjacency matrix. For example, given a node embedding $z_u$ of the node $u$, the decoder attempts to predict $u$’s set of neighbors $\mathcal{N}(u)$. Decoders are often defined in a pairwise form, which can be illustrated as predicting each pair of nodes’ similarity. The overall goal is to solve an optimization problem that minimizes the reconstruction loss so that the similarity measures produced by the decoder are as close to the ones defined in the original graph as possible. In a more formal way, we have

$$\text{Dec}((\text{Enc}(u), \text{Enc}(v))) = \text{Dec}(z_u, z_v) \approx S[u, v].$$

(25)

Here, we assume that $S[u, v]$ is a certain kind of similarity measure between a pair of nodes such as the entry in the adjacency matrix $A[u, v]$. To achieve the reconstruction objective (25), the standard practice is to minimize the empirical reconstruction loss $\mathcal{L}$.

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \ell(\text{Dec}(\text{Enc}(u), \text{Enc}(v)), S[u, v])$$

(26)

where $\ell: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function for every node pair to compute the inconsistency between the true similarity values and the decoded ones.

In most of the work on node embedding, the encoder can be classified into a shallow embedding approach, in which this encoder is simply a lookup function based on the node ID.
In addition, the encoder can use both node features and the local graph structure around each node as the input to generate an embedding, such as GNNs. These methods are further categorized into the deep embedding method.

A. Shallow Graph Embedding

Some specialized optimization methods based on matrix factorization can be employed as a deterministic way to solve the optimization problem (26). Generally speaking, the whole task can be considered as using matrix factorization methods to learn a low-dimensional approximation of a similarity matrix \( S \), where \( S \) encodes the information related to the original adjacency matrix or other matrix measurements. Unlike the deterministic factorization methods, recent years have witnessed a surge in successful methods that use stochastic measures of neighborhood overlap to generate shallow embeddings. The key innovation in these approaches is that node embeddings are optimized under the assumption that if two nodes in the graph co-occur on some short-length random walks with high probability, they tend to share similar embeddings [101]. For ease of reading, we list some pros and cons of these two types of methods in Table VI.

| Factorization-based Methods                  | Advantages                                      | Disadvantages                                      |
|----------------------------------------------|------------------------------------------------|---------------------------------------------------|
| [13] [14] Scales with the embedding dimension size. | Unable to handle non-convex manifolds             | Unstable on outliers and noise                     |
| Efficient with distributed framework         | Loses accuracy in preserving global structure    | Expensive computation for SVD                      |
| [30] [32] Captures high-order global structural information | Strong assumption for inductive setting          | More parameters compared with factorization        |
| Performs random walks on hierarchical graphs to capture higher-order information | Time-consuming compared with other random-walk-based methods |       |

Adding another two constraints \( \frac{1}{2} Y^T Y = I \) and \( \sum_{i} Y_i = 0 \) into the above optimization equation, translational invariance can be eliminated since the embedding is forced to be around the origin. It has been proven that the solution to this problem is to compute all the eigenvectors of the sparse matrix \((I - W)^T(I - W)\), sort the corresponding eigenvalues in the descending order, and take the first \( d \) eigenvectors as the final embedding result.

b) Laplacian eigenmaps: This makes strongly connected nodes close to each other in the embedding space [14]. Unlike the LLE [13], the objective function is designed in a pairwise manner

\[
\phi(Y) = \frac{1}{2} \sum_{i,j} |Y_i - Y_j|^2 W_{ij} = \text{tr}(Y^T L Y) \tag{29}
\]

where \( L \) is the Laplacian matrix. Similar to LLE [13], it is necessary to add another constraint \( Y^T D Y = I \) so that some trivial solutions can be removed. The optimal solution is achieved by choosing the eigenvectors of the normalized Laplacian matrix whose corresponding eigenvalues are among the \( d \) smallest ones.

c) Graph factorization: Graph factorization (GF) [20] is the first algorithm to reduce the time complexity of previous graph embedding algorithms to \( O(E) \). Instead of targeting at factorizing Laplacian matrix such as LLE [13] and Laplacian Eigensmaps [14], GF directly employs the adjacency matrix and minimizes the objective function

\[
\phi(Y, \mu) = \frac{1}{2} \sum_{(i,j) \in E} (W_{ij} - (Y_i, Y_j))^2 + \frac{\mu}{2} \sum_i ||Y_i||^2 \tag{30}
\]

where \( \mu \) is a hyperparameter for the introduced regularization term. Because the adjacency matrix is not necessarily positive semidefinite, the summation over all the observed edges can be regarded as an approximation for the sake of scalability.

d) GraRep: It utilizes the node transition probability matrix [30] that is defined as \( T = D^{-1} W \) and \( k \)-order proximity is preserved by minimizing the loss \( ||X^k - Y_{s} X^{k-1} Y_{s}^T||_F^2 \), where \( X^k \) is derived from \( T^k \) and \( Y_s \) and \( Y_r \) are source and target embedding vectors, respectively. It then concatenates \( Y_k^s \) for all \( k \) to form \( Y_s \). The drawback of GraRep is scalability since \( T_k \) can have \( O(|V|^2) \) nonzero entries.

e) HOPE: Similar to GraRep [30], HOPE [32] preserves higher order proximity by minimizing another objective function \( ||S - Y_s Y^T_s||_F^2 \), where \( S \) is now the proximity matrix. The similarity measurement is defined in the form of \( S = M_{l}^{-1} M_{s} \).
where $M_q$ and $M_t$ are both sparse matrices. In this fashion, singular value decomposition (SVD) can be applied so as to acquire node embeddings in an efficient manner.

