Ph.D. Dissertation

Probabilistic Approaches for Node and Graph Classification

정점 및 그래프 분류를 위한 확률 기반 접근법

February 2022

Department of Computer Science and Engineering
College of Engineering
Seoul National University

Jaemin Yoo

Probabilistic Approaches for Node and Graph Classification

U Kang

Submitting a Ph.D. Dissertation

January 2022

Department of Computer Science and Engineering
College of Engineering
Seoul National University

Jaemin Yoo

Confirming the Ph.D. Dissertation Written by
Jaemin Yoo
December 2021

Chair	Hyoung-Joo Kim	(Seal)
Vice Chair	U Kang	(Seal)
Examiner	Sun Kim	(Seal)
Examiner	Kunsoo Park	(Seal)
Examiner	Hwanjo Yu	(Seal)

Abstract

Graphs appear as various forms in real-world applications: social networks, web graphs, recommender systems, or chemical compounds. Such real-world graphs include various attributes, such as node features or labels, which determine the properties of relationships between connected entities. The naive assumption of homophily does not apply to such attributed graphs, where adjacent nodes can have different or even opposite characters based on attribute values. Thus, it is a challenging problem to devise effective algorithms for real-world graphs due to their dynamic and complex nature embedded in attribute information.

Our goal in this thesis is to understand the dynamic relationships between features, labels, and the structure of a graph through probabilistic approaches. Particularly, we solve various graph-related problems that require bidirectional mappings between different types of graph attributes: a) node classification, b) cold-start inductive learning, c) node feature estimation, d) subgraph sampling, and e) graph augmentation. Our proposed approaches improve previous works in real-world graphs especially when the number of observations is insufficient to train complex models such as graph neural networks, thanks to the robust and scalable nature of probabilistic modeling which makes a uniform consistent assumption on the entire graph with only a few parameters.

Our proposed approaches are categorized into structural exploitation and structural modification. The goal of structural exploitation is to fully exploit a graph structure, without changing it, to understand the unknown properties of nodes and their complex relationships. The goal of structural modification is

to change a graph structure to make it better represent target properties or to give richer evidence to the training of estimators. Structural modification is a powerful way to improve the performance of estimators, especially in noisy and unreliable datasets that provide incomplete information for the training.

Extensive experiments on many real-world datasets demonstrate that our approaches make a superior performance compared with existing probabilistic and neural network-based approaches in two representative problems in graphs: node and graph classification. In node classification, we make up to 15.6% higher accuracy in transductive learning, 5.2% higher accuracy in inductive learning, 13.7% higher F1 score through subgraph sampling, and 7.0% higher accuracy through missing feature estimation, compared to the best competitors. In graph classification, we improve the accuracy of a base classifier up to 2.1× compared to the best algorithms for model-agnostic graph augmentation.

Keywords : Graphical inference, graph neural networks, loopy belief propagation, Markov random fields, inductive learning, node classification, feature estimation, subgraph sampling, graph classification, graph augmentation

Student Number : 2016-21218

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are widely used to represent real-world applications such as social networks, e-commerce, streaming services, or the world wide web. Understanding graph-structured data is equivalent to understanding the relationships between target entities of interest, and it allows us to devise effective solutions to real-world problems by combining partial evidence drawn from individual examples. Thus, most recommender systems or social analysis tools exploit the graphical structure between entities to achieve high accuracy.

Many real-world graphs have attribute information such as node features, labels, or edge features that determine the specific properties of relationships. Such attributes play an essential role in understanding the true properties of a graph. However, the true meanings of attributes are determined dynamically depending on circumstances. For example, in a streaming service, a review from a user to a movie creates an attributed edge with a rating between 1.0 and 5.0. The meanings of scores are determined by various factors, such as the user who made the review or the time when it was made. Some users use the score of 3.0 as a negative opinion, while some use it to express good impression.

This thesis studies the dynamic relationships between features, labels, and the structure of a graph. These three types of attributes make interactions in various forms, and it is challenging to understand their complex relationship by a single unidirectional problem. Thus, we focus on various graph-related tasks

	Structural Exploitation		Structural Modification
	$P(Y\|X,G)$	$P(X\|G)$	$P(G'\|G)$
Purely Probabilistic	Transductive Learning **[ICDM'17]**	-	Subgraph Sampling **[WSDM'20]**
Fusion with Deep Learning	Inductive Learning **[IJCAI'19]**	Feature Estimation **[submitted]**	Graph Augmentation **[WWW'22]**

Figure 1.1: An overview of works included in this thesis. They are categorized into a) structural exploitation and b) structural modification based on whether the graph structure is preserved or not. The variables X, Y, and G represent a node feature, a node label, and a graph structure, respectively.

that have different objectives and assumptions: transductive node classification, hard inductive node classification, node feature estimation, subgraph sampling, and graph augmentation. The solutions to such tasks allow us to learn complex relationships between the target properties, which result in the improvement of accuracy in representative problems of node and graph classification.

Figure 1.1 shows an overview of our works included in this thesis. The five works are mainly categorized by whether the fixed graph structure is exploited or a new structure is learned from the given graph. The structural exploitation is a basic approach to understand and utilize given graphs, while the structural modification enhances the effectiveness of traditional algorithms. The proposed methods are also divided into purely probabilistic ones and deep learning-based ones. Our research started with purely probabilistic approaches and then deep learning techniques were adopted to enhance the limited representation power of probabilistic approaches and to embrace abundant training data.

Part I (Chapter 3 to 5) focuses on estimating node features or predicting node labels with limited observations, without changing the structure of a given graph. We aim to answer the following research questions:

- **Chapter 3:** How can we accurately classify nodes in a graph having edge attributes? How can we improve loopy belief propagation for that?
- **Chapter 4:** How can we accurately classify unseen test nodes having no neighborhood information in hard inductive scenarios?
- **Chapter 5:** How can we accurately estimate missing node features with no information given for describing the target nodes?

Part II (Chapter 6 and 7) focuses on modifying the structure of a graph to effectively solve essential graph-based tasks. The solutions proposed in this part can be utilized to improve the algorithms in Part I. The research questions that we aim to answer in this part are summarized as follows:

- **Chapter 6:** How can we improve the accuracy and speed of inference for node classification through sampling efficient subgraphs? Can we achieve higher classification accuracy in subgraphs with partial edges?
- **Chapter 7:** How can we augment the structure of a graph in an effective and balanced way for accurate graph classification? How can we minimize the risk of changing semantic information of graphs?

1.1 Contributions

We provide a summary of our contributions in this thesis:

- Node classification in an edge-attributed graph (Chapter 3): We propose SBP, an approach to accurately classify nodes in edge-attributed graphs. SBP models the propagation strength as a function of edge attributes to

control the degree of propagation via supervised learning.

- Hard inductive node classification (Chapter 4): We propose BPN, a deep learning-based approach to accurately classify unseen isolated nodes at the test time. BPN shows a superior performance on hard inductive learning based on the separable structure of classification and diffusion.
- Node feature estimation (Chapter 5): We propose MGA, an autoencoder-like model that generates missing features from only partial observations existing in a graph. MGA presents a strong structural regularizer to avoid overfitting while estimating high-dimensional numerical features.
- Subgraph sampling (Chapter 6): We propose BTW, an algorithm to sample subgraphs with bounded treewidth. The generated subgraphs can be used to a) speed up approximate inference or b) support tractable exact inference for higher accuracy of node classification.
- Graph augmentation (Chapter 7): We propose NodeSam and SubMix to augment graph structures for accurate graph classification. They satisfy five desired properties for effective augmentation by making balanced and sufficient changes of the graph structure in a model-agnostic way.

1.2 Overall Impact

In addition to the contributions above, our works have a broad impact on the field of graph mining. We highlight the impact of our work as follows:

- Our work on transductive node classification [1] was awarded the Google Conference Scholarship for attending ICDM 2017.
- Our work on subgraph sampling [2] was awarded the Qualcomm Innova-

tion Fellowship Korea and the Samsung HumanTech Paper Award.

- Our two works on node classification [1, 3] were presented as invited talks at Korea Software Congress 2017 and 2019, respectively.
- The works included in this thesis [1, 2, 3, 4] were supported by the Google PhD Fellowship and the Yulchon AI Star Award.
- We have a registered patent on node classification [1] and two filed patents on node feature estimation and graph augmentation [4], respectively.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we introduce basic concepts and notations used throughout this thesis. In Part I (Chapter 3 to 5), we introduce our approaches to estimate node features or labels exploiting the structure of graphs. In Chapter 3, we present SBP for classifying nodes in edge-attributed graphs. In Chapter 4, we propose BPN for classifying unseen isolated nodes in inductive learning scenarios. In Chapter 5, we present MGA for estimating missing features. In Part II (Chapter 6 to 7), we introduce our approaches that modify graph structures to improve the efficiency of node and graph classification. In Chapter 6, we introduce BTW for sampling subgraphs to support accurate and efficient graphical inference. In Chapter 7, we propose NodeSam and SubMix for balanced augmentation of graphs to accurate graph classification. In Part III (Chapter 8 to 9), we make a conclusion and present future research directions in Chapter 8 and 9, respectively.

Chapter 2

Background

In this chapter, we introduce concepts and notations that are used throughout this thesis. Frequently-used symbols are summarized in Table 2.1.

2.1 Basic Concepts

Graphs A *graph* $G = (\mathcal{V}, \mathcal{E})$ is a data structure that represents the relationships between *nodes* in a set $\mathcal{V}$ based on a set $\mathcal{E}$ of *edges*. Each edge $(i, j) \in \mathcal{E}$ represents that nodes i and j are connected. The set $\mathcal{E}$ of edges is often represented with an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ whose elements are either zero or one. $A_{ij} = 1$ if $(i, j) \in \mathcal{E}$, and $A_{ij} = 0$ if $(i, j) \notin \mathcal{E}$. We do not consider the directions of edges in this thesis, i.e., $A_{ij} = 1$ if and only if $A_{ji} = 1$.

Features A graph is often given with a feature matrix $\mathbf{X} \in \mathbb{R}^{l \times d}$, which can contain either node features or edge features. $l = |\mathcal{V}|$ if it is for node features, and $l = |\mathcal{E}|$ if it is for edge features. d represents the number of features.

Labels We study two kinds of classification problems in this thesis: node-level and graph-level classification. In node classification, we assume that each node i has a discrete label y_i that represents its property. In this case, if each node has also a feature vector, the problem can be considered as typical classification where the relationships between examples are given as auxiliary information.

Table 2.1: Table of symbols.

Symbol	Definition
G	Graph consisting of nodes $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{V}$	Set of nodes
$\mathcal{E}$	Set of undirected edges
$\mathcal{N}(i)$	Set of neighbors of node i
ϵ	Propagation strength
ϕ	Node potential function
ψ	Edge potential function

In graph classification, we assume that a set of graphs is given, each of which has a discrete label. In this case, each graph works as a target example to be classified, and the extraction of structural properties is more important.

2.2 Probabilistic Modeling

Pairwise Markov random field A *pairwise Markov random field* (MRF) is a set of discrete random variables whose joint relationships are modeled as an undirected graph. Given variables $X = (X_i)_{i \in \mathcal{V}}$ which are modeled as a graph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of nodes and edges, respectively, the joint probability $p(X = x)$ is computed by multiplying all the *potentials* ϕ and ψ as

$$p(X = x) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \phi_i(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j),$$

where Z is a normalization constant. A node potential $\phi_i(x_i)$ represents an unnormalized probability of node i being in state x_i without considering the influences from other nodes. An *edge potential* $\psi_{ij}(x_i, x_j)$ represents a joint unnormalized probability of nodes i and j being in states x_i and x_j.

Potential functions The potential functions ϕ and ψ determine the probabilistic properties of a graph in the Markov modeling. The node potential ϕ_i for each node i can be a constant following the observed state of node i (in Chapter 3 and 6) or the output of a classifier function (in Chapter 4). The edge potential ψ_{ij} for each edge (i, j) can also be the output of a function that takes an edge attribute as an input (in Chapter 3) or a constant that assumes the homophily property (in Chapter 4 and 6). It is also possible to generalize the target variables from discrete ones to continuous ones that follow a Multivariate Gaussian distribution, making Gaussian Markov random field (in Chapter 5). Still, we assume target nodes with discrete states in most of this thesis.

The edge potential function is mostly modeled with a single hyperparameter, ϵ, which we call the *propagation strength*. We use ϵ to control how much adjacent nodes are correlated with each other. If ϵ is large, adjacent nodes are assumed to have a high probability for having the same state, allowing us to propagate more information through the graph structure by inference.

