Semi-Supervised Node Classification on Graphs: Markov Random Fields vs. Graph Neural Networks

Binghui Wang, Jinyuan Jia, Neil Zhenqiang Gong
ECE Department, Duke University, Durham, NC 27708, USA
{binghui.wang, jinyuan.jia, neil.gong}@duke.edu

Abstract
Semi-supervised node classification on graph-structured data has many applications such as fraud detection, fake account and review detection, user’s private attribute inference in social networks, and community detection. Various methods such as pairwise Markov Random Fields (pMRF) and graph neural networks were developed for semi-supervised node classification. pMRF is more efficient than graph neural networks. However, existing pMRF-based methods are less accurate than graph neural networks, due to a key limitation that they assume a heuristics-based constant edge potential for all edges. In this work, we aim to address the key limitation of existing pMRF-based methods. In particular, we propose to learn edge potentials for pMRF. Our evaluation results on various types of graph datasets show that our optimized pMRF-based method consistently outperforms existing graph neural networks in terms of both accuracy and efficiency. Our results highlight that previous work may have underestimated the power of pMRF for semi-supervised node classification.

Introduction
Given an undirected graph, a feature vector for each node in the graph, and a small set of labeled nodes, semi-supervised node classification aims to classify the remaining unlabeled nodes simultaneously. Semi-supervised node classification has many applications such as fraud detection (Pandit et al. 2007; Chau et al. 2011; Ye et al. 2011; Tamersoy, Roundy, and Chau 2014; Hooi et al. 2016), fake account and review detection (Wang et al. 2011; Caò et al. 2012; Akoglu, Chandy, and Faloutsos 2013; Rayana and Akoglu 2015; Wang, Zhang, and Gong 2017; Wang, Gong, and Fu 2017; Gao et al. 2018; Wang et al. 2018), user’s private attribute inference in social networks (Gong et al. 2014; Jia et al. 2017; Gong and Liu 2018; Wang, Jia, and Gong 2019), and community detection (He et al. 2018; Jin et al. 2019a,b). Therefore, many methods have been proposed for semi-supervised node classification. These include conventional methods such as label propagation (LP) (Zhu, Ghahramani, and Lafferty 2003), manifold regularization (ManiReg) (Belkin, Niyogi, and Sindhwani 2006), deep semi-supervised embedding (SemiEmb) (Weston et al. 2012), iterative classification algorithm (ICA) (Lu and Getoor 2003), and pairwise Markov Random Fields (pMRF) (Gatterbauer 2017), as well as recent graph neural networks (Duvenaud et al. 2015; Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Battaglia et al. 2018; Xu et al. 2018; Gao, Wang, and Ji 2018; Ying et al. 2018; Veličković et al. 2018; Wu et al. 2019; Qu, Bengio, and Tang 2019; Ma et al. 2019a; Wang et al. 2020).

In particular, pMRF associates a discrete random variable with each node to model its label. pMRF defines a node potential for each node, which is a function of the node’s random variable and models the node’s label using the node’s feature vector. pMRF also defines an edge potential for each edge \((u, v)\), which is a function of the two random variables associated with nodes \(u\) and \(v\). Finally, pMRF defines a joint probability distribution of all random variables as the product of the node potentials and edge potentials. The edge potential for an edge \((u, v)\) is essentially a matrix, whose \((i, j)\)th entry indicates the likelihood that \(u\) and \(v\) have labels \(i\) and \(j\), respectively. Therefore, an edge potential is also called coupling matrix. Moreover, the standard Belief Propagation (Pearl 1988) or linearized Belief Propagation (Gatterbauer et al. 2015; Gatterbauer 2017; Jia et al. 2017) were often used to perform inference on the pMRF. Compared to graph neural networks, pMRF inferred via linearized Belief Propagation is much more efficient in terms of running time. However, existing pMRF-based methods (Pandit et al. 2007; Tamersoy, Roundy, and Chau 2014; Akoglu, Chandy, and Faloutsos 2013; Rayana and Akoglu 2015; Jia et al. 2017; Wang, Gong, and Fu 2017) are often less accurate than graph neural networks, due to a key limitation: they assume a heuristics-based constant edge potential or coupling matrix for all edges.