2) Random-Walk-Based Methods: The random walk is a powerful tool to gain approximate results about certain properties of the given graph, such as node centrality [102] and similarity [103]. Consequently, random-walk-based node embedding methods are effective under some scenarios when only part of the graph is accessible or the graph’s scale is too large to handle efficiently.

a) DeepWalk: Inspired by the skip-gram model [104], DeepWalk [28] follows the main goal of HOPE [32] and thus preserves higher order proximity of each node pair. However, DeepWalk takes another approach by maximizing the possibility of encountering the previous $k$ nodes and the following $k$ nodes along one specific random walk with center $v_i$. In other words, DeepWalk maximizes the log-likelihood function, which is defined as $\log \Pr(v_i \mid v_{i-k}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+k} \mid Y_t)$, where $2k+1$ is the length of the random walk. The decoder is a basic form of a dot product to reconstruct graph information from the encoded node embeddings.

b) Planetoid: It is another GSSL method based on the random walk [33], where the embedding of a node is jointly trained to predict the class label and also the context in the given graph. The highlight of Planetoid is that it can be trained to predict the class label and also the context in the given graph. The random walk is a function of input feature vectors, while the transductive variant only embeds graph structure information.

c) Node2Vec: Following the same idea of DeepWalk [28], node2vec [105] also tries to preserve higher order proximity for each node pair but makes full use of biased random walks so that it can balance between the breadth-first search (BFS) and depth-first search (DFS) on the given graph to generate more expressive node embeddings. To be more specific, many random walks with fixed length are sampled, and then, the possibility of occurrence of subsequent nodes along these biased random walks is maximized.

d) LINE: Previously mentioned methods do not scale in large real-world networks and LINE [31] is therefore introduced to fix this issue by preserving both local and global graph structures with scalability. In particular, LINE combines the first-order and the second-order proximity, and they are optimized using the Kullback–Leibler (KL) divergence metric. A decoder based on the sigmoid function is used in the first-order objective, while another decoder identical to the one in node2vec and DeepWalk is used in the second-order objective. Unlike node2vec and DeepWalk, LINE explicitly factorizes proximity measurement instead of implicitly incorporating it with sampled random walks.

e) HARP: It is a general strategy [43] to improve the abovementioned solutions [28], [31], [105] by avoiding local optima with the help of better weight initialization. The hierarchy of nodes is created by node aggregation in HARP using a graph coarsening technique based on the preceding hierarchy layer. After that, the new embedding result can be generated from the coarsen graph, and the refined graph (i.e., the graph in the next level up in the hierarchy) can be initialized with the previous embedding. HARP propagates these node embeddings level by level so that it can be used in combination with random-walk-based approaches in order to achieve better performance.

3) Summary: Table VII applies the encoder–decoder perspective to summarize some representative shallow embedding approaches. The similarity function $p_G(v \mid u)$ corresponds to the probability of visiting $v$ on a fixed-length random walk starting from $u$. Even though shallow embedding methods can be divided into two groups based on whether it is deterministic or stochastic, random-walk-based methods can actually be transformed into the factorization-based group in general. Qiu et al. [106] provided a theoretical analysis of the aforementioned random-walk-based methods to show that they all essentially perform implicit matrix factorization and have closed-form solutions. Although shallow embedding methods have achieved impressive success on many SSL-related tasks by addressing previously mentioned drawbacks of graph-regularization-based methods such as expensive cubic running time, it is worth noting that it introduces two common disadvantages as well. The critical problem with shallow embedding approaches is
that they fail to leverage node features. However, rich feature information could be informative in the encoding process. This is especially true for SSL tasks where each node represents valuable feature information. Another potential issue is that most shallow embedding methods are inherently transductive and challenging to adapt for inductive applications.

B. Deep Graph Embedding

In recent years, a great number of deep embedding approaches have been proposed, which differ from the shallow embedding approaches in which a much more complex encoder based on deep neural networks (DNNs) [109] is constructed and employed. In this manner, the encoder module would incorporate both the structural and attribute information of the graph. For SSL tasks, a top-level classifier needs to be trained to predict class labels for unlabelled nodes under the transductive setting based on the node embeddings generated by these deep learning models. For the ease of reading, we list some pros and cons of the reviewed methods in Table VIII.

1) AutoEncoder-Based Methods: Apart from the use of deep learning models, autoencoder-based methods also vary from the shallow embedding methods in which a unary decoder is employed instead of a pairwise one. Under the framework of autoencoder-based methods, every node, \( i \), is represented by a high-dimensional vector extracted from a row in the similarity matrix, namely, \( s_i = i \)th row of \( S \), where \( S_{ij} = s_{G}(i, j) \). The autoencoder-based methods aim to first encode each node based on the corresponding vector \( s_i \) and then reconstruct it again from the embedding results, subject to the constraint that the reconstructed one should be as close to the original one as possible (Fig. 5)

\[
\text{Dec} (\text{Enc}(s_i)) = \text{Dec}(z_i) \approx s_i. \tag{31}
\]

From the perspective of the loss function for autoencoder-based methods, it commonly keeps the following form:

\[
\mathcal{L} = \sum_{i \in V} \| \text{Dec}(z_i) - s_i \|^2_2. \tag{32}
\]

From (31), it should be pointed out that the encoder module actually depends on the given \( s_i \) vector. This allows autoencoder-based deep embedding approaches to incorporate local structural information into the encoder, while it is simply impossible for the shallow embedding approaches to do so. The primary components of these methods are summarized in Table IX.

![Fig. 5. For AutoEncoder-based methods, a high-dimensional vector \( s_i \) is extracted and fed into the AutoEncoder for generating a low-dimensional \( z_i \) embedding.](image-url)

Despite this noticeable enhancement, the autoencoder-based methods may still suffer from some problems. In particular, the computational cost of it is still intolerable for large-scale graphs. Moreover, the structure of the autoencoder is predefined and unchanged during the training, so it is strictly transductive and thus fails to cope with evolving graphs. The up-to-date, relevant representative works to tackle these issues are [110] and [111].

a) Structural deep network integration: Structural deep network integration (SDNE) [34] is developed with the help of deep autoencoders to preserve the proximity for first and second orders. The first-order proximity describes the similarity between each node pair, while the second-order proximity between each node pair describes the proximity of their neighborhood structure. The method takes advantage of nonlinear functions to acquire the embedding results. The unsupervised part is an autoencoder designed to produce an embedding result for each node that can be used to rebuild the similarity measure \( s_i \). For the supervised part, Laplacian Eigenmaps is utilized so that a penalty is imposed if connected nodes are encoded far away in the embedding space.