Loopy belief propagation *Loopy belief propagation* (LBP) is an approximate algorithm to compute the marginal distribution of a pairwise MRF by passing *messages* between the variables [5, 6]. A message $m^*_{ij}(x_j)$ can be considered as an opinion of node i about the probability of node j being in state x_j. LBP uniformly initializes all messages and updates them through iterations until convergence. Equation (3.1) shows how to update $m^*_{ij}(x_j)$ at each iteration where $\mathcal{N}(i)$ denotes the set of neighbors of node i. All the incoming messages

from $\mathcal{N}(i)$ except node j are multiplied to compute $m^*_{ij}(x_j)$.

$$m^*_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j)\frac{\prod_{k\in\mathcal{N}(i)} m^*_{ki}(x_i)}{m^*_{ji}(x_i)} \tag{2.1}$$

LBP returns *beliefs* as a result. A belief $b_j(x_j)$ is an approximate marginal probability of node j being in state x_j, which is computed from the converged messages and then normalized. Equation (3.2) shows how to compute the unnormalized beliefs and then how to normalize them accordingly.

$$\begin{aligned} b_j(x_j) &= \frac{b^*_j(x_j)}{\sum_{x'_j} b^*_j(x'_j)} \text{ where} \\ b^*_j(x_j) &= \phi_j(x_j) \prod_{i\in\mathcal{N}(j)} m^*_{ij}(x_j) \end{aligned} \tag{2.2}$$

A popular application of LBP is node classification whose goal is to infer the states of unobserved nodes based on the known states of observed nodes. LBP solves this by assigning different node potentials to the nodes and then propagating the messages. For instance, assuming random variables having two states, LBP gives a node potential $(0.9, 0.1)$ or $(0.1, 0.9)$ to a observed node based on its observed state. On the other hand, LBP gives a node potential $(0.5, 0.5)$ to the unobserved nodes. LBP propagates the messages from the biased potentials of the observed nodes and classifies the unobserved nodes. Although it is not guaranteed, the messages often converge in a small number of iterations for most real-world networks.

Part I

Structural Exploitation

Chapter 3

Node Classification with Edge Attributes

Given an undirected network where some of the nodes are labeled, how can we classify the unlabeled nodes with high accuracy? Loopy belief propagation (LBP) is an inference algorithm widely used for this purpose with various applications including fraud detection, malware detection, web classification, and recommendation. However, previous methods based on LBP have problems in modeling complex structures of attributed networks because they manually and heuristically select the most important parameter, the propagation strength.

In this chapter, we propose SBP (Supervised Belief Propagation), a scalable and novel inference algorithm which automatically learns the optimal propagation strength by supervised learning. SBP is generally applicable to attributed networks including weighted and signed networks. Through extensive experiments, we demonstrate that SBP generalizes previous LBP-based methods and outperforms previous LBP- and RWR-based methods in real-world networks.

3.1 Motivation

Given an attributed network whose edges have weights or signs, how can we classify its nodes into different categories? Node classification is a crucial task with many applications including anomaly and fraud detection [7, 8, 9, 10], link pre-

diction [11, 12, 13], sign prediction [14], and recommendation [15]. Loopy belief propagation (LBP) [5], an inference algorithm for probabilistic graphical models, has been widely used for solving node classification problems. Intuitively, LBP is based on the notion of *guilt-by-association*: if a user is a drug-abuser, then it is likely that its neighbors are drug-abusers as well. LBP propagates prior knowledge on observed nodes to infer labels or states of unobserved nodes. Due to its simplicity and generality, LBP has been widely used for solving real-world problems including malware detection [8, 16], fraud detection [7, 10], image processing [17], etc.

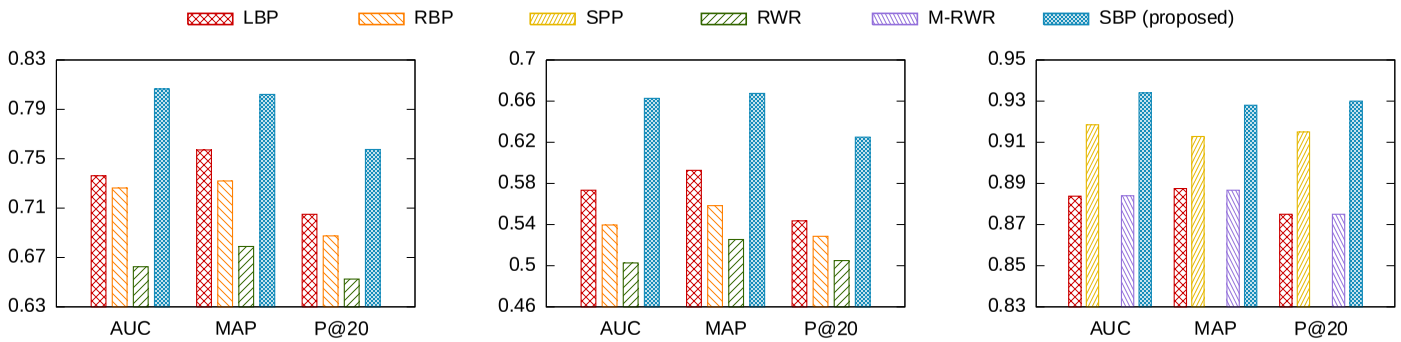
However, previous works on LBP have two main limitations. First, they do not provide an algorithmic way of determining the *propagation strength,* which models the degrees of association between adjacent nodes. Instead, heuristic methods are applied to choose the value with no theoretical justification. The difficulty of manually choosing the propagation strength leads to using only a small number of parameters in LBP-based methods: e.g., many existing works [8, 9, 10] use a single heuristically determined propagation strength to uniformly model all the edges in a network. Second, previous works on LBP do not utilize rich information available in attributed networks since choosing the propagation strength is difficult and thus they need to simplify the networks. Although some LBP-based methods were proposed to use the information in attributed networks, they solely focus on specific types of networks, not general ones [15, 18]. Furthermore, they do not provide a way to choose the propagation strength as well. As a result, this rich information is left unused, although it could give useful insights on the identity of nodes in networks.

In this chapter, we propose SBP (Supervised Belief Propagation), a scal-

able inference algorithm for attributed networks designed to overcome the limitations of previous LBP-based methods. SBP learns optimal propagation strengths which had to be chosen manually and heuristically in previous methods. Moreover, SBP extends the model capacity compared to previous methods: SBP allows the use of multiple parameters to consider rich features in attributed networks in determining the propagation strengths. SBP outperforms state-of-the-art node classification methods based on LBP and RWR (Random Walk with Restart). Our main contributions are the following:

- **Algorithm.** We propose SBP, a scalable inference algorithm for attributed networks. SBP automatically learns optimal propagation strengths and reveals relative importance of each attribute. SBP is general enough to take any attributed network as an input and model its attributes.
- **Accuracy.** SBP shows the highest accuracy in node classification, outperforming previous LBP and RWR based methods. SBP provides up to 15.6% higher AUC in real-world datasets as shown in Figure 3.1.
- **Scalability.** SBP is scalable to large networks, providing linear running time and memory requirement with regard to the number of edges.

The codes and datasets for this chapter are available at `http://datalab.snu.ac.kr/sbp`. The rest of this chapter is organized as follows: Section 3.2 presents preliminaries. Section 3.3 proposes SBP and shows its scalability. Section 5.4 presents experimental results. Section 3.5 introduces related works. We summarize the chapter and give ideas of future work in Section 7.6.



(a) MovieLens (b) Epinions-R (c) Epinions-S

Figure 3.1: Classification accuracy of SBP and baseline approaches in three datasets: (a) MovieLens, (b) Epinions-R, and (c) Epinions-S. SBP shows the highest accuracy in all the datasets: up to 15.6% higher AUC, 12.6% higher MAP, and 15.0% higher P@20 compared to the best competitors. RWR shows a low AUC of 71.6% for Epinions-S, which is not shown in (c).

Table 3.1: Table of symbols.

Symbol	Definition
θ_{ij}	feature vector for edge (i, j)
w	weight vector to be learned
s_p, s_n	positive and negative states
P, N	sets of positive and negative nodes
$(\cdot)_{\text{obs}}$	observed nodes for the propagation step
$(\cdot)_{\text{trn}}$	training nodes for the weight update step
ϵ_{ij}	propagation strength for edge (i, j)
m_{ij}	normalized message for edge (i, j)
b_i	normalized belief for node i
ϕ_i	normalized node potential for node i
$m^*_{ij}(\cdot)$	unnormalized message for edge (i, j)
$b^*_i(\cdot)$	unnormalized belief for node i
$E(\cdot)$	cost function to be minimized
α, β, λ	parameters for gradient update of w
η	number of recursive updates for ∂w

3.2 Preliminaries

We describe preliminaries on SBP (Supervised Belief Propagation). Symbols we use throughout this chapter are summarized in Table 3.1. We first introduce loopy belief propagation (LBP) and then the *edge potential table*, which is used widely to make LBP-based methods applicable to large real-world networks.

3.2.1 Loopy Belief Propagation

A *pairwise Markov random field* (MRF) is a set of discrete random variables whose joint relationships are modeled as an undirected graph. Given variables $X = (X_i)_{i \in \mathcal{V}}$ which are modeled as a graph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of nodes and edges, respectively, the joint probability $p(X = x)$ is

computed by multiplying all the *potentials* ϕ and ψ as

$$p(X = x) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \phi_i(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j),$$

where Z is a normalization constant. A node potential $\phi_i(x_i)$ represents an unnormalized probability of node i being in state x_i without considering the influences from other nodes. An *edge potential* $\psi_{ij}(x_i, x_j)$ represents a joint unnormalized probability of nodes i and j being in states x_i and x_j.

Loopy belief propagation (LBP) is an approximate algorithm to compute the marginal distribution of a pairwise MRF by passing *messages* between the variables [5, 6]. A message $m^*_{ij}(x_j)$ can be considered as an opinion of node i about the probability of node j being in state x_j. LBP uniformly initializes all messages and updates them through iterations until convergence. Equation (3.1) shows how to update $m^*_{ij}(x_j)$ at each iteration where $\mathcal{N}(i)$ denotes the set of neighbors of node i. All the incoming messages from $\mathcal{N}(i)$ except node j are multiplied to compute $m^*_{ij}(x_j)$.

$$m^*_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i) \psi_{ij}(x_i, x_j) \frac{\prod_{k \in \mathcal{N}(i)} m^*_{ki}(x_i)}{m^*_{ji}(x_i)} \tag{3.1}$$

LBP returns *beliefs* as a result. A belief $b_j(x_j)$ is an approximate marginal probability of node j being in state x_j, which is computed from the converged messages and then normalized. Equation (3.2) shows how to compute the un-

normalized beliefs and then how to normalize them accordingly.

$$
\begin{aligned}
b_j(x_j) &= \frac{b_j^*(x_j)}{\sum_{x'_j} b_j^*(x'_j)} \text{ where} \\
b_j^*(x_j) &= \phi_j(x_j) \prod_{i \in \mathcal{N}(j)} m_{ij}^*(x_j)
\end{aligned}
\tag{3.2}
$$

A popular application of LBP is node classification whose goal is to infer the states of unobserved nodes based on the known states of observed nodes. LBP solves this by assigning different node potentials to the nodes and then propagating the messages. For instance, assuming random variables having two states, LBP gives a node potential $(0.9, 0.1)$ or $(0.1, 0.9)$ to a observed node based on its observed state. On the other hand, LBP gives a node potential $(0.5, 0.5)$ to the unobserved nodes. LBP propagates the messages from the biased potentials of the observed nodes and classifies the unobserved nodes. Although it is not guaranteed, the messages often converge in a small number of iterations for most real-world networks.

3.2.2 Edge Potential Table

LBP has been widely used in solving real-world problems due to its simplicity and scalability [7, 10, 9]; running time of LBP scales linearly with the number of edges. Nevertheless, previous methods have difficulty in assigning edge potentials because it requires too many parameters. Thus, a majority of existing methods tend to simplify the model and assume that all edges in a graph have the same edge potential, which is represented by the *edge potential table*. The use of edge potential tables makes such methods simple enough to be applicable to large networks by reducing the number of parameters.

Table 3.2: Edge potential table with propagation strength ϵ for a graph whose nodes have two possible states: positive or negative.