In this work, we aim to address the limitation of existing pMRF-based methods. In particular, we propose a novel method to learn the coupling matrix. Specifically, we formulate learning the coupling matrix as an optimization problem, where the objective function is to learn a coupling matrix such that 1) the training loss is small (modeled as cross-entropy loss) and 2) the coupling matrix of an edge \((u, v)\) is consistent with the predicted labels of nodes \(u\) and \(v\) (modeled as a regularization term). Moreover, we propose a novel algorithm to solve the optimization problem. Our results on various types of graph datasets show that our method is more accurate and efficient than existing graph neural networks.
Problem Definition and Background

Problem Definition

We aim to solve the following semi-supervised node classification problem on graph-structured data.

Definition 1. Suppose we are given an undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Each node $v$ has an $F$-dimensional feature vector $x_v$, and each node has a label from a label set $\{1, 2, \ldots, C\}$. Moreover, we are given a small set of labeled nodes $L$, which we call training dataset. Our goal is to predict the labels of the remaining unlabeled nodes using the nodes’ feature vectors and graph structure.

Pairwise Markov Random Field (pMRF)

pMRF was traditionally widely used for semi-supervised node classification in various applications such as auction fraud detection (Pandit et al. 2007), fake accounts and reviews detection (Wang et al. 2011; Cao et al. 2012; Akoglu, Chandy, and Faloutsos 2013; Rayana and Akoglu 2015; Wang, Zhang, and Gong 2017; Wang, Gong, and Fu 2017; Chandy, and Faloutsos 2013; Rayana and Akoglu 2015; Wang, Gong, and Fu 2017; Wang, Jia, and Gong 2019), malware detection (Tamersoy, Roundy, and Chau 2014), and user’s private attribute inference in social networks (Gong et al. 2014; Jia et al. 2017; Gong and Liu 2018; Wang, Jia, and Gong 2019). Specifically, we associate a random variable $r_v$ with each node $v$ to model its label, where $r_v = i$ means that $v$’s label is $i$.

pMRF essentially models the joint probability distribution as follows:

$$Pr(r_1, r_2, \ldots, r_n) \propto \prod_{v \in V} \phi_v(r_v) \prod_{(u,v) \in E} \psi_{uv}(r_u, r_v),$$

where $\phi_v(r_v)$ is called node potential and $\psi_{uv}(r_u, r_v)$ is called edge potential. For convenience, we represent the node potential $\phi_v(r_v)$ as a probability distribution $q_v$ over the $C$ labels $\{1, 2, \ldots, C\}$. Specifically, $q_v$ is a row vector and $q_v(i) = \phi_v(r_v = i)$ for $i \in \{1, 2, \ldots, C\}$. If nodes’ feature vectors are available, $q_v$ can be learnt.

In particular, given the labeled nodes $L$ and nodes’ feature vectors, we learn a multi-class logistic regression classifier and use the classifier to predict the probability distribution $q_v$ for each node. If nodes’ feature vectors are unavailable, we can assign $q_v$ based on the labeled nodes $L$. Specifically, if node $v$ is a labeled node, $q_v$ is set as the node $v$’s one-hot label vector; otherwise, all entries in $q_v$ have the equal value $1/C$.

Moreover, we represent the edge potential $\psi_{uv}(r_u, r_v)$ as a symmetric matrix $H_{uv} \in \mathbb{R}^{C \times C}$, where $H_{uv}(i, j) = \psi_{uv}(r_u = i, r_v = j)$ for $i, j \in \{1, 2, \ldots, C\}$. $H_{uv}$ models the coupling strength between the labels of nodes $u$ and $v$, and thus $H_{uv}$ is also called coupling matrix. In particular, a larger $H_{uv}(i, j)$ means that $u$ and $v$ are more likely to have labels $i$ and $j$, respectively.