b) DNNs for Learning Graph Representations: DNNs for learning graph representations (DNGR) [35] integrates random surfing with autoencoders to generate node embeddings. This model has three components: 1) random surfing; 2) estimation of positive pointwise mutual information (PPMI) matrix; and 3) stacked denoising autoencoders. For the input graph, random surfing is first applied to produce a co-occurrence probability matrix similar to HOPE. This probabilistic matrix is then converted into a PPMI matrix and fed into a stacked denoising autoencoder to generate the final
embedding result. The feedback of the PPMI matrix guarantees that the high-order proximity is captured and maintained by the autoencoder. Moreover, the introduction of stacked denoising autoencoders enhances the model’s robustness to noise. c) S2S-AE: Unlike previous methods whose encoders are all based on MLP, other models manage to extend the form of the encoder to recurrent neural network (RNN) models. S2S-AE [44] uses long short-term memory (LSTM) [112] autoencoders to embed the graph sequences generated from random walks into a continuous vector space. The final representation is computed by averaging its graph sequence representations. The advantage of S2S-AE is that it can support arbitrary-length sequences, unlike others, which often suffer from the limitation of the fixed-length inputs.

d) Deep recursive network embedding: Deep recursive network embedding (DRNE) [45] holds an assumption that the embedding of a node needs to approximate the aggregation of the embeddings of nodes within its neighborhood. It also uses LSTM [112] to aggregate a node’s neighbors, so the reconstruction loss is different from the one in S2S-AE [44]. In this way, DRNE can solve the issue that the LSTM model is not invariant when the given nodes’ sequence permutes in different ways.

e) GAE and VGAE: Both MLP-based and RNN-based methods only consider structural information and ignore the nodes’ feature information. Graph auto-encoder (GAE) [36] leverages GCN [39] to encode both. The encoder takes the form that

$$\text{Enc}(A, X) = \text{GraphConv}(\sigma(\text{GraphConv}(A, X)))$$

where GraphConv() is a graph convolutional layer defined in [39], $\sigma(\cdot)$ is the activation function, $A$ is the adjacency matrix, and $X$ is the attribute matrix. The decoder of GAE is defined as $z_f^T z_o$.

It may have some overfitting issue when the adjacency matrix is reconstructed in a direct way. Variational GAE (VGAE) [36] learns the distribution of data, in which the evidence lower bound (ELBO) $L$ is optimized

$$L = \mathbb{E}_q(z|A)[log p(A | Z)] - KL[q(Z | A) || p(Z)]$$

where $KL[q(\cdot) || p(\cdot)]$ is the Kullback–Leibler divergence between $q(\cdot)$ and $p(\cdot)$.

f) ARGA and ARVGA: To further improve the empirical distribution, $q(Z | X, A)$ in accordance with the prior distribution $p(A | Z)$ in GAE and VGAE, Pan et al. [49] proposed ARGA and ARVGA with the help of the generative adversarial networks (GANs) [113], in which they take GAE and VGAE as encoder, respectively.

2) GNN-Based Methods: GNN, which is heavily utilized in state-of-the-art deep embedding approaches, is considered as a general scheme for defining DNNs in graph-structured data. The main idea is that the representation vectors of nodes can depend not only on the structure of the graph but also on any feature information associated with the nodes. Like other deep node embedding methods, a classifier is trained on top of the node embeddings generated by the final hidden state in GNN-based models explicitly or implicitly.

As pointed out in [114], the fundamental feature of a basic GNN is that it takes advantage of neural message passing framework in which messages are exchanged and updated between each pair of the nodes by using neural networks. More specifically, during each neural message passing iteration in a basic GNN, a hidden embedding $h_u^{(k)}$ corresponding to each node $u$ is updated according to message or information aggregated from $u$’s neighborhood $\mathcal{N}(u)$ (see Fig. 6). This general message passing update rule can be expressed as follows:

$$h_u^{(k+1)} = \text{Update}^{(k)}(h_u^{(k)}, \text{Aggregate}^{(k)}(\{h_v^{(k)} : \forall v \in \mathcal{N}(u)\}))$$

$$= \text{Update}^{(k)}(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)})$$

TABLE IX

| Method       | Learning Setting | Encoder | Decoder | Similarity Measure | Loss Function | Time Complexity | Description |
|--------------|------------------|---------|---------|--------------------|---------------|-----------------|-------------|
| SDNE [34]    | Transductive     | MLP     | MLP     | $s_u$              | $\sum_{u \in V} ||\text{Dec}(s_u) - s_u||^2$ | $\mathcal{O}(|V||E|)$ | First Autoencoder-based deep GSSL model |
| DNGR [35]    | Transductive     | MLP     | MLP     | $s_u$              | $\sum_{u \in V} ||\text{Dec}(s_u) - s_u||^2$ | $\mathcal{O}(|V|^2)$   | Reconstructs PPMI matrix by denoising Autoencoder |
| S2S-AE [44]  | Transductive     | LSTM    | LSTM    | $s_u$              | $\sum_{u \in V} ||\text{Dec}(s_u) - s_u||^2$ | $\mathcal{O}(|V|^2)$   | Uses LSTM Autoencoder |
| DRNE [45]    | Both             | LSTM    | LSTM    | $s_u$              | $\sum_{u \in V} \left(\sum_{v \in \mathcal{N}(u)} \text{LSTM}(s_v)^2\right)$ | $\mathcal{O}(|V||E|)$ | Reconstructs the node from its neighbors via LSTM |
| GAE [36]     | Transductive     | GCN     | $A_{uv}$| $A_{uv}$           | $\sum_{u \in V} ||\text{Dec}(s_u) - A_{uv}||^2$ | $\mathcal{O}(|V||E|)$ | Reconstructs adjacency matrix via GCN directly |
| VGAE [36]    | Transductive     | GCN     | $A_{uv}$| $A_{uv}$           | $\mathcal{E}(Z|X,A) \log p(A | Z)$ - $\mathcal{K}(p(Z|X,A) || p(Z))$ | $\mathcal{O}(|V||E|)$ | Improves GAE with GAN |
| ARGA [49]    | Transductive     | GAE     | $A_{uv}$| $A_{uv}$           | $\log \max_{p(z)} \mathbb{E}_{z \sim p_z}[\log D(Z)] + \mathbb{E}_{z \sim p_z}[\log (1 - D(G(X, A)))]$ | $\mathcal{O}(|V||E|)$ | Improves GAE with GAN |
| ARVGA [49]   | Transductive     | VGAE    | $A_{uv}$| $A_{uv}$           | $\log \max_{p(z)} \mathbb{E}_{z \sim p_z}[\log D(Z)] + \mathbb{E}_{z \sim p_z}[\log (1 - D(G(X, A)))]$ | $\mathcal{O}(|V||E|)$ | Improves VGAE with GAN |

Fig. 6. GNN-based methods can generate node embeddings by aggregating embeddings from its neighbors.
It is noteworthy that in (35), both the operation \textbf{Update} and \textbf{Aggregate} must be differentiable functions, typically, neural networks. Moreover, \(m_{N(u)}\) is the exact message that is aggregated from node \(u\)’s neighborhood \(N(u)\). Combining the message from neighborhood with the previous hidden embedding state, the new state is generated according to (35). After a certain preset number of iterative steps, the last hidden embedding state converges so that this final state is regarded as the embedding output for each node. Formally, we have \(z_u = h^{(K)}_u, \forall u \in V\). It should be stressed that both the basic GNN and many of its variants follow this generalized framework.