State	positive	negative
positive	ϵ	$1-\epsilon$
negative	$1-\epsilon$	ϵ

Given a real-world network modeled as a pairwise MRF, an edge potential table is defined as a $k \times k$ table where k is the number of states of its variables. Table 3.2 shows an example of an edge potential table in a graph whose nodes have two states: positive and negative. Given a *propagation strength* ϵ between 0 and 1, the table denotes that the joint probability of adjacent nodes is determined by ϵ. Formally, the edge potential $\psi_{ij}(x_i, x_j)$ for each edge (i, j) is given as follows:

$$\psi_{ij}(x_i, x_j) = \begin{cases} \epsilon, & \text{if } x_i = x_j \\ 1-\epsilon, & \text{otherwise} \end{cases} \tag{3.3}$$

It has been a challenging problem to choose the right value of the propagation strength [9]. A common workaround is a grid search which finds the best parameter from a finite set of reasonable values by running the algorithm for each possible case. For example, we choose ϵ in $\{0.6, 0.51, 0.501, 0.5001\}$. However, such an approach restricts introducing more parameters since it takes exponential time in the number of parameters. Previous methods [15] use only one or two propagation strengths for rich attributed networks although their attributes contain meaningful information about the inference.

We introduce Figure 3.2 to further elaborate on the limitation of the current model used in existing LBP-based methods. The figure shows an attributed

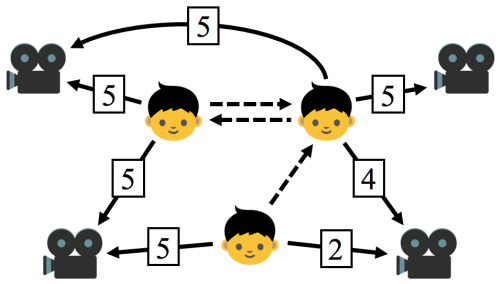


Figure 3.2: Attributed network which contains two types of nodes (users and items) and edges (trusts and reviews). The dashed arrows represent trusts, and the solid arrows represent reviews with ratings between 1 and 5.

network which contains two types of nodes (users and items) and edges (trusts and reviews). The review edges contain ratings between 1 and 5, while the trust edges have no additional information. Assuming that we want to classify the items into *recommended* and *not recommended* for a given user, it is impractical to find the best propagation strength for each type and rating using the grid search. The typical approach is to simplify the edges by eliminating the attributes. However, such simplification suffers from the loss of rich information which leads to poor accuracy.

3.3 Proposed Method: SBP

In this section, we propose SBP (Supervised Belief Propagation), a scalable inference algorithm for attributed networks with the following improvements:

- SBP automatically learns optimal propagation strengths which previous methods have manually chosen.
- SBP utilizes rich information of attributed networks by considering these attributes in the message propagation.

Algorithm 1: SBP (Supervised Belief Propagation)

Input: attributed network G, sets P and N of positive and negative nodes, and node potential ϕ

Output: beliefs b for all nodes

1: $P_{\text{obs}}, P_{\text{trn}} \leftarrow$ randomly split P into two sets
2: $N_{\text{obs}}, N_{\text{trn}} \leftarrow$ randomly split N into two sets
3: $w \leftarrow$ an initial weight vector
4: **while** convergence criterion of w is not met **do**
5: $\quad b, m \leftarrow \mathit{propagate}(w, P_{\text{obs}}, N_{\text{obs}}, \phi)$
6: $\quad w \leftarrow \mathit{weight\_update}(w, b, m, P_{\text{trn}}, N_{\text{trn}})$
7: **end while**
8: $b, m \leftarrow \mathit{propagate}(w, P, N, \phi)$
9: **return** b

3.3.1 Overview

SBP is an inference algorithm which overcomes the limitations of previous LBP-based methods and thus is generally applicable to attributed networks. Given an attributed network whose attributes are encoded as feature vectors, SBP learns *weights* of the features to determine the propagation strengths; edges with different feature vectors have different propagation strengths depending on the learned weights. This enables SBP to utilize rich information in attributed networks, resulting in more accurate inference than existing methods.

SBP initializes a weight vector w and alternates two main steps to train it. In the *propagation step*, SBP computes the messages and beliefs based on the current w. This step is similar to the standard LBP algorithm which uses fixed values for the propagation strengths. In the *weight update step*, SBP updates w based on the computed messages and beliefs so that w moves toward a local optimum. These steps are described in detail in Section 3.3.3 and 3.3.4, respectively.

The main idea of weight optimization is to train the weight vector w so that the beliefs of nodes are in accordance with their true states. For this purpose, given a set of nodes whose true states are known, we hide the labels of some nodes to use them as correct answers in training w. We call these nodes as *training nodes* and the others as *observed nodes*. We use only the observed nodes as evidence of inference and use the training nodes to update w.

Although the main ideas of SBP are generally applicable to multi-label classification, we focus on a binary case due to its simplicity. In binary classification, we assume that every node has two possible states s_p and s_n, and we call a node positive or negative if its state is known as s_p or s_n, respectively.

Algorithm 1 shows SBP. In lines 1 and 2, we split each set of nodes into observed and training, as we discussed above, in order to separate the nodes for the propagation and weight update steps. In lines 3 to 7, we initialize w and then iteratively update it until it converges. After the convergence, we compute the beliefs using all the positive and negative nodes and return them. Figure 3.3 shows a summary of the algorithm.

The stopping criterion in line 4 of Algorithm 1 for updating w is either one of the following: 1) the difference between w in consecutive iterations is within a small threshold, or 2) the maximum number of iterations is reached. This criterion is also used for the message updates in the propagation step.

3.3.2 Encoding Attributed Networks

We encode an attributed network as a directed graph where each edge (i, j) contains a feature vector θ_{ij} generated from the original attributes. SBP determines propagation strength ϵ_{ij} customized for edge (i, j) using its feature

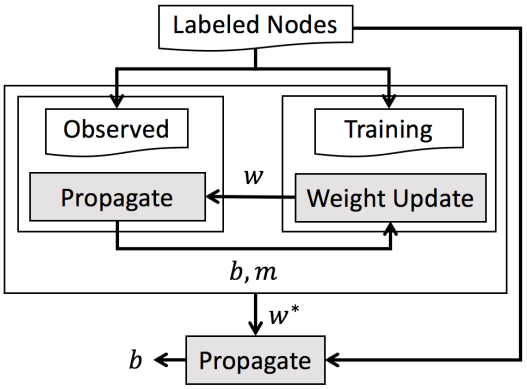


Figure 3.3: Flowchart of SBP representing Algorithm 1. SBP splits labeled nodes into observed and training, and then alternates propagation and weight update steps until w converges. After the convergence, SBP computes beliefs of all the nodes using the learned w^* and returns them.

vector θ_{ij} and the globally trained weight vector w. As there can be multiple possible encodings for a single attributed network, choosing a proper encoding θ_{ij} from the attributes is important for the performance of the algorithm.

For instance, let us consider the attributed network in Figure 3.2, which is discussed in Section 3.2.2. The network contains two kinds of attributes: edge type and rating. While the edge type attribute is naturally encoded as a binary vector of length 1, the rating can be encoded in various ways: 1) a vector of length 1 with the raw score or 2) a one-hot encoded vector of length 5.

At first glance, the first encoding seems natural. However, it is problematic in that discrete attributes are modeled as a continuous integer. This assumes a strict linear scaling of preferences where rating 5 is exactly 5 times higher than rating 1 although their exact correlations in the network are unknown. On the contrary, the second separates all the ratings so that each represents a distinctive evaluation. In SBP, which learns different optimal weights for each

Table 3.3: Two types of feature vectors with different encodings for the network of Figure 3.2. While the first encoding assumes the linear correlation between the ratings, the second makes SBP learn independent weights for each feature.

Edge Type	**Encoding 1**	**Encoding 2**
Trust	$(1, 0)$	$(1, 0, 0, 0, 0, 0)$
Review with rating 1	$(0, 1)$	$(0, 1, 0, 0, 0, 0)$
Review with rating 2	$(0, 2)$	$(0, 0, 1, 0, 0, 0)$
Review with rating 3	$(0, 3)$	$(0, 0, 0, 1, 0, 0)$
Review with rating 4	$(0, 4)$	$(0, 0, 0, 0, 1, 0)$
Review with rating 5	$(0, 5)$	$(0, 0, 0, 0, 0, 1)$

feature element, the second gives better results than the first one.

After encoding each attribute as a vector, we concatenate them to generate the feature vectors. Table 3.3 shows two types of feature vectors with different encodings for the network in Figure 3.2. The first element represents the edge type as a binary integer and the rest represent the rating.

3.3.3 Propagation Step

In the propagation step (lines 5 and 8 in Algorithm 1), the messages and beliefs are computed based on the current weight vector w. After the messages converge through iterations, the beliefs are computed from the converged messages. The message propagation in SBP have two main differences compared to that in previous LBP-based methods: 1) the messages are propagated with different strengths customized for the feature vector of each edge and 2) computations of the messages and beliefs are optimized for binary networks.

Propagation strength ϵ_{ij} which is customized for each edge (i, j) is modeled as Equation (3.4). Its value is determined by the inner product between the feature vector θ_{ij} of edge (i, j) and the current weight vector w, and is

interpreted as a probability between 0 and 1. As we model ϵ_{ij} in this way, we 1) easily consider both θ_{ij} and w in determining the propagation strength, and 2) improve stability of the algorithm by avoiding extreme values of ϵ_{ij}. It also can be considered as a function of θ_{ij} given w as a parameter. Thus, its value changes during the algorithm as w is updated at each iteration.

$$\epsilon_{ij} = (1 + \exp(-\theta_{ij}^{\mathrm{T}} w))^{-1} \tag{3.4}$$

Next, we describe efficient computations of SBP for binary networks. We optimize Equations (3.1) and (3.2) of LBP assuming a binary network whose nodes have two possible states: s_p and s_n. As a result, we have Equations (3.5), (3.6), and (3.7) which are more efficient. Note that the simplified equations lead to the same results as those from the original equations.

First, we normalize the messages and use only the positive ones. It gives the following advantages: 1) we avoid numerical underflow when multiplying a lot of messages on high-degree nodes, 2) it saves the cost of space to store both messages, and 3) it simplifies the equations. Resulting beliefs computed from the converged messages remain the same because any constant multiplied to the messages cancels out when we normalize the beliefs. Equation (3.5) shows how to compute the normalized message m_{ij} from the unnormalized ones.

$$m_{ij} = m_{ij}^*(s_p)/(m_{ij}^*(s_p) + m_{ij}^*(s_n)) \tag{3.5}$$

Then, unnormalized beliefs $b_j^*(s_p)$ and $b_j^*(s_n)$ for each node j are computed as in Equation (3.6) from the normalized messages and then normalized as in Equation (3.2). Note that the node potentials ϕ_j are normalized in the same

way as the messages; ϕ_j and $1 - \phi_j$ represent $\phi_j(s_p)$ and $\phi_j(s_n)$, respectively.

$$\begin{aligned} b_j^*(s_p) &= \phi_j \prod_{i \in \mathcal{N}(j)} m_{ij} \\ b_j^*(s_n) &= (1 - \phi_j) \prod_{i \in \mathcal{N}(j)} (1 - m_{ij}) \end{aligned} \tag{3.6}$$

Lastly, we rewrite Equation (3.1) as follows. First, we replace edge potential $\psi_{ij}(x_i, x_j)$ by propagation strength ϵ_{ij} as discussed in Section 3.2.2; the difference is that we use customized strength for each edge instead of the global strength. Second, we replace the unnormalized messages $m_{ij}^*(x_i, x_j)$ by the normalized ones. Third, we replace the multiplications of incoming messages by the pre-computed beliefs to avoid duplicate calculations [19]. Resulting equations are given as Equation (3.7).

$$\begin{aligned} m_{ij}^*(s_p) &\leftarrow \epsilon_{ij} \frac{b_i}{m_{ji}} + (1 - \epsilon_{ij}) \frac{1 - b_i}{1 - m_{ji}} \\ m_{ij}^*(s_n) &\leftarrow (1 - \epsilon_{ij}) \frac{b_i}{m_{ji}} + \epsilon_{ij} \frac{1 - b_i}{1 - m_{ji}} \end{aligned} \tag{3.7}$$

Algorithm 2 shows a summary of the propagation step. In lines 1 to 3, we initialize the node potentials. The normalized potential ϕ is given as a parameter. Node potentials for the unobserved nodes are uniformly set to 0.5. In lines 4 to 5, we initialize all the messages and propagation strengths. In lines 6 to 9, we iteratively update the messages until they converge. In line 10, we return the computed messages and beliefs.