Inference via Linearized Belief Propagation (LinBP): Given the joint probability distribution in Equation 1, the probability distribution $q_v$ for each node, and the coupling matrix $H_{uv}$ for each edge, we infer the marginal probability distribution $\tilde{p}_v$ (a row vector) for each node $v$ and use it to predict $v$’s label. Specifically, Belief Propagation (BP) (Pearl 1988) is a standard method to infer the marginal probability distributions. However, BP is not guaranteed to converge on loopy graphs and is not scalable due to message maintenance on each edge. Recent work (Gatterbauer et al. 2015; Gatterbauer 2017) proposed to linearize BP (LinBP) to address these limitations. Given LinBP, the approximate marginal probability distributions $p_v$ are solutions to the following linear system (Gatterbauer 2017):

$$p_v = q_v + \sum_{u \in \Gamma_v} p_u H_{uv}, \forall v \in V,$$

where $\Gamma_v$ is the set of neighbors of $v$, $q_v = \tilde{q}_v - \frac{1}{C}$, and $H_{uv} = \tilde{H}_{uv} - \frac{1}{C}$. Roughly speaking, $q_v$ and $H_{uv}$ are centered versions of $\tilde{q}_v$ and $H_{uv}$, respectively. Note that $p_v$ solved by LinBP from Equation 2 is not necessarily a probability distribution anymore, e.g., it may not sum to 1. The label of $v$ is predicted as the one that has the largest value in $p_v$.

Limitation: The key limitation of existing pMRF-based semi-supervised node classification methods is that they assume a heuristics-based constant coupling matrix for all edges. For example, $H_{uv}(i, i) = 0.9$ and $H_{uv}(i, j) = \frac{0.1}{C-1}$, where $i, j \in \{1, 2, \ldots, C\}$ and $i \neq j$, which means that two linked nodes are more likely to have the same label and does not distinguish between labels and between edges. In this work, we address this limitation via proposing a method to learn the coupling matrix. Moreover, we empirically show that, after learning the coupling matrix, pMRF outperforms state-of-the-art graph neural networks on multiple benchmark datasets.

Learning the Coupling Matrix (LCM)

Formulating an Optimization Problem

Constraining the coupling matrices: In pMRF, each edge has a coupling matrix $H_{uv}$. Therefore, it is challenging to learn a different coupling matrix for each edge as the number of parameters is much larger than the training dataset size. To address the challenge, we constrain the coupling matrices of different edges as different scalings of the same coupling matrix. In particular, we assume $H_{uv} = W_{uv} H$ for each edge $(u, v) \in E$. We model the scaling parameter $W_{uv}$ as the weight of the edge $(u, v)$ and denote all the scaling parameters as a symmetric weight matrix $W$ of the graph. Given such assumption on the coupling matrices, we can transform Equation 2 as follows:

$$P = Q + WPH,$$

where the matrix $P = [p_1; p_2; \ldots; p_n]$ and the matrix $Q = [q_1; q_2; \ldots; q_n]$. According to (Gatterbauer et al. 2015), Equation 3 has a solution when $I - H \otimes W$ is invertible, where $I$ is an identity matrix and $\otimes$ is the Kronecker product. LinBP uses the power iteration method to solve Equation 3 and it can find the solution when $\rho(H) < \frac{1}{\mu(W)}$,

\footnote{The spectral radius of a square matrix is the maximum of the absolute values of its eigenvalues.}
Formulating an optimization problem: Intuitively, we aim to achieve two goals when learning the coupling matrix and weight matrix. The first goal is that the labels predicted for the training nodes based on the learnt coupling matrix and weight matrix should match the ground truth labels of the training nodes. The second goal is that the predicted label distributions $\sigma(p_u)$ and $\sigma(p_v)$ of nodes $u$ and $v$ should be consistent with the coupling matrix $W_{uv}$ for each edge $(u, v)$. Specifically, if $u$ and $v$ are more likely to be predicted to have labels $i$ and $j$ (i.e., the probabilities $\sigma(p_u)_i$ and $\sigma(p_v)_j$ are larger), respectively, then the entry $W_{uv}H(i, j)$ should be larger. This is because $W_{uv}H(i, j)$ encodes the likelihood that $u$ and $v$ have labels $i$ and $j$, respectively.