The basic GNN message passing [19] is defined as follows:

\[
\begin{align*}
    h^{(k)}_u &= \sigma \left( W^{(k)}_{self} h^{(k-1)}_u + W^{(k)}_{neigh} \sum_{v \in N(u)} h^{(k-1)}_v + b^{(k)} \right)
\end{align*}
\]  

(36)

where \(W^{(k)}_{self}, W^{(k)}_{neigh}\) are trainable parameters and \(\sigma\) is the activation function. The messages from the neighbors \(m_{N(u)} = \sum_{v \in N(u)} h_v\) are first summarized. Then, the neighborhood information is combined together with the node’s previous hidden embedding results by using a basic linear combination. Finally, a nonlinearity activation function is applied on the combined information.

Next, some representative improvements on the two main components of basic GNN, aggregation operation and update operation, are reviewed in detail.

a) \textbf{Neighborhood normalization}: As previously stated, the most basic neighborhood aggregation operation solely computes the sum of the neighborhood’s embedding states. The main problem with this approach is that it could be unstable and susceptible to the node’s degree since nodes with a large degree tend to receive a large total value from more neighbors than those with fewer neighbors. One typical and simple solution to this issue is to normalize the aggregation operation based on the degree of the central nodes. The simplest approach is to take an average rather than the sum, but methods with other normalization factors with similar ideas were proposed and achieved remarkable performance gain, such as the following symmetric normalization employed in the graph convolutional network (GCN) [39] as shown in the following equation:

\[
\begin{align*}
    m_{N(u)} &= \sum_{v \in N(u)} \frac{h_v}{\sqrt{|N(u)| \cdot |N(v)|}}.
\end{align*}
\]  

(37)

There exist a great number of GCN variants to enhance the SSL performance from different aspects. Li et al. [107] are the first to provide deep insights into GCN’s success and failure on SSL tasks. Later on, extensions to GCN for SSL begin to proliferate. Jiang et al. [115] explored the way to do graph construction based on GCN. Yang et al. [108] combined the classic graph regularization methods with GCN. Other research work on GCN extensions for SSL can be found in [116]–[119].

GCNs inherit unnecessary complexity and redundant computation cost in nature as it derives inspiration from deep learning methods. SGC [50] reduces this excess complexity by eliminating the nonlinearities among every GCN layer and collapsing the original nonlinear function into a simple linear mapping function. More importantly, these simplifications do not harm the prediction performance in many downstream applications.

b) \textbf{Advanced pooling}: Aggregation operation is essentially a mapping from a set of neighborhood embedding results to a single vector with encoded information about the local structure and the feature of neighbor nodes’ feature. In the previously reviewed settings of GNN models, the mapping function in the aggregation operation is simply the basic summation or linear functions over neighbor embeddings.

In fact, according to [120], one principal approach for designing an aggregation function is focused on the theory of permutation-invariant neural networks. Permutation invariance on graphs in general means that the aggregation function does not depend on the arbitrary order of the rows/columns in the adjacency matrix. For example, set pooling [40] shows that an aggregation function with the following form can be considered as a universal set function approximator:

\[
\begin{align*}
    m_{N(u)} &= \text{MLP}_\theta \left( \sum_{v \in N(u)} \text{MLP}_\phi(h_v) \right).
\end{align*}
\]  

(38)

Another alternative method, called Janossy pooling [51] uses an entirely different approach. Instead of using a permutation-invariant reduction (e.g., a sum or a mean), a permutation-sensitive function is applied, and the outcome is averaged over many potential permutations. Let \(\pi_i \in \Pi\) denote a permutation function that maps the set \(\{h_v : \forall v \in N(u)\}\) to a specific sequence \(\{h_{\pi_i(1)}, h_{\pi_i(2)}, \ldots, h_{\pi_i(|N(u)|)}\}\). Namely, \(\pi_i \in \Pi\) takes the unordered set of embedding states from the neighbors and puts them in a sequence dependent on some random ordering arbitrarily. The Janossy pooling approach then performs neighborhood aggregation operation by

\[
\begin{align*}
    m_{N(u)} &= \text{MLP}_\theta \left( \frac{1}{|\Pi|} \sum_{\pi_i \in \Pi} \rho_{\phi}(h_{\pi_i(1)}, h_{\pi_i(2)}, h_{\pi_i(|N(u)|)})(\pi_i) \right)
\end{align*}
\]  

(39)

where \(\Pi\) denotes a collection of permutations and \(\rho_{\phi}\) is a permutation-sensitive function, e.g., a neural network that operates on sequential data such as LSTM.

c) \textbf{Neighborhood attention}: A common approach for enhancing the aggregation layer in GNNs is to implement some attention mechanisms [121], in addition to more general forms of set aggregation. The basic principle is to assign a weight or value of importance to each neighbor, which is used during the aggregation phase to weigh this neighbor’s effect. The first GNN model to apply this style of attention is graph attention network (GAT) [46], which uses attention weights to define a weighted sum of the neighbors

\[
\begin{align*}
    m_{N(u)} &= \sum_{v \in N(u)} \alpha_{u,v} h_v
\end{align*}
\]  

(40)

where \(\alpha_{u,v}\) denotes the attention on neighbor \(v \in N(u)\) when we are aggregating information at node \(u\). In the original GAT paper, the attention weights are defined as

\[
\begin{align*}
    \alpha_{u,v} &= \frac{\exp(a^T[W h_u + W h_v])}{\sum_{v' \in N(u)} \exp(a^T[W h_{u} + W h_{v'}])}
\end{align*}
\]  

(41)

where \(a\) is a trainable attention vector, \(W\) is a trainable matrix, and \(\oplus\) denotes the concatenation operation.

d) \textbf{Concatenation}: Oversmoothing is a major issue for GNN. The oversmoothing is almost inevitable after many message iterations when the node-specific information becomes “washed away.” In such cases, the modified node representations are too highly dependent on the incoming message.
aggregated by the neighbors at the cost of previous layers’ node hidden states. One reasonable way to mitigate this problem is to use vector concatenations or skip connections, which aim to retain information directly from previous rounds of the update. For general purposes, Update\(_{\text{base}}\) denotes the simple update rule that will be built on.