Algorithm 2: *propagate*$(\cdot)$

Input: weight vector w, sets P_{obs} and N_{obs} of positive and negative observed nodes, and node potential ϕ

Output: computed beliefs b and messages m

1: $\phi_i \leftarrow 0.5$ for each node i not in P_{obs} and N_{obs}
2: $\phi_i \leftarrow \phi$ for each node i in P_{obs}
3: $\phi_i \leftarrow 1 - \phi$ for each node i in N_{obs}
4: $m_{ij} \leftarrow 0.5$ for each edge (i, j)
5: $\epsilon_{ij} \leftarrow (1 + \exp(-\theta_{ij}^{\mathrm{T}} w))^{-1}$ for each edge (i, j)
6: **while** convergence criterion of m is not met **do**
7: $\quad b \leftarrow$ compute beliefs using m and ϕ
8: $\quad m \leftarrow$ compute messages using ϵ, m, and b
9: **end while**
10: **return** b, m

3.3.4 Weight Update Step

In the weight update step (line 6 in Algorithm 1), we update the weight vector w using a gradient-based approach. First, we define a cost function of w which we try to minimize. The cost function should be defined in a way that its minimization makes better classification. Second, we differentiate the cost function with respect to w and compute the gradient. Finally, we update w to the negative direction of the derivative.

The cost function $E(w)$ is defined as Equation (3.8), which is computed from the pairwise differences between the beliefs of positive and negative training nodes. Given an increasing loss function h, $E(w)$ is minimized as the beliefs b_n of negative nodes are minimized and the beliefs b_p of positive nodes are maximized. An L2 regularization parameter λ is introduced to avoid overfitting and

decrease the model complexity.

$$E(w) = \lambda ||w||_2^2 + \sum_{p \in P_{\text{trn}}} \sum_{n \in N_{\text{trn}}} h(b_n - b_p) \tag{3.8}$$

We use the loss function $h(x) = (1 + \exp(-x/d))^{-1}$ which approximates the step function when d is small. It is known that AUC (area under the ROC curve) of binary classification is maximized when h is used as a loss function and d is small enough [12, 20]. We set d to 0.0001.

Then, we differentiate the cost function as follows, where an error term $x = b_n - b_p$ is introduced to simplify the equation. $\partial h(x)/\partial x$ is easily computed since $h(x)$ is a simple sigmoid function of x. The problem is to compute the derivatives of beliefs, which is discussed in Section 3.3.5.

$$\frac{\partial E(w)}{\partial w} = 2\lambda w + \sum_{p \in P_{\text{trn}}} \sum_{n \in N_{\text{trn}}} \frac{\partial h(x)}{\partial x} \left(\frac{\partial b_n}{\partial w} - \frac{\partial b_p}{\partial w} \right)$$

$$\text{where } \frac{\partial h(x)}{\partial x} = d^{-1} h(x)(1 - h(x))$$

Assuming we have the derivatives of beliefs, we differentiate the cost function as $w' = \partial E/\partial w$ and update the weight vector w, as shown in Equation (3.9). $\min\{\cdot, \cdot\}$ selects the vector whose L2 norm is smaller than the other. New parameters α and β are introduced; α is a step size that determines quality and speed of the convergence, and β limits the L2 norm of the gradient step to avoid a steep change.

$$w \leftarrow w - \min \left\{ \alpha w', \frac{\beta}{||\alpha w'||_2} \alpha w' \right\} \tag{3.9}$$

Algorithm 3: *weight_update*$(\cdot)$

Input: beliefs b, messages m, and sets P_{trn} and N_{trn} of positive and negative training nodes

Output: updated weight vector w

1: $b' \leftarrow \mathit{differentiate}(b, m)$
2: $w' \leftarrow 2\lambda w$
3: **for** p in P_{trn} and n in N_{trn} **do**
4: $\quad h \leftarrow (1 + \exp(-d^{-1}(b_n - b_p)))^{-1}$
5: $\quad w' \leftarrow w' + d^{-1}h(1-h)(b'_n - b'_p)$
6: **end for**
7: **return** $w - \max\{\alpha w', \frac{\beta}{|\alpha w'|}\alpha w'\}$

Algorithm 3 shows a summary of the weight update step. In line 1, we differentiate the beliefs. In lines 2 to 6, we compute the derivative of the cost function with regularization. In line 7, we update the weight vector and return the result.

3.3.5 Derivatives for the Weight Update

Here we describe how to approximately compute the derivatives of the beliefs with regard to the weight vector (line 1 of Algorithm 3). The problem is that the beliefs are not expressed as closed-form functions of w since they are computed from the messages which converge through iterations. Thus, we use the chain rule to express $\partial b_j / \partial w$ as a function of $\partial b_j / \partial m_{ij}$ and $\partial m_{ij} / \partial w$ for each neighboring node i, and then Lemma 1 to replace $\partial b_j / \partial m_{ij}$. Equation (3.10) shows the result.

$$\begin{aligned} \frac{\partial b_j}{\partial w} &= \sum_{i \in \mathcal{N}(j)} \frac{\partial b_j}{\partial m_{ij}} \frac{\partial m_{ij}}{\partial w} \\ &= \sum_{i \in \mathcal{N}(j)} \frac{b_j(1-b_j)}{m_{ij}(1-m_{ij})} \frac{\partial m_{ij}}{\partial w} \end{aligned} \tag{3.10}$$

Lemma 1. *Belief b_j of node j is differentiated by an incoming message m_{ij} from a neighboring node i as*

$$\frac{\partial b_j}{\partial m_{ij}} = \frac{b_j(1-b_j)}{m_{ij}(1-m_{ij})}.$$

Proof. We differentiate Equation (3.6) to get the derivatives of the unnormalized beliefs $b_j^*(s_p)$ and $b_j^*(s_n)$ as

$$\frac{\partial b_j^*(s_p)}{\partial m_{ij}} = \frac{b_j^*(s_p)}{m_{ij}} \qquad\qquad \frac{\partial b_j^*(s_n)}{\partial m_{ij}} = -\frac{b_j^*(s_n)}{1-m_{ij}}.$$

Then, we differentiate the first part of Equation (3.2) to express $\partial b_j / \partial m_{ij}$ as a function of the above derivatives. After substituting the values, we get the equation in the lemma. □

Next, we apply the chain rule again to express the message derivative as Equation (3.11). This is not a closed-form solution because the derivative terms in the form of $\partial m_{ij}/\partial w$ exist in both sides of the equation. Unlike the messages which converge through iterations in most real-world networks, the derivatives are shown to diverge because of positive loops existing in a cyclic network. Thus, we introduce a recursion parameter η to limit the number of updates.

$$\frac{\partial m_{ij}}{\partial w} = \underbrace{\frac{\partial m_{ij}}{\partial \epsilon_{ij}} \frac{\partial \epsilon_{ij}}{\partial w}}_{\text{base terms}} + \sum_{k \in \mathcal{N}(i) \setminus j} \underbrace{\frac{\partial m_{ij}}{\partial m_{ki}} \frac{\partial m_{ki}}{\partial w}}_{\text{recursive terms}} \tag{3.11}$$

We separate the derivatives in the right hand side of Equation (3.11) into the base and recursive terms. The first base term $\partial m_{ij}/\partial \epsilon_{ij}$ is computed from

Lemma 2 and the second base term $\partial\epsilon_{ij}/\partial w$ is computed as $\epsilon_{ij}(1-\epsilon_{ij})\theta_{ij}$ since ϵ_{ij} is a simple sigmoid function of w. The coefficient $\partial m_{ij}/\partial m_{ki}$ of the recursive terms is computed from Lemma 3. As a result, we get Equation (3.12):

$$\frac{\partial m_{ij}}{\partial w} = k_1(b_i - m_{ji})\theta_{ij} + \sum_{k\in\mathcal{N}(i)\setminus j} \frac{\epsilon_{ij} - k_2}{k_3}\frac{\partial m_{ki}}{\partial w}, \tag{3.12}$$

where the following nonnegative constants k_1, k_2 and k_3 are introduced to simplify the equation:

$$\begin{aligned}
k_1 &= \epsilon_{ij}(1-\epsilon_{ij})(b_i + m_{ji} - 2b_i m_{ji})^{-1} \\
k_2 &= m_{ij} + m_{ji} - 2m_{ij}m_{ji} \\
k_3 &= m_{ki}(1-m_{ki})(b_i + m_{ji} - 2b_i m_{ji})b_i^{-1}(1-b_i)^{-1}.
\end{aligned}$$

Lemma 2. *Message m_{ij} of edge (i,j) is differentiated by the propagation strength ϵ_{ij} of the same edge as*

$$\frac{\partial m_{ij}}{\partial \epsilon_{ij}} = \frac{b_i - m_{ji}}{b_i + m_{ji} - 2b_i m_{ji}}.$$

Proof. We differentiate Equation (3.7) by ϵ_{ij} to get the derivatives of the unnormalized messages $m^*_{ij}(s_p)$ and $m^*_{ij}(s_n)$:

$$\begin{aligned}
\frac{\partial m^*_{ij}(s_p)}{\partial \epsilon_{ij}} &= \frac{b_i}{m_{ji}} - \frac{1-b_i}{1-m_{ji}} \\
\frac{\partial m^*_{ij}(s_n)}{\partial \epsilon_{ij}} &= -\frac{b_i}{m_{ji}} + \frac{1-b_i}{1-m_{ji}}.
\end{aligned}$$

Then, we differentiate Equation (3.5) to express $\partial m_{ij}/\partial\epsilon_{ij}$ as a function of the above derivatives. After substituting the values, we get the equation in the

lemma. □

Lemma 3. *When nodes k and j are both neighbors of node i, message m_{ij} is differentiated by message m_{ki} as*

$$\frac{\partial m_{ij}}{\partial m_{ki}} = \frac{z_{ij} b_i (1 - b_i)}{m_{ki}(1 - m_{ki})},$$

where $z_{ij} = (\epsilon_{ij} - m_{ji} - m_{ij} + 2m_{ij}m_{ji})(b_i + m_{ji} - 2b_i m_{ji})^{-1}$.

Proof. We differentiate Equation (3.7) by m_{ki} to get the derivatives of the unnormalized messages $m^*_{ij}(s_p)$ and $m^*_{ij}(s_n)$:

$$\begin{aligned}
\frac{\partial m^*_{ij}(s_p)}{\partial m_{ki}} &= \frac{\epsilon_{ij}}{m_{ji}} \frac{\partial b_i}{\partial m_{ki}} - \frac{1 - \epsilon_{ij}}{1 - m_{ji}} \frac{\partial b_i}{\partial m_{ki}} \\
&= \frac{\epsilon_{ij} - m_{ji}}{m_{ji}(1 - m_{ji})} \frac{b_i(1 - b_i)}{m_{ki}(1 - m_{ki})} \\
\frac{\partial m^*_{ij}(s_n)}{\partial m_{ki}} &= \frac{1 - \epsilon_{ij} - m_{ji}}{m_{ji}(1 - m_{ji})} \frac{b_i(1 - b_i)}{m_{ki}(1 - m_{ki})}
\end{aligned}$$

Then, we differentiate Equation (3.5) to express $\partial m_{ij}/\partial m_{ki}$ as a function of the above derivatives. After substituting the values, we get the equation in the lemma. □

Algorithm 4 shows a summary of the differentiation process. In lines 1 to 4, we compute the base terms in Equation (3.12) and the coefficient z_{ij} in Lemma 3. In line 5, we initialize the message derivatives using the base terms. In lines 6 to 12, we iteratively update the message derivatives η times. Specifically, in line 7, we compute the recursive terms in Equation (3.12) to avoid duplicate computations, and in lines 8 to 11, we update all the message derivatives. In lines 13 to 14, we compute the belief derivatives using the computed message

Algorithm 4: *differentiate*$(\cdot)$

Input: beliefs b and messages m
Output: derivatives $\partial b / \partial w$ of the beliefs

for each edge (i, j) **do**
  $(m'_{ij})_{\text{base}} \leftarrow \frac{\epsilon_{ij}(1-\epsilon_{ij})(b_i - m_{ji})}{b_i + m_{ji} - 2b_i m_{ji}}$
  $z_{ij} \leftarrow \frac{\epsilon_{ij} - m_{ji} - m_{ij} + 2m_{ij}m_{ji}}{b_i + m_{ji} - 2b_i m_{ji}}$
end for
$m'_{ij} \leftarrow (m'_{ij})_{\text{base}}$ for each edge (i, j)
for η times **do**
  $m''_{ij} \leftarrow \frac{b_j(1-b_j)}{m_{ij}(1-m_{ij})} m'_{ij}$ for each edge (i, j)
  for each edge (i, j) **do**
    $(m'_{ij})_{\text{new}} \leftarrow (m'_{ij})_{\text{base}} + z_{ij} \sum_{k \in \mathcal{N}(i) \setminus j} m''_{ki}$
  end for
  substitute m'_{ij} with $(m'_{ij})_{\text{new}}$ for each edge (i, j)
end for
$b'_j \leftarrow \sum_{i \in \mathcal{N}(j)} \frac{b_j(1-b_j)}{m_{ij}(1-m_{ij})} m'_{ij}$ for each node j
return b'

derivatives.