To quantify the first goal, we adopt a loss function over the training dataset. Specifically, we consider a standard cross-entropy loss function $L(W, H)$ as follows:

$$L(W, H) = - \sum_{l \in L} y_l^T \log \sigma(p_l),$$

where $\sigma$ is the softmax function, which is defined as $\sigma(p)_l = \frac{\exp(p_l)}{\sum_{j} \exp(p_j)}$ and converts the vector $p_l$ into a probability distribution; the vector $y_l$ is the one-hot encoding of the node $l$’s label, i.e., if node $l$ belongs to the $l$th class, then $y_l(i) = 1$ and $y_l(j) = 0$, for all $j \neq i$. To quantify the second goal, we define the following term $R(W, H)$:

$$R(W, H) = - \sum_{(u,v) \in E} \sigma(p_u) \cdot W_{uv}H \cdot \sigma(p_v)^T.$$  

$R(W, H)$ is smaller if the second goal is better satisfied, i.e., the predicted label distributions of two nodes are more consistent with the coupling matrix of the corresponding edge. Combining the two goals, we propose to learn the coupling matrix and weight matrix via solving the following optimization problem:

$$\min_{W, H} L(W, H) = L(W, H) + \lambda R(W, H),$$

where $\lambda > 0$ is a hyperparameter that balances between the two goals. Our term $R(W, H)$ can also be viewed as a regularization term, and we call it consistency regularization. We note that a similar consistency regularization term was also proposed by (Wang, Jia, and Gong 2019), but only for binary classification.

Learning $W(t)$ and $H(t)$: Given the weight matrix $W(t-1)$ and the coupling matrix $H(t-1)$, and the matrix $P(t-1)$ in the $(t-1)$th iteration, we learn the weight matrix $W(t)$ and the coupling matrix $H(t)$ in the $t$th iteration as a solution to the following optimization problem:

$$\min_{W(t), H(t)} L(W(t), H(t)) = - \sum_{l \in L} y_l^T \log \sigma(p_l^{(t+1)}) - \lambda \sum_{(u,v) \in E} \sigma(p_u^{(t)}) \cdot W_{uv}^{(t)} H^{(t)} \cdot (\sigma(p_v^{(t)}))^T,$$

where $P^{(t+1)} = Q + W^{(t-1)}P^{(t-1)}H^{(t)}$. We leverage gradient descent to learn $W(t)$ and $H(t)$. Specifically, we have the following gradients:

$$\frac{\partial L(W(t), H(t))}{\partial W_{uv}^{(t)}} = - \sum_{l \in L} \sum_{j = 1}^{C} (y_l(j) - \sigma(p_l^{(t+1)})) \frac{\partial p_l^{(t+1)}(j)}{\partial W_{uv}^{(t)}} - \lambda \sigma(p_u^{(t)}) \cdot H^{(t)} \cdot (\sigma(p_v^{(t)}))^T,$$

$$\frac{\partial L(W(t), H(t))}{\partial H_{ij}^{(t)}} = - \sum_{l \in L} \left( (y_l(j) - \sigma(p_l^{(t+1)})) \frac{\partial p_l^{(t+1)}(j)}{\partial H_{ij}^{(t)}} + (y_l(i) - \sigma(p_l^{(t+1)})) \frac{\partial p_l^{(t+1)}(i)}{\partial H_{ij}^{(t)}} \right) - \lambda \sum_{(u,v) \in E} W_{uv}^{(t)} \sigma(p_u^{(t)}) \cdot \sigma(p_v^{(t)})^T,$$