One of the simplest updates for skip connection is GraphSAGE [41], which uses a concatenation vector to hold more information from node level during message passing process

\[
\text{Update}(h_u, m_N(u)) = [\text{Update}_{\text{base}}(h_u, m_N(u)) \oplus h_u]
\]  

(42)

where the output from the simple update function is concatenated with the node’s previous layer representation. The core intuition is that the model is encouraged to dissociate information during the message passing. Besides concatenation methods, some other forms of skip connections can also be applied, such as the linear interpolation method proposed on column network (CLN) [42].

e) Gated updates: Parallel to the abovementioned work, the researchers have also taken inspiration from the approaches used by RNNs to strengthen stability. One way to interpret the GNN message passing algorithm is to collect an observation from the neighbors from the aggregation operation, which then updates the hidden state of each node. From this perspective, some methods for updating the hidden status of RNN architectures can be directly applied based on the observation. For example, one of the earliest GNN variants, which put this idea into practice, is proposed in GatedGNN [37], in which the update operation is defined as shown in the following equation:

\[
h_u^{(k)} = \text{GRU}\left(h_u^{(k-1)}, m_N^{(k)}\right)
\]  

(43)

where GRU is a gating mechanism function in RNNs [122]. Another approach NeuroSAT [52] has employed updates based on the LSTM architecture as well.

f) Jumping knowledge (JK) connections: In Sections V-B2a–V-B2e, it is implicitly assumed that the last layer’s output is considered as the final embedding result. In other words, the node representations used for a downstream job, such as SSL tasks, are identical to the final layer’s node embedding in the GNN. A complementary approach to increase the effectiveness of final node representations is to use the combination of layer on the message passing, rather than merely the final layer’s output. In a more formal way

\[
z_u = f_{\text{JK}}(h_u^{(0)} \oplus h_u^{(1)} \oplus \ldots \oplus h_u^{(K)})
\]  

(44)

This technique is originally introduced and tested in JK Net [47]. The \(f_{\text{JK}}\) function can be used as the identity function for various applications, meaning that a simple concatenation is essentially performed by the embeddings learned by popular supervised learning methods over others is that they consider nodes’ attributes for generating embeddings so that the inconsistency in the proceeding graph construction step can be alleviated to some degree compared to AutoEncoder-based methods. However, both these two groups of deep graph embedding methods are initially targeted for graph-structured data such as social networks, and thereby, the graph construction step is no longer needed. The potential influence of these methods on the general Euclidean data is still yet unexplored.

VI. APPLICATIONS

A. Datasets and Implementations

We summarize the commonly used datasets in GSSL according to seven domains: citation networks, copurchase networks, web page citation networks, and others. As shown in Appendix B of the supplementary material, the summary results on selected benchmark datasets on GSSL are listed with their respective statistical analysis. We also list some open-source implementations for GSSL in Appendix C of the supplementary material.

B. Domains

GSSL has a large number of successful applications across various domains. Some domains have explicit graph-structured data in nature already, while others do not. The former would be scenarios where raw data samples have relational structure and can be easily constructed into a graph, such as traffic networks in the cyber-physical systems (CPS), molecular structure in biomedical engineering, and friend recommendation in social networks. From the nongraph-structured data, however, a graph cannot be extracted directly. Typical examples would be more common scenarios, such as image
classification in computer vision (CV) and text classification in traditional natural language processing (NLP). Therefore, the graph construction step is necessary for this scenario.

1) Computer Vision: Among many CV tasks, hyperspectral image classification (HSI) is a representative example for GSSL applications since labeled data in HSI are costly and scarce. Shao et al. [123] proposed a spatial and class structure regularized sparse representation graph for semi-supervised HSI classification. Later, He et al. [124] extended this work [123] by providing a more scalable algorithm based on anchor graph [125].

2) Natural Language Processing: Subramanya et al. [126] first introduced LP [15] into traditional NLP tasks and made pioneering work on part-of-speech (POS) tagging. Later, several works [127], [128] extend it and use some GCN-based methods [39] to make the model more robust on other NLP tasks. More recent works on how to combine GSSL and NLP tasks center around graph smoothing problems [129]. Mei et al. [130] proposed a brand-new general optimization framework for smoothing language models with graph structures by constructing a similarity graph of documents and words.

3) Social Networks: A social network such as Twitter is a set of people with some pattern of interactions or “ties” between them and has graph-structured data explicitly. Alam et al. [131] adopted a graph-based deep learning framework by Yang et al. [132] for learning an inductive semi-supervised model to classify tweets in a crisis situation. Later, Balaanand et al. [133] improved classic GSSL methods to detect fake users from a large volume of Twitter data. Another popular topic in social networks associated with GSSL is POI recommendations, such as friend recommendation [134] and follower suggestion [125].

4) Biomedical Science: Graphs are also ubiquitous in the area of biomedical science, such as semantic biomedical knowledge graphs, molecular graphs for drugs, and protein-drug interaction for drug proposals. Torshizi and Petzold [135] used GSSL methods with genomic data integration to do phenotype classification tasks. In the meantime, Luo et al. [136] provided a new graph regularization framework in heterogeneous networks to predict human miRNA-disease [137].

VI. OPEN PROBLEM

According to the chronological overview of GSSL development over the last two decades in Fig. 4, it has witnessed several shifts of trend in this active research area [138]. Apart from the common issues like how to build scalable GSSL [139]–[141], we list three possibly promising open problems with the future outlook. Besides, some related pioneering works published in top-tier venues throughout this and last year (2020–2021) are also referred to support our insights and perspectives.