3.3.6 Scalability

We show that time and space complexities of SBP are linear with the number of edges in a given network. The complexities are given by Lemmas 4 and 5. We have two variables T_1 and T_2 representing the numbers of iterations. T_1 is the number of iterations for the propagation step and is relatively small; it does not exceed 10 for all the networks we use in the experiments. T_2 is the number of weight updates and varies from 20 to 100 depending on the network. Assuming the number of training nodes are fixed, the term $|\theta||P_{\text{trn}}||N_{\text{trn}}|T_2$ added in the time complexity can be considered as a constant.

Lemma 4. *Space complexity of SBP is* $O(|\theta||E|)$ *where* $|\theta|$ *is the number fea-*

tures and $|E|$ is the number of edges.

Proof. The largest variables SBP needs to store are the message derivatives. Since every edge has two derivatives of size $|\theta|$ when comparing the old and new, the space complexity is proportional to the number of edges and features. □

Lemma 5. *Time complexity of SBP is given as*

$$O(((T_1 + \eta|\theta|)|E| + |\theta||P_{\text{trn}}||N_{\text{trn}}|)T_2),$$

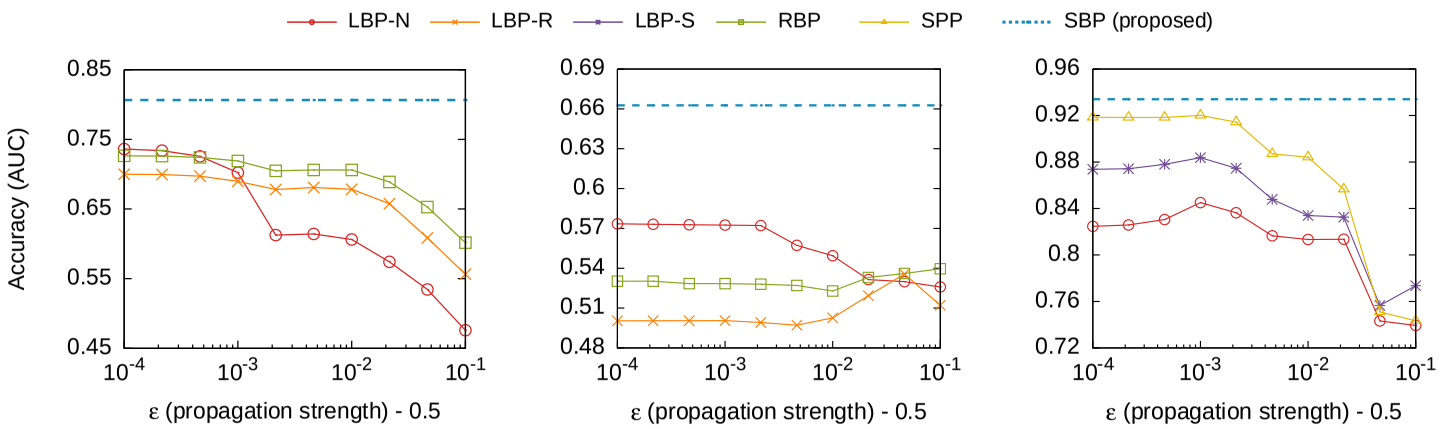
where $|\theta|$ is the number of features, $|E|$ is the number of edges, T_1 is the number of iterations for the propagation step, η is the number of derivative updates for the update step, $|P_{\text{trn}}|$ and $|N_{\text{trn}}|$ are the number of positive and negative training nodes, respectively, and T_2 is the number of weight updates.

Proof. Time complexity of the propagation step is $O((|\theta| + T_1)|E|)$ since all the messages are updated T_1 times. Time complexity of the update step is $O(|\theta|(\eta|E| + |P_{\text{trn}}||N_{\text{trn}}|))$ since the running time mostly depends on the computations of message derivatives. We prove the lemma by summing them up and multiplying T_2 as SBP alternates them T_2 times. □

3.4 Experiments

We present experimental results of SBP to answer the following questions:

- **Q1. Accuracy (Section 3.4.2).** How accurately does SBP classify nodes in attributed networks?
- **Q2. Learning process (Section 3.4.3).** How accurately does SBP find



(a) MovieLens (b) Epinions-R (c) Epinions-S

Figure 3.4: AUCs of the LBP-based methods for varying values of the propagation strength. Each plot illustrates the result for each dataset: (a) MovieLens, (b): Epinions-R, and (c) Epinions-S. The existing methods are shown to be sensitive to the propagation strength. SBP, our proposed method denoted by the dashed blue lines, provides the highest accuracies in all the datasets.

the optimal weights? How do values of the cost function and AUC change during iterations?

- **Q3. Scalability (Section 3.4.4).** How does running time of SBP scale with regard to the size of a network?

3.4.1 Experimental Settings

Datasets We use three publicly available datasets which are summarized in Table 3.4. Epinions-R [21] is a heterogeneous network which we introduced in Section 3.2.2. It contains two kinds of nodes (users and items) and edges (reviews and trusts). The review edges connect users and items with ratings between 1 and 5, while the trust edges connect only users with no additional information. Epinions-S [22] is a signed social network whose edges are either positive or negative. The sign of an edge (u, v) indicates either a positive or negative feeling of user u towards user v. MovieLens [23] is a bipartite review network for movies from the users of MovieLens, whose edges contain integer ratings between 1 and 5.

Encoding We model the networks based on one-hot encoded feature vectors to make SBP learn independent weights for the feature elements. The simplest network is Epinions-S whose attributes (signs) are easily modeled as one-hot encoded vectors of length 2. On the other hand, we have two factors to consider in MovieLens: directions and ratings. We ignore the directions of edges since the network is bipartite; the edges represent undirected relationships between users and movies. Then, we encode the ratings as one-hot encoded vectors of length 5. Epinions-R is modeled as described in Section 3.3.2. The ratings are

Table 3.4: Summary of the datasets.

Dataset	Nodes	Edges	Attributes
Epinions-R[1]	189,028	1,152,005	ratings and trusts
Epinions-S[2]	131,828	841,372	signs (trusts or distrusts)
MovieLens[3]	9,940	1,000,209	ratings (1 to 5)

[1] `http://www.trustlet.org/downloaded_epinions.html`
[2] `http://www.trustlet.org/extended_epinions.html`
[3] `http://grouplens.org/datasets/movielens/1m`

modeled as the same in MovieLens and an additional element representing the edge type is concatenated. As a result, the feature vectors are of length 6.

Competitors We compare SBP with node classification methods which are based on Loopy Belief Propagation (LBP) and Random Walk with Restart (RWR). LBP is an inference algorithm introduced in Section 3.2, which is the basis of SBP. RWR [24] is an algorithm to compute node relevance. Starting from a seed node, it walks through other nodes and jumps back to the seed node with a certain probability. As a result, nodes near the seed node are likely to have high visiting probabilities compared to the ones far way from the seed node. Since LBP and RWR are both based on the notion of guilt-by-association, they share common characteristics [25].

RBP [15] and SPP [18] are LBP-based methods for review and signed networks, respectively. RBP uses the rating information to assign different node potentials and neglects edges with low ratings to ensure homophily relationships between the nodes. SPP uses two edge potential tables to model different propagation strengths of positive and negative edges. M-RWR [26] is an RWR-based method for signed networks, which runs RWR for each subgraph consisting of only positive or negative edges, and then subtracts the probabilities.

Furthermore, we use various LBP settings as the baselines of SBP. We use three baseline methods LBP-N, LBP-S, and LBP-R whose strategies of determining the propagation strengths are chosen heuristically. LBP-N is the simplest method which assumes a uniform propagation strength by ignoring the attributes. LBP-S works for Epinions-S; it assumes propagation strengths of ϵ and $\epsilon/2$ for the positive and negative edges, respectively. LBP-R works for the review networks; it models a linear scaling of propagation strengths of the review edges. In other words, it assumes the propagation strength of $r(\epsilon/5)$ for the edges with rating r.

Experimental process Since the nodes in the networks are not initially labeled, we pick seed nodes for each network and then label other nodes based on each seed node. Given a seed node u, we label the others as follows: for MovieLens and Epinions-R, items that received ratings 5 from node u are positive, and items that received ratings below 5 are negative [15]. For Epinions-S, users that received the trust edges from node u are positive, and users that received the distrust edges are negative. Labels are not defined for the nodes which are not connected to the seed node. We uniformly set node potential ϕ to 0.55 in all the LBP-based methods, except RBP which uses its own node potential values, since they generally give the best accuracies for the datasets. In other words, we set $\phi_i(s_p)$ and $\phi_i(s_n)$ of node i to 0.55 and 0.45, respectively. In SBP, we set the number η of recursive updates to 1 because the results are shown to be insensitive to η.

Evaluation We use three kinds of evaluation metrics: the area under the ROC curve (AUC), mean average precision (MAP), and precision at k (P@k).

They are computed from the beliefs and visiting probabilities of the test nodes whose labels are hidden. AUC is generated from a result of binary classification by plotting the true positive rates against the false positive rates at various threshold settings. MAP is the mean of the average precision for each query, where a query represents a seed user in our experiments. P@k is the ratio of true positive nodes from the top-k ranked nodes. We set k to 20.

3.4.2 Classification Accuracy

We randomly pick k_1 seed nodes for each network, run all the methods for each seed node, and average the accuracies. Each seed node is picked from the nodes connected to at least k_2 positive and k_2 negative nodes since the propagation does not work well when the number of observed nodes is too small. Then, we do the following for each seed node: we 1) randomly sample k_2 nodes from each set of positive and negative nodes to balance the number of nodes in both sets, 2) divide each set of sampled nodes into training and test, 3) run the algorithms using only the training sets, and 4) compute the classification accuracies for the test sets. We set k_1 to 20 and k_2 to 80.

Figure 3.4 shows accuracies of the LBP-based methods for varying values of the propagation strength. Values of ϵ equal to or less than 0.5 are not included in the experiment since they violate the assumption of guilt-by-association. As seen in the figure, accuracies of the existing methods significantly change depending on its value. On the other hand, SBP, our proposed method which automatically learns the propagation strengths, shows higher accuracies than the others even when compared to their optimal performances. This is because SBP learns different propagation strengths of the edges while the others neglect

the attributes or depend on the heuristically determined strengths.

Figure 3.1 shows overall results comparing SBP to the other methods in three evaluation metrics. LBP represents the best baseline method for each dataset; LBP-N for MovieLens and Epinions-R, and LBP-S for Epinions-S. SBP shows the highest accuracies for all the datasets. Since performances of the LBP-based methods depend on the values of the propagation strength, we use the best ones found in the experiments of Figure 3.4 for a fair comparison. Note that different groups of competing methods are used for each network; SPP and M-RWR are used only for Epinions-S, and RBP is used only for Epinions-R and MovieLens. This is because they are designed for specific types of networks, not general ones.

3.4.3 Learning Process

Figures 3.5 and 3.6 show the iterative process of SBP learning the optimal weight vector w in the experiment of MovieLens. Figure 3.5 shows that the elements of w converge through the iterations starting from the initial values of 0.001, although it is not theoretically guaranteed. The kth element of w, denoted by w_k, represents the propagation strength of the edges with rating k since the feature vectors in the network are one-hot encoded. Figure 3.6 shows the values of the cost function $E(w)$ and AUC during the same iterations. The costs are minimized for both training and test sets as the weights are updated, resulting in increased AUCs.