where we have:

$$\frac{\partial p_l^{(t+1)}(j)}{\partial W_{uv}^{(t)}} = \begin{cases} (P^{(t)}H^{(t)})_{uv}, & \text{if } u = l \\ 0, & \text{otherwise} \end{cases},$$

$$\frac{\partial p_l^{(t+1)}(i)}{\partial H_{ij}^{(t)}} = \begin{cases} (W^{(t-1)}P^{(t-1)})_{li}, & \text{if } v = l \\ 0, & \text{otherwise} \end{cases}.$$

Solving the Optimization Problem

We propose an approximate algorithm to efficiently solve the optimization problem. Our key observation is that $P$ is often iteratively solved from Equation 3 as follows:

$$P^{(t)} = Q + W^{(t-1)}P^{(t-1)}H^{(t)}.$$
We have both \( \frac{\partial p_t^{(t+1)}(j)}{\partial H_{ij}^{(t)}} \) and \( \frac{\partial p_t^{(t+1)}(i)}{\partial H_{ij}^{(t)}} \) when computing \( \partial E(W^{(t)}, H^{(t)})/\partial H_{ij}^{(t)} \) because \( H \) is symmetric. We initialize \( W^{(t)} \) and \( H^{(t)} \) as \( W^{(t-1)} \) and \( H^{(t-1)} \), respectively. Then, we use gradient descent to update \( W^{(t)} \) and \( H^{(t)} \) with learning rates \( \gamma_1 \) and \( \gamma_2 \), respectively.

We note that the learnt weight matrix \( W \) and coupling matrix \( H \) are bounded during optimization. This is because the edge potential matrix \( H_{uv} \) is non-negative and each row sums to 1. Moreover, \( W_{uv} H = H_{uv} = H_{uv} - 1/C \). Therefore, \( W_{uv} H \) is bounded. During optimization, we can clip \( W_{uv} H \) to satisfy such constraints if they are too large. However, we found that \( W_{uv} H \) is small even if we do not clip in our experiments. This is because of our initializations of \( W_{uv} \) and \( H_{uv} \) and that our LCM method requires a small number of iterations.

### Evaluation

#### Benchmark Datasets

Following previous work (Yang, Cohen, and Salakhutdinov 2016; Kipf and Welling 2017), we use three benchmark citation graphs (i.e., Cora, Citeseer, and Pubmed) (Sen et al. 2008) and one benchmark knowledge graph NELL (Carlson et al. 2010; Yang, Cohen, and Salakhutdinov 2016). We also adopt two real-world large-scale social graphs (i.e., Google+ (Gong et al. 2012; Jia et al. 2017) and Twitter (Wang, Zhang, and Gong 2017)) to evaluate our method. Descriptions of these datasets and their basic statistics are shown in (Wang, Jia, and Gong 2020).

#### Training, Validation, and Testing

We split each dataset into a training dataset, a validation set, and a testing dataset. We use the validation dataset to tune hyperparameters and use the testing dataset for evaluation.

For the citation graphs, previous work (Kipf and Welling 2017; Velickovic et al. 2018) has set 1,000 nodes in each graph as the testing dataset. For consistent comparisons, we adopt these testing datasets and fix them. Moreover, we sample 20 nodes from each class uniformly at random as the validation dataset. For the NELL graph, previous work (Yang, Cohen, and Salakhutdinov 2016) has set 969 nodes as the testing dataset, which we adopt. Moreover, like previous work (Kipf and Welling 2017), we consider the extreme case where just one node from each class is sampled for training. We also randomly sample 500 nodes in total as the validation dataset. We repeat sampling the training and validation datasets 5 trials and report the average accuracies and standard deviations on the testing dataset.