A. Generative Models for GSSL

Most of the reviewed methods are deterministic by learning the decision boundary in the latent space, which lacks interpretability and fails to function for outlier detection. However, generative models solve these issues by modeling the joint distribution of features and labels and using the Bayes rule to conduct inference. Unlike existing generative GNN models operating on the given fixed graph [36], [142], the challenge of incorporating generative models into GSSL is to find the appropriate graphical model to extract the relations between features and labels based on the generated affinity graph under the manifold assumption. GraphEBM [143] is the first to utilize the energy-based model for graph construction with the benefits of dynamic update and interpretable metrics. Furthermore, G3NN [144] is the first to apply a flexible generative model on the label inference step. By viewing the graph as a random variable, the generated joint distribution can extract more general relationships among attributes, labels, and the graph structure. Some latest follow-up works aiming to solve this uncertainty issue by means of generative models include BGCN [145] via a nonparametric Bayesian framework for the posterior distribution of adjacency matrices and S-BGCN-T-K [146] via a graph-based kernel Dirichlet distribution estimation (GKDE) method to predict reducecrtive errors.

B. Deep GSSL With Classical Methods

Even though GNN-based methods currently hold the dominant position in the research area due to the massive growth of deep learning, they are only limited to label inference step and performs well for the non-Euclidean data empirically, where the graph structure is already given without the manually delicate process of graph construction. How to design a GNN-based method specifically for the generated affinity graph of the general Euclidean data is still an open question. Therefore, it is promising to combine the heated deep-learning-based methods with the classical methods for GSSL so as to enjoy the benefits of both [147]. One latest work on graph construction [70] is to transform this task into a well-established matrix completion problem via a parameterized autoencoder-based method so that the affinity graph can be updated accordingly whenever the parameters of the succeeding label inference model are changed. More similar insights can be found in the label inference step. An easy yet effective solution is to first generate pseudo-labels for the unlabeled data by LGC [78] and then train a DNN for SSL [148]. More recently, it has been empirically shown that combining the conventional LP method with simple MLP can outperform many SOTA GNN-based methods [149]. Furthermore, it has also been theoretically proven that the traditional graph regularization method indeed shares a deep connection with the GCN model [150].

C. Resilient and Robust GSSL

Graphs with noise or missing attributes are also hot topics since the generated affinity graph could be suboptimal with wrong links or misleading edge weights. Moreover, most existing GSSL methods end up fully trusting the given few labels, but in real life, these labels are highly unreliable as they are often produced by humans who are prone to mistakes. Graph agreement model (GAM) [151] is presented for enhanced noise resilience by introducing an auxiliary model that predicts the probability of two nodes sharing the same label. GNM [152] also imputes missing nodes by estimating the nonignorable missingness. In addition, some theoretical insights [153] into this topic are given for practical guidance. Similar to noise-resilience, attack robustness is another increasingly popular concern in the field of GSSL due to the growing need for trustworthy machine learning models. The first general framework for data poisoning attacks to
GSSL is initiated recently [154]. While this framework [154] focuses more on how to generate a successful attack, latest works [155]–[157] defend GNN models from these adversarial attacks.

VIII. CONCLUSION

To sum up, we conduct a comprehensive review of GSSL. A new taxonomy is proposed, in which all the popular label inference methods are grouped into two main categories: graph regularization and graph embedding. Moreover, they can be generalized within the regularization framework and the encoder–decoder framework respectively. Then, a wide range of applications of GSSL are introduced along with relevant datasets, open-source codes for some of the GSSL methods. Finally, three open problems for future research directions are discussed as well.

REFERENCES

[1] A. Subramanya and P. P. Talukdar, "Graph-Based Semi-Supervised Learning (Synthesis Lectures on Artificial Intelligence and Machine Learning)," San Rafael, CA, USA: Morgan & Claypool, 2014.
[2] X. Zhu, "Semi-supervised learning literature survey," Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep., 1530, 2005.
[3] N. N. Pise and P. Kulkarni, "A survey of semi-supervised learning techniques," 2014, arXiv:1402.4645.
[4] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," Mach. Learn., vol. 109, no. 2, pp. 373–440, 2020.
[5] Y. Ouali, C. Hudelet, and M. Tami, "An overview of deep semi-supervised learning," 2020, arXiv:2006.05278.
[6] Y. Chong, Y. Ding, Q. Yan, and S. Pan, "Graph-based semi-supervised learning: A review," Neurocomputing, vol. 408, pp. 216–230, Sep. 2020.
[7] K. Ozaki, M. Shimbo, M. Komachi, and Y. Matsumoto, "Using the mutual k-nearest neighbor graphs for semi-supervised classification on natural language data," in Proc. CoNLL, 2011, pp. 154–162.
[8] T. Jebra, J. Wang, and S.-F. Chang, "Graph construction and b-matching for semi-supervised learning," in Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML), 2009, pp. 441–448.
[9] H. Cheng, Z. Liu, and I. Yang, "Sparsity induced similarity measure for label propagation," in Proc. IEEE 12th Int. Conf. Intell. Syst., Sep. 2009, pp. 317–324.
[10] L. Zhuang, H. Gao, Z. Lin, Y. Ma, X. Zhang, and N. Yu, "Non-negative low rank and sparse graph for semi-supervised learning," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2012, pp. 2328–2335.
[11] M. Karasuyama and H. Mamitsuka, "Manifold-based similarity adaptation for label propagation," in Proc. NIPS, 2013, pp. 1547–1555.
[12] T. S. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," Science, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.
[13] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in Proc. Adv. Neural Inf. Process. Syst., vol. 12, pp. 585–591.
[14] Z. Zhu, S. Ghaahramani, 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.
[15] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in Proc. NIPS. Cambridge, MA, USA: MIT Press, 2003, pp. 321–328.
[16] D. Zhou, J. Huang, and B. Schölkopf, "Learning from labeled and unlabeled data on a directed graph," in Proc. 22nd Int. Conf. Mach. Learn. (ICML), 2005, pp. 1036–1043.
[17] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," J. Mach. Learn. Res., vol. 7, pp. 2399–2434, Dec. 2006.
[18] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in Proc. 22nd Int. Conf. World Wide Web (WWW), 2013, pp. 37–48.
[19] F. Wang and C. Zhang, "Label propagation through linear neighborhoods," in Proc. 23rd Int. Conf. Mach. Learn. (ICML), 2006, pp. 985–992.
[20] F. Nie, S. Xiang, Y. Liu, and C. Zhang, "A general graph-based semi-supervised learning with graph class discovery," Neural Comput. Appl., vol. 19, no. 4, pp. 549–555, Jun. 2010.
[21] B. Wang, Z. Tu, and J. K. Tsotsos, "Dynamic label propagation for semi-supervised multi-class multi-label classification," in Proc. ICCV, Dec. 2013, pp. 425–432.
[22] M. Belkin, I. Matveeva, and P. Niyogi, "Tikhonov regularization and semi-supervised learning on large graphs," in Proc. IEEE Int. Conf. Signal Process., Speech, Signal Process., May 2004, pp. 1080–1083.
[23] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in Proc. ICML, 2010, pp. 1–8.
[24] L. Berton and A. D. A. Lopes, "Graph construction based on labeled instances for semi-supervised learning," in Proc. 22nd Int. Conf. Pattern Recognit., Aug. 2014, pp. 2477–2482.
[25] D. A. Vega-Oliveros, L. Berton, A. M. Eberle, A. D. A. Lopes, and L. Zhao, "Regular graph construction for semi-supervised learning," J. Mach. Learn. Res., vol. 19, no. 1, pp. 61–80, Dec. 2018.
L. Zhuang, Z. Zhou, S. Gao, J. Yin, Z. Lin, and Y. Ma, “Label F. Wang and C. Zhang, “Label propagation through linear neighbor-