We note the following observations in Figure 3.5. First, the elements show different patterns as the iteration proceeds; w_3 and w_4 rapidly increase at first, and then slowly decrease until the convergence. It shows that the weights are

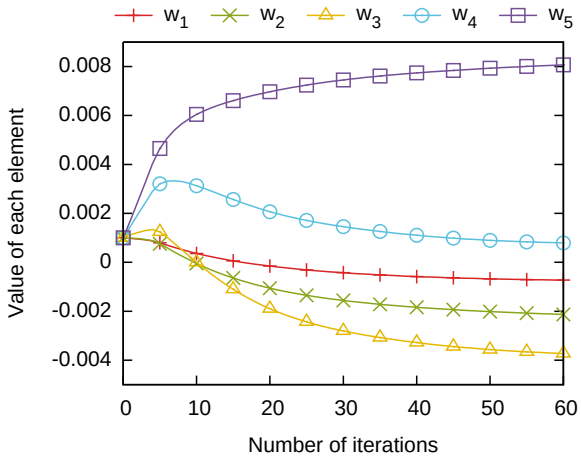


Figure 3.5: Changing values of the weight elements in w during the iterations for MovieLens. The weights w_4 and w_5 for high ratings increase above zero, while the others decrease below zero. All the weights are initialized to 0.0001.

updated toward a local optimum. Second, the optimal weights for low ratings (from 1 to 3) are sorted in the reverse order; w_1 converges to a higher value than w_3. This is because the ratings in the dataset are biased toward high ratings; the average of all the ratings is 3.58, and the number of rating 3 is greater than the sum of the numbers of ratings 1 and 2. Thus, rating 3 possibly represents a lower evaluation than ratings 1 and 2 in this dataset, and the result shows that SBP is capable of learning dataset-specific relationships without any given prior knowledge.

3.4.4 Scalability

We measure running time of SBP for various networks to verify the linear scalability proved in Lemma 5. To generate smaller networks with similar characteristics, we sample principal submatrices from the adjacency matrix of each

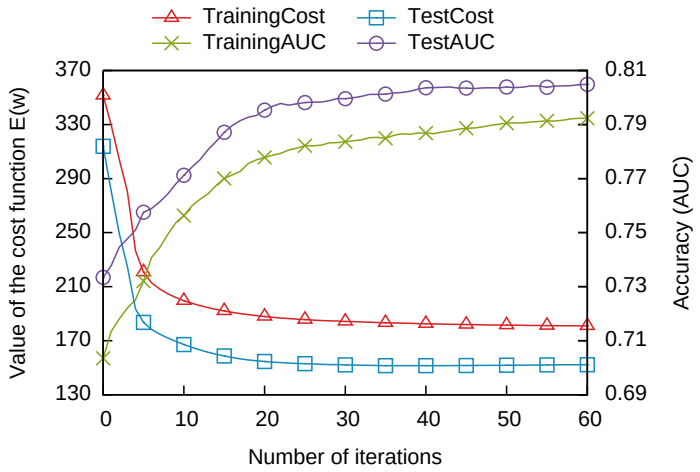


Figure 3.6: Changing values of the cost function $E(w)$ and AUC during the iterations for MovieLens. The cost keeps decreasing as the weights are updated, resulting in increased AUCs for both training and test sets.

dataset. For consistency, we randomly choose one seed node for each dataset and use it for all the subgraphs, and fix the number of weight updates (T_2 in Lemma 5) to 40. A laptop with 2.2GHz Intel Core i7 processors is used to measure the running time. Figure 3.7 shows the result; running time of SBP scales linearly with regard to the number of edges for all the datasets. SBP takes the longest in MovieLens since it is 15× more dense than the other networks; it takes more iterations for the messages to converge (T_1 in Lemma 5).

3.5 Related Works

In this section, we review related works which are categorized into three parts: Loopy Belief Propagation (LBP), real-world applications of LBP, and node classification methods based on Random Walk with Restart (RWR).

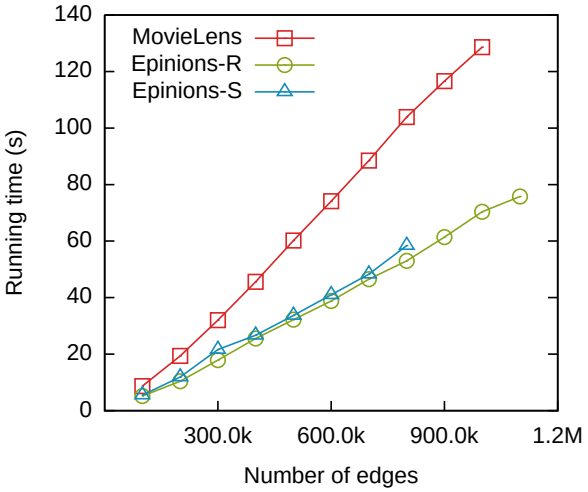


Figure 3.7: Running time of SBP for the generated subgraphs. SBP shows linear scalability with respect to the numbers of edges for all three datasets we use in the experiments.

Loopy belief propagation Yedidia et al. showed that LBP can be applied to various probabilistic graphical models without loosing generality [5]. Gonzalez et al. and Elidan et al. increased speed of the convergence in LBP by updating the messages in an asynchronous way [27, 28]. Chechetka et al. concentrated the computation of messages on more important areas of a real-world graph for faster convergence [29]. Kang et al. introduced a distributed LBP algorithm on MapReduce [30, 19]. Koutra et al. showed that three guilt-by-association methods including LBP, RWR, and Semi-supervised learning lead to a similar matrix inversion problem [25].

Real-world applications of LBP Pandit et al. and Chau et al. applied LBP in detecting fraudulent users in online auction networks [7, 10]. Akoglu et al. proposed an algorithm that extends LBP to a signed network in order to spot

fraudsters and fraudulent reviews from online review networks [31]. Jang et al. and Akoglu also applied LBP in signed networks with different propagation strengths [18, 32]. Chau et al. and Tamersoy et al. applied LBP to large scale malware detection problems [8, 16]. McGLohon et al. applied LBP to graph labeling and risk detection problems [9]. Ayday et al. and Ha et al. applied LBP to recommendation problems [15, 33]. Felzenszwalb et al. and Yang et al. applied max-product LBP to computer vision problems [17, 34]. Koutra et al. used a variant of LBP to compute the node affinities of two graphs required to measure the graph similarity [35].

RWR-based methods Haveliwala proposed RWR, an algorithm to measure relevance between nodes based on random walks [24]. Backstrom and Leskovec proposed a supervised algorithm based on RWR which determines transition probabilities as a function of node and edge features [12]. Shahriari and Jalili proposed M-RWR to solve sign prediction problems [26]. Jung et al. proposed an RWR-based algorithm on signed social networks [36].

3.6 Summary

We propose SBP (Supervised Belief Propagation), a novel and scalable graph inference algorithm for general attributed networks. SBP assigns different edge potentials based on the attributes of each edge by learning the optimal weights for the attributes. As a consequence, SBP generalizes existing methods with higher accuracies on various real-world networks. Experimental results show that SBP brings up to 15.6% higher AUC on node classification problem compared to the best existing methods. SBP enjoys linear scalability with the num-

ber of edges for both time and space complexities. Future research directions include extending SBP for distributed computing environments and learning the node potentials as well.

Chapter 4

Node Classification with Inductive Learning

Given graph-structured data, how can we train a robust classifier in a semi-supervised setting that performs well without neighborhood information? We propose BPN (Belief Propagation Network), a novel approach to train a deep neural network in a hard inductive setting, where the test data are given without neighborhood information. BPN uses a differentiable classifier to compute the prior distributions of nodes, and then diffuses the priors through the graphical structure, independently from the prior computation. This separable structure improves the generalization performance of BPN for isolated test instances, compared with previous approaches that jointly use the feature and neighborhood without distinction. As a result, BPN outperforms state-of-the-art methods in four datasets with an average margin of 2.4% points in accuracy.

4.1 Motivation

Given graph-structured data, how can we train a robust classifier in a semi-supervised setting that performs well without neighborhood information? Semi-supervised learning aims to utilize unlabeled data to improve the performance of a model. When the data are explicitly structured a graph, a typical approach to address the problem is to correlate labeled and unlabeled nodes by the graphical

structure [37, 38, 39].

Especially, many recent works have studied *inductive* approaches to develop models that perform generally well with unseen data [40]. Compared with *transductive* approaches that use the feature and neighborhood information of test data at the training time to improve the performance, such models do not use any information of the test data at the training time. As a result, they become more robust to unseen data, and their performances at the training time are easily reproduced when evaluated by unseen data.

However, recent works for inductive learning assume that every instance has enough neighborhood when input to the models, which is not true in many cases. For instance, when classifying new users in a social network who have few relationships with the others, the performance of those models is severely damaged, while being sensitive to further changes of relationships that such users generate. At the same time, it is difficult to interpret the trained relationship between features and labels since the models depend both on the feature vector and neighborhood of an instance: it is not clear which of the information mainly contributes to the performance.

In this chapter, we solve the *hard inductive* graph-based classification [37], which is to learn a classifier that works well with *isolated* test instances having no neighborhood information. This is essentially different from the *soft inductive* learning above, because it is not allowed to directly use the neighborhood information in prediction: it can help the training, but should be excluded from the final classifier. This prevents recent inductive approaches such as graph attention networks [39] from working well, since they use the neighborhood as essential evidence.

We propose BPN (Belief Propagation Network), our novel approach for training a deep neural network in a hard inductive setting. Unlike previous approaches that use features and neighborhood information together, BPN maximizes its generalization performance by separating its steps for using features and neighborhood information. First, the classification step uses an independent classifier to compute the prior distributions of nodes: it works with feature vectors and requires no neighborhood. Then, the diffusion step propagates the priors based on the *message propagation* [6], which is used for graphical inference, and computes approximate *beliefs* of nodes based on the given graph.

BPN uses two kinds of loss functions on the computed beliefs. The first is a classification loss between the beliefs and observed labels, which makes the classifier work well in labeled nodes. The second is an induction loss that minimizes the difference between the priors and beliefs, which makes the classifier robust to the non-existence of neighborhood.

BPN can be coupled with any classifier because of its separable structure. For instance, a convolutional neural network (CNN) is preferred when the features represent images, or the logistic regression is useful when the relationship between features and labels should be interpreted. This makes BPN a general structure which can be used with various types of data and different objectives. BPN can also take a classifier that has been already trained and fine-tune its parameters, instead of training a new classifier from the beginning.

We conduct extensive experiments to demonstrate the performance of BPN, which shows the highest classification accuracy on four datasets compared with the state-of-the-art approaches for inductive learning. Due to its efficient structure, training BPN is up to 150× faster than training the baselines. We also

show that BPN makes us interpret the relationship between features and labels, which is difficult with other approaches for soft inductive learning.

4.2 Problem Definition and Related Works

We define the problem of hard inductive learning and review previous works of BPN for graph-based learning.

Problem definition We are given an undirected graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of nodes and edges, respectively. Each node i represents a classification instance with a feature vector $\mathbf{x}_i$ and a label y_i. All feature vectors are accessible in training, but the labels have been observed for a small subset $\mathcal{V}_o \subset \mathcal{V}$ of the nodes. Then, we have a test set $\mathcal{D}$ that consists of only feature vectors, without the graphical structure. The problem of *hard inductive semi-supervised classification* is to train a classifier f that predicts the label of each instance $\mathbf{x} \in \mathcal{D}$.

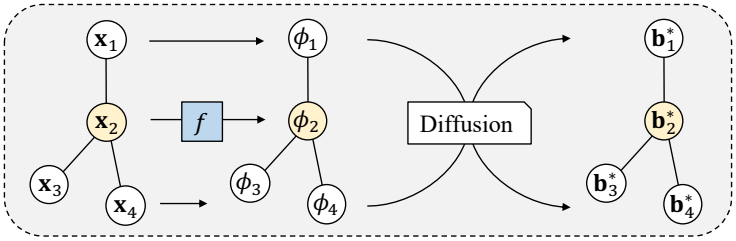
Transductive learning Transductive learning assumes that the test data are included in the graph G. Thus, a model uses their features and neighborhood information at training time and does not generalize to unseen data. Many recent approaches have been proposed for transductive learning [41, 38, 42, 43, 44, 45, 46], but they are not applicable to a hard inductive setting where isolated and unseen test data are given. Previous approaches for node embedding [47, 48] are also transductive since they generate low-dimensional representations based on the whole structure of a graph.

Inductive learning On the other hand, inductive learning does not use any information of test data at training [49, 50]. Planetoid [37] is an embedding-based approach that uses the feature vectors and labels to generate representations optimized for semi-supervised learning. In its inductive variant, the embedding is a parametric function of a feature vector and requires no neighborhood information in its prediction. To the best of our knowledge, Planetoid is the only method for hard inductive learning among recent approaches, which generalizes well to isolated test instances preserving a good performance.

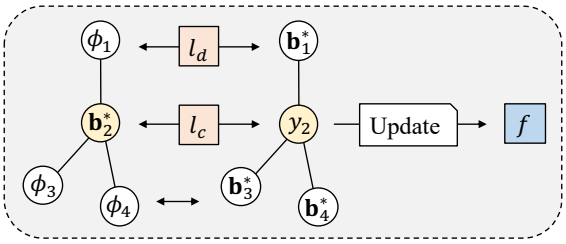
Other recent works focus on soft inductive learning where the test instances are given with neighborhood. GraphSAGE [40] generates embeddings by sampling and aggregating features from a local neighborhood. SEANO [51] is also an embedding approach but specialized for semi-supervised classification. GAT [39] is a neural network architecture that leverages self-attentions to address the shortcomings of graph convolutions. Compared with these approaches that jointly use features and neighborhood without distinction, BPN uses a fully-separable structure for using these information, improving its generalization performance for hard inductive learning.