For the Google+ dataset, for each city, we randomly select 1,000 positive users, i.e., who live/lived in the city, and 1,000 negative users, i.e., who do/did not live in the city, as the training dataset; we randomly select 1,000 positive users and 1,000 negative users as the validation dataset; and we treat the remaining users who disclosed at least one city as the testing dataset. Note that we have 50 cities in the Google+ dataset, where each city is treated as a binary classification problem. The accuracy on the Google+ dataset is averaged over the 50 cities. For the Twitter dataset, we randomly sample 1,000 genuine users and 1,000 fake users as the training dataset; we randomly sample 1,000 genuine users and 1,000 fake users as the validation dataset; and we randomly select another 5,000 genuine users and 5,000 fake users as the testing dataset. We sample a relatively larger number of training users in Google+ and Twitter, due to their large graph sizes.

### Compared Methods

We compare our method with graph embedding methods, graph neural networks, and belief propagation methods.

- **Graph embedding methods**: These methods first learn an embedding vector for each node; then they train a standard classifier (we consider multi-class logistic regression) using the training dataset and the embedding vectors; finally, they use the classifier to predict labels for the testing nodes. We consider two representative graph embedding methods, i.e., DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) and Planetoid (Yang, Cohen, and Salakhutdinov 2016). We use 128-dimension embedding vectors.

- **Graph neural network methods**: These methods learn the node embeddings and classify nodes simultaneously. In particular, we compare with Chebyshev (Kipf and Welling 2017), graph convolutional network (GCN) (Kipf and Welling 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), and graph attention network (GAT) (Velicković et al. 2018).

- **Belief propagation methods**: We use the standard BP (Pearl 1988) or the linearized BP (LinBP) (Gatterbauer et al. 2015) to perform inference in the pMRF. In particular, BP and LinBP infer the approximate marginal probability distributions \( P \) and use them to predict labels of testing nodes. We obtained the implementation of BP and LinBP from (Gatterbauer et al. 2015).

- **Our method LCM and its variants**: In order to show the effectiveness of our consistency regularization term, we consider several variants of LCM. Specifically, LCM-wo is the variant of our method that does not use the consistency regularization term. Moreover, we compare our consistency regularization with the standard \( L_1 \) and \( L_2 \) regularizations, which are denoted as \( R(W, H) = \sum_{(u,v) \in E} |W_{uv}|_1 + \sum_{i,j} |H_{ij}|_1 \) and \( R(W, H) = \sum_{(u,v) \in E} W_{uv}^2 + \sum_{i,j} H_{ij}^2 \), respectively. Accordingly, we denote the two variants as LCM-L1 and LCM-L2, respectively.

We observe that BP and LinBP outperform other traditional methods such as label propagation (LP) (Zhu, Ghahramani, and Lafferty 2003), manifold regularization (ManiReg) (Belkin, Niyogi, and Sindhwani 2006), deep semi-supervised embedding (SemiEmb) (Weston et al. 2012), and iterative classification algorithm (ICA) (Lu and Getoor 2003). For instance, using the same experimental setting as Velickovic et al. (Velickovic et al. 2018),
LinBP achieves classification accuracies 0.785, 0.709, and 0.787 on the three citation graphs, respectively. According to (Veličković et al. 2018), ICA consistently outperforms LP, ManiReg, and SemiEmb on the three citation graphs. However, ICA only achieves accuracies 0.751, 0.691, and 0.739 on the three citation graphs, respectively. Thus, we omit the results of these traditional methods for simplicity. The parameter settings of these methods are detailed in (Wang, Jia, and Gong 2020).

### Results

Table 1 shows the classification accuracies averaged over 5 trials and the standard deviations of compared methods on the six graphs. "-" means the methods cannot be executed on our machine as they require more memory than we have. We note that all the compared graph embedding methods and graph neural networks cannot be executed on our machine for the two large-scale social graphs.