F. Breve, L. Zhao, M. Quiles, W. Pedrycz, and J. Liu, “Particle compe-

C. A. R. de Sousa, S. O. Rezende, and G. E. A. P. A. Batista, “Influence F. Breve and L. Zhao, “Particle competition and cooperation in net-

F. Breve and L. Zhao, “Particle competition and cooperation for semi-supervised learning with label noise,” Neurocom-

F. Dornaika and Y. El Traboulsi, “Joint sparse graph and flexible P. Koliopoulos, A. Argyriou, and M. Pontil, “Learning and transfer-

J. Wang, F. Wang, C. Zhang, H. C. Shen, and L. Quan, “Linear K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, “Simplifying graph convolutional networks,” in Proc. 36th Int. Conf. Mach.

F. Breve and L. Zhao, “Particle competition and cooperation for semi-supervised learning,” in Proc. Int. Conf. Neural Netw. (IJCNN), Jun. 2012, pp. 1–6.

F. Breve, L. Zhao, and M. G. Quiles, “Particle competition and cooperation for semi-supervised learning with label noise,” Neurocomputing, vol. 160, pp. 63–72, Jul. 2015.

F. Dornaika and Y. El Traboulsi, “Joint sparse graph and flexible embedding for graph-based semi-supervised learning,” Neural Netw., vol. 114, pp. 91–95, Jun. 2019.

M. Kang, K. Lee, Y. H. Lee, and C. Suh, “Autoencoder-based graph F. Breve and L. Zhao, “Particle competition and cooperation for semi-supervised learning with concept drift,” in Proc. Int. Conf. Neural Netw. (IJCNN), Jun. 2012, pp. 1–6.

F. Breve and L. Zhao, “Particle competition and cooperation in networks for semi-supervised learning with concept drift,” in Proc. Int. Conf. Neural Netw. (IJCNN), Jun. 2012, pp. 1–6.

F. Breve and L. Zhao, “Particle competition and cooperation in networks for semi-supervised learning,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, no. 9, pp. 1608–1618, Sep. 2019.

F. Breve and L. Zhao, “Particle competition and cooperation in networks for semi-supervised learning,” in Proc. Int. Conf. Neural Netw. (IJCNN), Jun. 2012, pp. 1–6.

F. Breve and L. Zhao, “Particle competition and cooperation in networks for semi-supervised learning,” in Proc. Int. Conf. Neural Netw. (IJCNN), Jun. 2012, pp. 1–6.

F. Breve, L. Zhao, and M. Quiles, “Particle competition and cooperation for semi-supervised learning with label noise,” Neurocomputing, vol. 160, pp. 63–72, Jul. 2015.

F. Dornaika and Y. El Traboulsi, “Joint sparse graph and flexible embedding for graph-based semi-supervised learning,” Neural Netw., vol. 114, pp. 91–95, Jun. 2019.

M. Kang, K. Lee, Y. H. Lee, and C. Suh, “Autoencoder-based graph construction for semi-supervised learning,” in Proc. 22nd Int. Conf. Mach. Learn. (ICML), 2005, pp. 824–831.

M. Argyriou, M. Herbster, and M. Pontil, “Combining graph Laplacians for semi-supervised learning,” in Proc. NIPS, 2005, pp. 67–74.

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.

M. Wang, W. Fu, S. Hao, L. Liu, and X. Wu, “Learning on big graph: Label inference and regularization with anchor hierarchy,” IEEE Trans. Knowl. Data Eng., vol. 29, no. 5, pp. 1101–1114, May 2017.

D. Zhou and B. Schölkopf, “A regularization framework for learning from graph data,” in Proc. ICML Workshop Stat. Relational Learn. Connections Other Fields (SRL), 2004, pp. 132–137.

R. K. Ando and T. Zhang, “Learning on graph with Laplacian regular-ization,” in Proc. Adv. Neural Inf. Process. Syst., vol. 2007, pp. 25–32.

J. Calder and D. Slepčev, “Properly-weighted graph Laplacian for semi-supervised learning,” Appl. Math. Optim., vol. 82, no. 3, pp. 1111–1159, 2019.

F. Hoffmann, B. Hosseini, Z. Ren, and A. M. Stuart, “Consistency of semi-supervised learning algorithms on graphs: Probit and one-hot methods,” J. Mach. Learn. Res., vol. 21, no. 186, pp. 1–55, 2020.

X. Zhu and Z. Ghahramani, “Learning from labeled and unlabeled data with label propagation,” School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-ICALD-02-107, 2002.

F. Wang, Z. Zhang, M. Zhao, and T. W. S. Chow, “Graph based constrained semi-supervised learning framework via label propagation over adaptive neighborhood,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 9, pp. 2362–2376, Sep. 2015.

F. Nie, D. Xu, X. Li, and S. Xiang, “Semi-supervised dimensionality reduction and classification through virtual label regression,” IEEE Trans. Syst. Man, Cybern. B, Cybern., vol. 41, no. 3, pp. 675–685, Jun. 2011.