Loopy belief propagation Loopy Belief Propagation (LBP) is an approximate inference algorithm that computes the posterior distributions of random variables in a graphical model [6]. Due to its generality and scalability, previous works used LBP for node classification such as large-scale malware detection [52, 16], social network analysis [18, 32], and recommendation [15].

A recent approach [1] has modeled LBP as a numerical function and learned the potentials by direct backpropagation from the calculated posteriors. Our



(a) Forward propagation of BPN.



(b) Backward propagation of BPN.

Figure 4.1: Overview of the BPN structure. A graph of four nodes is given as an input and node 2 is the only labeled node. (a) In the forward propagation, BPN uses a classifier f to predict the priors of nodes and diffuses them to compute the beliefs. (b) In the backward propagation, BPN uses two kinds of loss functions l_c and l_d to use all unlabeled and labeled nodes to train a robust classifier for a hard inductive setting.

work is inspired by their approach, and we use LBP to correlate nodes of a graph using an independent classifier for computing the node priors. This is effective for semi-supervised learning in a large graph, where it is difficult to estimate the potentials by a probabilistic way as in [53].

4.3 Proposed Method: BPN

We propose BPN (Belief Propagation Network), a novel approach for hard inductive semi-supervised learning. Figure 4.1 depicts the overall structure of BPN. In the forward propagation, BPN first computes the prior ϕ_i of every node

i using a classifier f. Then, it diffuses the priors following the graphical structure to compute the beliefs $\mathbf{b}^*$; this step is based on the message propagation of loopy belief propagation.

In the backward propagation of Figure 4.1, we use two types of loss functions to learn the classifier: the classification loss l_c and induction loss l_d. The classification loss l_c minimizes the classification error of $\mathbf{b}_2^*$ for node 2, whose label has been observed as y_2. At the same time, the induction loss l_d treats the beliefs as soft labels and uses the priors as predictions for the other unobserved nodes. This improves the generalization performance of f for isolated test instances, fully exploiting the graph structure.

4.3.1 Forward Propagation

We describe the forward propagation of BPN in detail, which consists of two steps: prior computation and diffusion.

Prior computation Each node i in the given graph G represents a target instance with a feature vector $\mathbf{x}_i$. BPN uses an independent classifier f to estimate the prior ϕ_i of every node i:

$$\phi_i = f(\mathbf{x}_i, \theta), \tag{4.1}$$

where θ is a set of learnable parameters in f. This prediction is done independently from the other nodes in G, ignoring the relationships between nodes. Our main objective is to train f with a small number of observed labels so that it works well with isolated test instances in a hard inductive setting.

Probabilistic modeling A *pairwise Markov network* [6] is a graphical model that represents random variables as nodes and their pairwise relationships as edges. Since the model is useful in describing properties of a real-world network with only a few hyperparameters, many previous works [52, 18] used it to model given graphs for node classification.

We model the given graph G as a pairwise Markov network and use an *edge potential* matrix ψ of size $|\mathcal{S}| \times |\mathcal{S}|$:

$$\psi = \exp(\mathrm{diag}(\epsilon)), \tag{4.2}$$

where $\mathcal{S}$ is the set of target states, $\mathrm{diag}(\epsilon)$ is a diagonal matrix whose elements are ϵ, and exp is applied elementwise to the matrix. This represents that adjacent nodes connected by an edge have the same label with a probability p proportional to $\exp(\epsilon)$. For instance, if $|\mathcal{S}| = 3$ and $\epsilon = 0.1$, the probability distribution of a node' state is given as $(0.32, 0.36, 0.32)$ if its neighbor has been observed as having the second label.

We call ϵ *propagation strength* because a property of the model is determined by the value of ϵ:

- If $\epsilon > 0$, the model induces connected nodes to have the same label because its probability is larger than the one of having different labels.
- If $\epsilon = 0$, the nodes are considered uncorrelated and thus no information is propagated through the diffusions.
- If $\epsilon < 0$, the nodes are negatively correlated: a node is induced to have a different label from those of its neighbors.

The case of $\epsilon > 0$ is the most common because most real-work networks

satisfy the *homophily* property, which states that nearby nodes are likely to have the same label. We correlate unlabeled and labeled nodes based on the assumption of $\epsilon > 0$, and choose its value as a hyperparameter.

Message propagation BPN diffuses the initial priors η times following the edges of G, maintaining two types of variables: messages and beliefs. After the t-th diffusion, a message $\mathbf{m}_{ij}^t$ from node i to node j represents local prediction of node j estimated by node i. A belief $\mathbf{b}_j^t$ is computed by aggregating the messages to node j and represents the current prediction of node j. BPN returns the beliefs $\{\mathbf{b}_j^\eta\}_{j\in\mathcal{V}}$ as a result of forward propagation.

We initialize all messages and beliefs before the diffusion starts, after computing all priors:

$$\mathbf{m}_{ij}^0 = \mathbf{1}/|\mathcal{S}|, \qquad \mathbf{b}_j^0 = \phi_j = f(\mathbf{x}_j, \theta), \tag{4.3}$$

where $\mathbf{1}$ represents a vector whose elements are all 1's. The messages are initialized as uniform distributions because the nodes have not exchanged any information at this stage. The beliefs are initialized with the predicted priors from f.

Then, the t-th messages and beliefs are computed as

$$\mathbf{m}_{ij}^t = \left[\psi(\mathbf{b}_i^{t-1} \oslash \mathbf{m}_{ji}^{t-1})\right], \tag{4.4}$$

$$\mathbf{b}_j^t = \mathrm{softmax}(\log \phi_j + \textstyle\sum_{i\in\mathcal{N}_j} \log \mathbf{m}_{ij}^t), \tag{4.5}$$

where ψ is the potential matrix, $[\cdot]$ represents a normalization function that divides a vector by the sum of its elements, $\oslash$ represents the elementwise division

between two vectors, and $\mathcal{N}_j$ represents the set of direct neighbors of node j.

The message $\mathbf{m}_{ij}^t$ propagates the belief $\mathbf{b}_i^{t-1}$ of node i to its neighbor j. Large propagation strength ϵ increases its skewness, while $\epsilon = 0$ gives a uniform distribution as a message. The belief $\mathbf{b}_j^t$ in Equation (4.5) aggregates the local information around node j by adding the logarithms of its prior ϕ_j and all incoming messages. As the aggregation is done at every diffusion step, the belief $\mathbf{b}_j^t$ represents a prediction for node j derived from its t-hop neighbors.

One important difference between BPN and previous approaches is that BPN aggregates a neighborhood's predictions with no learnable parameters. The result of diffusion is determined solely by the prediction of f. This is a main advantage since we focus on hard inductive learning, where the parameters relying on the neighborhood cannot be used at test time; BPN learns the parameters only in the classifier f.

4.3.2 Parameter Estimation

We learn the parameters θ of the classifier f by the backpropagation from the final beliefs. We propose two types of loss functions, classification loss l_c and induction loss l_d, for different objectives and combine them as a single loss function l which we aim to minimize.

Classification loss The classification loss l_c measures how well the observed labels are predicted by the beliefs:

$$l_c(\theta) = -\sum_{i \in \mathcal{V}_o} \log b_i^*(y_i), \tag{4.6}$$

where $\mathcal{V}_o$ is the set of nodes whose labels have been observed, y_i is the observed label of node i, and $b_i^*(y_i)$ is the predicted belief of node i for y_i.

This loss function involves only a few nodes in $\mathcal{V}_o$ whose labels have been observed. If we use the priors, instead of the beliefs, as predicted probabilities, l_c becomes a typical classification loss by f. Instead, we consider the η-hop neighborhoods of $\mathcal{V}_o$ by using the beliefs as our predictions. Most of the nodes in G participate in l_c if we use large η.

However, using l_c alone has two limitations. First, the performance of BPN becomes too dependent on the value of η. If η is too small, many nodes remain untouched and are not used in training f. If η is too large, the training may be unstable because of a long computational chain that causes problems in gradient computations; the training will become slow, decreasing the efficiency and scalability of the algorithm. Second, BPN learns the parameters θ of f expecting that the priors are propagated through the graphical structure. In other words, BPN does not consider f as an independent classifier, but as a component of its structure which computes the initial beliefs. As a result, the performance of f does not generalize to the test data which have no neighborhood information and thus the diffusion has no effect. We describe how to address these limitations in the next section.

Induction loss We propose the induction loss l_d to complement the limitations of the classification loss l_c:

$$l_d(\theta) = -\sum_{i \notin \mathcal{V}_o} \sum_{s \in \mathcal{S}} b_i^*(s)(\log \phi_i(s) - \log b_i^*(s)), \tag{4.7}$$

where $\mathcal{S}$ is the set of labels, and $\phi_i(s)$ is the prior of node i for a label s. This is the KL divergence loss between the beliefs and priors for all nodes whose labels are not given.

Minimizing l_d makes the classifier f produce the priors of unobserved nodes that are as closest as possible to their beliefs. In other words, we use the priors as predictions and the beliefs as *soft* labels to induce f to mimic the diffusion.

The value of l_d is close to zero at the first few epochs because the beliefs are distributed randomly. Then, the performance of f improves by minimizing l_c, and the predicted priors on $\mathcal{V}_o$ are propagated to their η-hop neighborhoods. This changes the beliefs of those nodes and increases the value of l_d. If f is learned to decrease l_d by fitting the priors to beliefs on those nodes, the new priors are propagated again to their neighborhoods, further increasing l_d on the new nodes. This is repeated until all nodes are considered by l_d.

Using l_d overcomes the limitations of l_c by the following reasons. First, BPN becomes robust to the value of η since even a small value of η is enough to correlate all nodes in G. Small η further shortens the computational chain, increasing the efficiency of gradient computation. Second, f works well on test data that have no neighborhood information, since it is trained to produce predictions that are close to the beliefs. BPN thus considers f as an independent classifier whose predictions are directly usable without further diffusions.

Overall loss function We incorporate the loss functions in Equations (4.6) and (4.7) and propose the final cost function that BPN aims to minimize:

$$l(\theta) = (1 - \beta)l_c(\theta) + \beta l_d(\theta) + \lambda||\theta||_2^2, \tag{4.8}$$

Algorithm 5: BPN (Belief Propagation Network)

Input: graph $G = (\mathcal{V}, \mathcal{E})$, feature vectors $\mathbf{x}_i$ for all $i \in \mathcal{V}$, and labels y_i for all $i \in \mathcal{V}_o$, where $\mathcal{V}_o \subset \mathcal{V}$

Input: classifier $f : \mathbb{R}^{|\mathbf{x}_i|} \to \mathbb{R}^{|\mathcal{S}|}$ with parameters θ

Output: the trained classifier f for hard inductive learning

1: **for** $[1, \text{num\_epochs}]$ **do**
2: $\quad \mathbf{b}_j^0 \leftarrow \phi_j \leftarrow f(\mathbf{x}_j, \theta)$ for all $j \in \mathcal{V}$
3: $\quad \mathbf{m}_{ij}^0 \leftarrow (1/|\mathcal{S}|)\mathbf{1}$ for all $(i, j) \in \mathcal{E}$
4: $\quad$ **for** t in $[1, \eta]$ **do**
5: $\quad\quad \mathbf{m}_{ij}^t \leftarrow [\psi(\mathbf{b}_i^{t-1} \oslash \mathbf{m}_{ji}^{t-1})]$ for all $(i, j) \in \mathcal{E}$
6: $\quad\quad \mathbf{b}_j^t \leftarrow \text{softmax}(\log \phi_j + \sum_{i \in \mathcal{N}_j} \log \mathbf{m}_{ij}^t)$ for all j
7: $\quad$ **end for**
8: $\quad l_c \leftarrow -\sum_{i \in \mathcal{V}_o} \log b_i^\eta(y_i)$
9: $\quad l_d \leftarrow -\sum_{i \notin \mathcal{V}_o} \sum_{s \in \mathcal{S}} b_i^\eta(s)(\log \phi_i(s) - \log b_i^\eta(s))$
10: $\quad l \leftarrow (1 - \beta) l_c + \beta l_d + \lambda ||\theta||_2^2$,
11: $\quad \theta \leftarrow$ update the parameters to minimize l
12: **end for**

where β adjusts the balance between the two loss functions, and λ is an L2 regularization parameter for θ. Note that BPN has no learnable parameters in its diffusion part; the target of training is the parameters θ of the classifier f. If $\beta = 0$, the induction loss is not used and f may not generalize well to the unseen test examples. If $\beta = 1$, the classification loss, which guides the induction loss in the first epochs, is not used. We thus set β in $(0, 1)$.