<table>
<thead>
<tr>
<th>Methods</th>
<th>DeepWalk</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>NELL</th>
<th>Google+</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ave. initialized weights</td>
<td>0.669±0.017</td>
<td>0.428±0.019</td>
<td>0.651±0.015</td>
<td>0.352±0.013</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ave. learnt weights</td>
<td>0.709</td>
<td>0.655±0.029</td>
<td>0.699±0.035</td>
<td>0.435±0.013</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Average initialized weights and weights learnt by our method for homogeneous edges vs. heterogeneous edges on the six graphs. “homo” and “heter” mean homogeneous edges and heterogeneous edges, respectively. Our method learns larger average weights for homogeneous edges than heterogeneous edges.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>NELL</th>
<th>Google+</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ave. initialized weights</td>
<td>homo</td>
<td>heter</td>
<td>homo</td>
<td>heter</td>
<td>homo</td>
<td>heter</td>
</tr>
<tr>
<td></td>
<td>0.225</td>
<td>0.197</td>
<td>0.315</td>
<td>0.347</td>
<td>0.138</td>
<td>0.147</td>
</tr>
<tr>
<td>Ave. learnt weights</td>
<td>0.354</td>
<td>0.261</td>
<td>0.493</td>
<td>0.448</td>
<td>0.652</td>
<td>0.548</td>
</tr>
</tbody>
</table>

LinBP achieves classification accuracies 0.785, 0.709, and 0.787 on the three citation graphs, respectively. According to (Veličković et al. 2018), ICA consistently outperforms LP, ManiReg, and SemiEmb on the three citation graphs. However, ICA only achieves accuracies 0.751, 0.691, and 0.739 on the three citation graphs, respectively. Thus, we omit the results of these traditional methods for simplicity. The parameter settings of these methods are detailed in (Wang, Jia, and Gong 2020).

#### LCM outperforms the compared methods:

First, we observe that graph neural networks outperform standard BP and LinBP methods. For example, GCN consistently achieves higher average accuracies than BP and LinBP on the three citation graphs and the NELL knowledge graph.

Second, LCM is consistently more accurate than BP and LinBP. For example, LCM achieves 0.018 to 0.060 higher accuracies than BP on the three citation graphs and the NELL knowledge graph; and achieves 0.015 to 0.059 higher accuracies than LinBP on the six graphs. The reason is that BP and LinBP do not learn the edge weights and coupling matrix (i.e., they use the initialized edge weights and coupling matrix), while LCM does. To further illustrate the difference between LCM and BP/LinBP, Table 2 shows the average initialized weights and average weights learnt by LCM for homogeneous edges and heterogeneous edges on the six graphs. An edge is homogeneous if the two corresponding nodes have the same label, otherwise the edge is heterogeneous. Moreover, Figure 1 shows the learnt coupling matrices for the six graphs. Our learnt coupling matrices are diagonal dominant, which means that two linked nodes are more likely to have the same label. Moreover, homogeneous edges have larger weights than heterogeneous edges on average after learning the edge weights. In particular, LCM increases edge weights for both homogeneous edges and heterogeneous edges, but LCM increases the weights of homogeneous edges more substantially.

Third, after learning the weight matrix and the coupling matrix, our method outperforms the graph neural networks and graph embedding methods. Specifically, LCM achieves 0.009 to 0.051 higher average accuracies than the graph neural networks, and 0.071 to 0.212 higher average accuracies than the graph embedding methods on the three citation graphs and the NELL knowledge graph.

#### Our consistency regularization is effective:

Our results also show that our consistency regularization term is effective. In particular, LCM outperforms LCM-wo, LCM-L1, and LCM-L2. Our method achieves 0.006 to 0.036 higher accuracies when using the consistency regularization term (i.e., LCM vs. LCM-wo). One possible reason is that L1/L2 regularization aims to prevent overfitting of complex models. However, pMRF is a simple model, which may not have overfitting on the datasets. Therefore, L1/L2 regularization could make LCM worse. Note that our consistency regularization does not aim to prevent traditional overfitting. Instead, it aims to improve the pMRF model itself and thus can outperform LCM-wo.
Figure 1: Learned coupling matrices $H$ on the six graphs. The coupling matrices are diagonal dominant. For the NELL graph, it is better to zoom in the subfigure to observe the diagonal dominant property of the learned coupling matrix.