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.

L. Jia, Z. Zhang, L. Wang, J. Jiang, and M. Zhao, “Adaptive neighborhood propagation by joint $\ell_1,\ell_2$-norm regularized sparse coding for representation and classification,” in Proc. IEEE 16th Int. Conf. Data Mining (ICDM), Dec. 2016, pp. 201–210.

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.

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, Nov. 2020, pp. 4538–4552.

H. Rue and L. Held, Gaussian Markov Random Fields: Theory and Applications. Boca Raton, FL, USA: CRC Press, 2005.

Z. Xu, I. King, M. R.-T. Lyu, and R. Jin, “Discriminative semi-supervised feature selection via manifold regularization,” IEEE Trans. Neural Netw. Learn. Syst., vol. 21, no. 7, pp. 1033–1047, Jul. 2010.

A. Argyriou, C. A. Micchelli, and M. Pontil, “When is there a represen-tor theorem? Vector versus matrix regularizers,” J. Mach. Learn. Res., vol. 10, no. 2, pp. 529–552, 2009.

P. Niyogi, “Manifold regularization and semi-supervised learning: Some theoretical analyses,” J. Mach. Learn. Res., vol. 14, pp. 1229–1250, May 2013.
Z. Chen, H. Cao, and K. C.-C. Chang, “GraphEBM: Energy-based graph construction for semi-supervised learning,” in Proc. IEEE Int. Conf. Data Mining (ICDM), Nov. 2020, pp. 62–71.

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.

S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, “Non parametric graph learning for Bayesian graph neural networks,” in Proc. Conf. Uncertainty Artif. Intell., 2020, pp. 1318–1327.

X. Zhao, F. Chen, S. Hu, and J. Cho, “Uncertainty aware semi-supervised learning on graph data,” in Proc. NeurIPS, 2020, pp. 12827–12836.

Q. Li, X.-M. Wu, H. Liu, X. Zhang, and Z. Guan, “Label efficient semi-supervised learning via graph filtering,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 9582–9591.

A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Label propagation for deep semi-supervised learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 5070–5079.

Q. Huang, H. He, A. Singh, S. Lim, and A. R. Benson, “Combining label propagation and simple models out-performs graph neural networks,” in Proc. ICLR, 2021, pp. 1–21.

H. Yang, K. Ma, and J. Cheng, “Rethinking graph regularization for graph neural networks,” in Proc. AAAI, 2021, pp. 4573–4581.

O. Stretcu, K. Viswanathan, D. Movshovitz-Attias, E. Platanioti, S. Ravi, and A. Tomkins, “Graph agreement models for semi-supervised learning,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8713–8723.

F. Zhou, T. Li, H. Zhou, H. Zhu, and Y. Jieping, “Graph-based semi-supervised learning with non-ignorable non-response,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 7015–7025.

M. M. Dunlop, D. Slepčev, A. M. Stuart, and M. Thorpe, “Large data and zero noise limits of graph-based semi-supervised learning algorithms,” Appl. Comput. Harmon. Anal., vol. 49, no. 2, pp. 655–697, Sep. 2020.

X. Liu, S. Si, J. Zhu, Y. Li, and C. Hsieh, “A unified framework for data poisoning attack to graph-based semi-supervised learning,” in Proc. NeurIPS, 2019, pp. 9777–9787.

X. Gao, W. Hu, and Z. Guo, “Exploring structure-adaptive graph learning for robust semi-supervised classification,” in Proc. IEEE Int. Conf. Multimedia Expo. (ICME), Jul. 2020, pp. 1–6.

P. Elinas, E. V. Bonilla, and L. C. Tiao, “Variational inference for graph convolutional networks in the absence of graph data and adversarial settings,” in Proc. NeurIPS, 2020, pp. 18648–18660.

X. Zhang and M. Zitnik, “GNNGuard: Defending graph neural networks against adversarial attacks,” in Proc. NeurIPS, 2020, pp. 9263–9275.

Zixing Song received the B.Eng. degree in computer science and engineering from Southeast University, Nanjing, China, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests are machine learning on graphs, semi-supervised learning, and social computing.

Xiangli Yang is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China.

Her research interests are semi-supervised learning, data mining, and machine learning.

Zenglin Xu (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, in 2009.

He is currently a Full Professor with Harbin Institute of Technology, Shenzhen, China. He has been working with Michigan State University, East Lansing, MI, USA, and a Cluster of Excellence at the Max Planck Institute for Informatics, Saarland University, Saarbrücken, Germany, Purdue University, West Lafayette, IN, USA, and the University of Electronic Science and Technology of China, Chengdu, China. His research interests include machine learning and its applications in computer vision, health informatics, and natural language processing.

Dr. Xu was a recipient of the Outstanding Student Paper Honorable Mention of AAAI 2015, the Best Student Paper Runner Up of ACML 2016, and the 2016 Young Researcher Award from APNNS. He serves as an Associate Editor for Neural Networks and Neurocomputing.

Irwin King (Fellow, IEEE) received the B.Sc. degree in engineering and applied science from the California Institute of Technology (Caltech), Pasadena, CA, USA, in 1984, and the M.Sc. and Ph.D. degrees in computer science from the University of Southern California (USC), Los Angeles, CA, USA, in 1987 and 1993, respectively.

He is currently the Chair Professor of computer science and engineering at The Chinese University of Hong Kong, Hong Kong. In early 2010, while on leave with AT&T Labs Research, San Francisco, CA, USA, he taught classes as a Visiting Professor at the University of California at Berkeley, Berkeley, CA, USA. His research interests include machine learning, social computing, artificial intelligence (AI), and data mining. In these research areas, he has over 350 technical publications in journals and conferences.

Dr. King was a recipient of several Test of Time Awards, including ACM CIKM 2019, ACM SIGIR 2020, and ACM WSDM 2022, for his contributions made in social computing with machine learning. He has served as the President for the International Neural Network Society (INNS) and the General Co-Chair for WebConf 2020, ICONIP 2020, WSDM 2011, RecSys 2013, and ACM 2015. He served in various capacities in top conferences and societies, such as WWW, NIPS, ICML, IJCAI, AAAI, and APNNS. He is an Associate Editor of Neural Networks (NN). He is a Distinguished Member of ACM.