Figure 2: Impact of the number of training nodes per class.

**Impact of the number of training nodes per class:** Figure 2 further shows the average accuracies of different methods on the six graph datasets when we sample a particular number of training nodes from each class. Our method consistently outperforms the compared methods when the training dataset size is small. For instance, when each class has
**Related Work**

**Conventional methods:** Semi-supervised node classification in graph-structured data has been studied extensively and many methods have been proposed. For instance, conventional methods include label propagation (LP) (Zhu, Ghahramani, and Lafferty 2003), manifold regularization (ManiReg) (Belkin, Niyogi, and Sindhwani 2006), deep semi-supervised embedding (SemiEmbed) (Weston et al. 2012), iterative classification algorithm (ICA) (Lu and Getoor 2003), and pairwise Markov Random Fields (pMRF) (Gatterbauer 2017). In particular, pMRF associates a discrete random variable with each node to model its label and defines a joint probability distribution for the random variables associated with all nodes, where the statistical correlations/independence between the random variables are captured by the graph structure. Then, either the standard Belief Propagation (BP) (Pearl 1988) or linearized BP (LinBP) (Gatterbauer et al. 2015; Gatterbauer 2017) is used for inference. Existing studies on pMRF-based semi-supervised node classification share a common limitation: they set a constant edge potential for all edges in the graph.

**Graph embedding methods and graph neural networks:** A recent trend is to extend neural networks to graph-structured data for semi-supervised node classification. Specifically, deep semi-supervised embedding methods such as LP and LinBP train node embeddings and then learn a standard classifier (e.g., logistic regression) using the embeddings to classify nodes, where learning the embeddings and learning the classifier are performed separately. Different graph embedding methods leverage different techniques to learn node embeddings. For instance, DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) learns nodes’ embeddings via generalizing the word to vector technique (Mikolov et al. 2013) developed for natural language processing to graph data. Specifically, DeepWalk treats a node as a word in natural language, generates node sequences using truncated random walks on a graph, and leverages the skip-gram model (Mikolov et al. 2013) to learn an embedding vector for each node.

Graph neural networks learn the node embeddings and the classifier to classify nodes simultaneously. In particular, the hidden layers represent node embeddings and the last layer models a classifier to classify nodes. Different graph neural networks use different neural network architectures. For instance, Graph Convolutional Network (GCN) (Kipf and Welling 2017) uses an architecture that is motivated by spectral graph convolutions (Duvenaud et al. 2015). Specifically, the input layer is the nodes’ features. A hidden layer models nodes’ embeddings. In particular, the neural network iteratively computes a node’s embedding vector in a hidden layer via aggregating the embedding vectors of the node’s neighbors in the previous layer. State-of-the-art graph neural networks are more accurate than conventional methods such as LP and pMRF on multiple benchmark datasets. However, graph neural networks are less efficient than conventional methods such as LP and LinBP. Our method addresses the limitation of pMRF-based methods and is more accurate and efficient than graph neural networks.

**Conclusion and Future Work**

We propose a novel method to learn the coupling matrix in pairwise Markov Random Fields for semi-supervised node classification in graph-structured data. We formulate learning coupling matrix as an optimization problem, whose objective function is the sum of the training loss and a consistency regularization term that we propose. Moreover, we propose an iterative algorithm to solve the optimization problem. Our evaluation results on six benchmark datasets show that our optimized pMRF-based method is more accurate and efficient than state-of-the-art graph neural networks. An interesting future work is to explore the connections between our optimized pMRF-based method and graph neural networks as well as unify them in a general framework.
Acknowledgments

We would like to thank the anonymous reviewers for their insightful reviews. This work was supported by the National Science Foundation under grants No. 1937787 and 1937786. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References


Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In ICLR.