4.3.3 Detailed Algorithm

We summarize BPN in Algorithm 5, which takes a classifier f as an input and trains it in a semi-supervised setting. BPN uses f to compute the priors of nodes in line 2. Then, in lines 3 to 7, BPN diffuses the priors η times to compute the final beliefs. In lines 8 to 11, BPN computes the loss l and updates the parameters θ of f. This is repeated until f is trained properly, and the

resulting f is evaluated by test data that are not observed at training time and have no neighborhood.

Implementation Most computations in BPN are carried out efficiently in a recent deep learning framework. However, the computation of a message $\mathbf{m}_{ij}^t$ in Equation (4.4) involves a random access to the reverse direction, which is difficult for most sparse representations of adjacency matrices. The coordinate (COO) and dictionary of keys (DOK) formats support constant-time access to a random entry, but lose the spatial locality of edges, which is crucial when computing the beliefs in Equation (4.5). The compressed sparse row (CSR) and the compressed sparse column (CSC) formats [54] preserve the locality, but do not support constant access to the reverse edges.

Thus, based on the CSR format, we use an additional index array R that stores the positions of reverse edges to support constant access. As a result, the graph is represented as four arrays M, I, D, and R. Each undirected edge is stored by two directed edges in M of length $2|\mathcal{E}|$. $I[j]$ contains the position of the first edge in M that is incident to node j, and $D[j]$ has the degree of node j. For each edge in M, we set $R[j]$ to store the position of the reverse edge of $M[j]$ in M. Our implementation addresses both requirements:

- **Locality.** If we want to compute the belief $\mathbf{b}_j$, all messages that go to node j are located in between $M[I[j]]$ and $M[I[j] + D[j]]$ continuously.
- **Reverse indexing.** If we want to find the reverse of the message $\mathbf{m}_{ij}$ in $M[k]$, it is located in $M[R[k]]$.

Table 4.1: Summary of datasets.

Name	Nodes	Edges	Attributes	Labels
Pubmed[1]	19,717	44,324	500	3
Cora[1]	2,708	5,278	1,433	7
Citeseer[1]	3,327	4,552	3,703	6
Amazon	32,966	63,285	3,000	3

[1] `https://github.com/kimiyoung/planetoid`

4.4 Experiments

Our experiments show that BPN outperforms the state-of-the-art methods for hard inductive learning.

4.4.1 Experimental Settings

We introduce our experimental settings including datasets, an experimental setup, and baseline methods. Our experiments are done in a workstation with Geforce GTX 1080 Ti.

Datasets We use four datasets summarized in Table 7.2. The first three datasets [55] were used to evaluate the previous approaches [39]. The nodes represent scientific publications classified by the research areas and have feature vectors about their textual contents: TF-IDF weighted vectors in Pubmed, and bag-of-words vectors in Citeseer and Cora. The edges represent citations between the articles.

We also use an Amazon dataset based on [56, 57]. The original dataset contains items of Amazon, each of which contains a text description, a category, and a list of related items. We use the description of each item as a bag-of-words

Table 4.2: Classification accuracy of BPN and the baseline methods. We report the average and standard deviation of ten runs with different random seeds. BPN consistently shows the highest accuracy with low deviations.

Method	**Pubmed**	**Cora**	**Citeseer**	**Amazon**
Planetoid	74.6 ± 0.5	66.2 ± 0.9	66.8 ± 1.0	70.1 ± 1.9
GCN-I	74.1 ± 0.2	67.8 ± 0.6	63.6 ± 0.5	76.5 ± 0.3
SEANO	75.7 ± 0.4	64.5 ± 1.2	66.3 ± 0.8	78.6 ± 0.6
GAT	76.5 ± 0.4	70.1 ± 1.0	66.7 ± 1.0	77.5 ± 0.4
BPN (ours)	**78.3 ± 0.3**	**72.2 ± 0.5**	**70.1 ± 0.9**	**81.5 ± 1.3**

feature vector after reducing the number of words [58]. Then, we create a network of the items by connecting related ones as edges and classify the category of each item into *Electronics*, *Cell Phones and Accessories*, or *Automotive*.

Experimental setup For each dataset, we use 20 nodes of each class for training, 1,000 nodes for testing, and 500 nodes for validation as done in [38]. Since we aim to solve the hard inductive problem, we remove the neighborhood information of the test nodes to treat them as independent instances. We run every method ten times and report the average and standard deviation of classification accuracy.

We use a feedforward neural network with one hidden layer as a classifier f. Since it has a shallow structure, we use tanh as the activation function and do not use a bias. The number of hidden units is set to 32, and we use dropout [59] of probability 0.5. Adam [60] is used as an optimizer for all datasets with different step sizes determined by validation performances.

Given the classifier f, we choose the hyperparameters of BPN based on the validation performance on Cora and use a similar setting in Citeseer: $\epsilon = 0.05$,

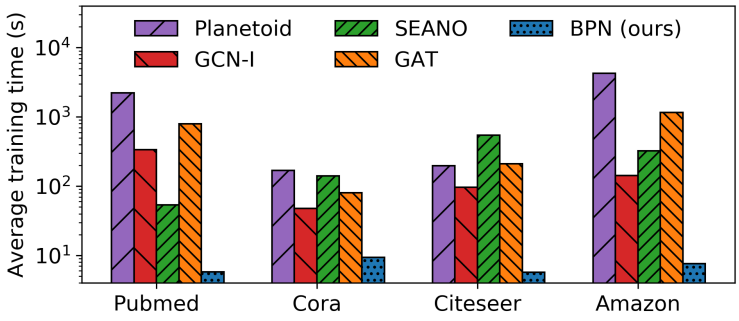


Figure 4.2: Average training time of BPN and the baselines for ten runs. BPN is up to 150× faster than the recent methods.

$\lambda = 10^{-4}$, and $\beta = 0.9$ in Cora, and ϵ is changed to 0.01 in Citeseer. Since Pubmed and Amazon have much less labels than in Cora and Citeseer, we change the parameters to more efficient training: $\epsilon = 1.0$ and $\beta = 0.5$. We lastly set $\lambda = 2 \cdot 10^{-2}$ in PubMed based on its small number of features. The number η of diffusion operations is set to one in all datasets, which is small but enough to correlate all nodes by the induction loss l_d.

Baselines We compare BPN with four competitive baselines recently proposed: Planetoid, GCN, GAT, and SEANO. The last two methods are considered as the state-of-the-art methods for inductive learning. We do not include GraphSAGE although it is also an inductive approach, because it focuses on either supervised or unsupervised learning: its performance on semi-supervised learning has been worse than those from the other baselines [51]. We include GCN-I, an inductive variant of GCN, by removing all edges connected with the test instances. We get public implementations of the baseline methods and measure the accuracy and training time.

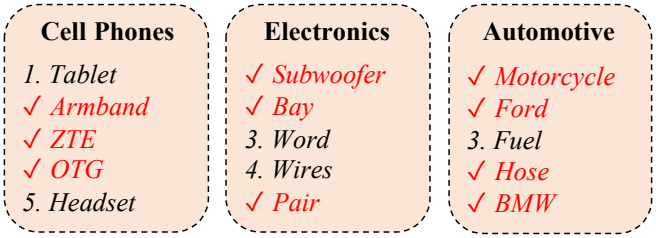


Figure 4.3: The most important keywords derived from the learned classifier of BPN for each target class of the Amazon dataset. Although the checked words do not appear in the labeled nodes, they are captured successfully by BPN.

4.4.2 Classification Performance

We demonstrate that BPN outperforms the baselines by both classification accuracy and training time. Table 4.2 shows that BPN produces the highest accuracy with an average margin of 2.4% points, which is a significant amount considering the low standard deviations. Moreover, Figure 4.2 shows that BPN is trained up to 150× faster than the recent baselines, which consider multi-hop neighbors of each instance with extensive amounts of computations. These are due to the efficient structure of BPN that considers only a small number of neighbors at each epoch but eventually correlates all labeled and unlabeled nodes by minimizing two kinds of loss functions.

4.4.3 Interpretability

Since the prediction of the previous approaches relies both on the feature and neighborhood information of an instance, it is difficult to interpret the resulting models. On the other hand, the learned classifier f from BPN uses only the features, and it is possible to analyze the relationship between the features and labels. One way to interpret f is to give a zero input and differentiate the

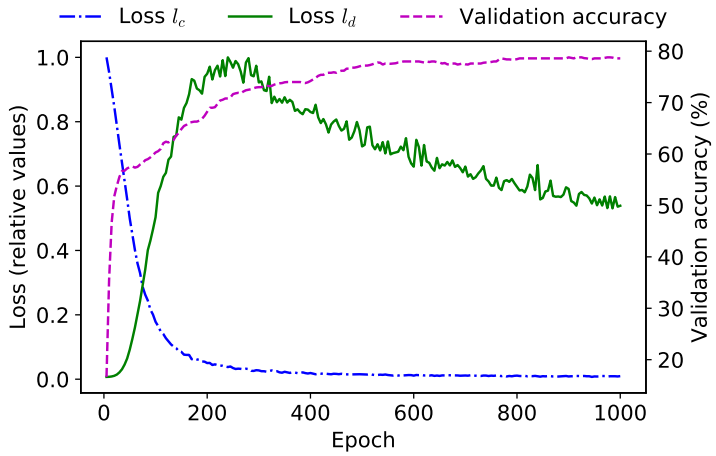
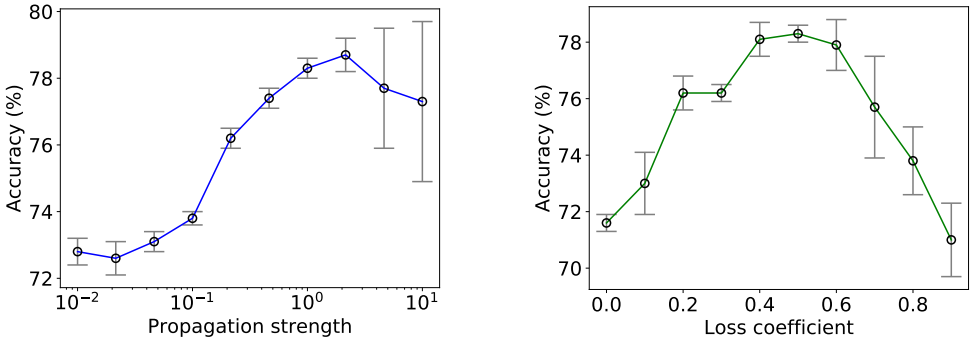


Figure 4.4: Loss values and validation accuracy during the training of BPN for the Cora dataset. BPN improves the validation accuracy by first minimizing the classification loss l_c and then minimizing the induction loss l_d.

prediction with regard to the input. Then, the gradient of each element represents its importance for classification. Figure 4.3 shows the top five keywords of the largest importances for each class on Amazon, each of which corresponds to an element of the bag-of-words vectors. Although the checked words are not present in the labeled nodes, we note that they are learned successfully by BPN and used to predict the test instances by f as important evidence.

4.4.4 Effects of the Loss Terms

BPN minimizes two kinds of loss functions to perform well in a hard inductive setting. Figure 4.4 shows their values during the training of BPN on Cora. At first, the classification loss l_c is at maximum, while the induction loss l_d is the smallest since most priors and beliefs are close to uniform distributions and thus similar to each other. BPN then minimizes l_c on the labeled nodes, changing the beliefs of their neighborhoods by diffusion; this increases l_d instead. After that,



(a) Propagation strength (b) Loss coefficient

Figure 4.5: Classification accuracy of BPN on Pubmed with two hyperparameters: propagation strength ϵ and the loss coefficient β. Optimal values exist in the middle in both plots, showing small standard deviations.

BPN moves to the unlabeled nodes and updates its parameters to produce priors that are as closest as possible to their beliefs, minimizing l_d in latter epochs. The validation accuracy keeps increasing whether we focus on l_c or l_d.

4.4.5 Parameter Sensitivity

We conduct various experiments on BPN to see the effect of each hyperparameter on its performance. The results for two hyperparameters ϵ and β are summarized in Figure 4.5. Larger ϵ leads to higher accuracy until a certain point, and then the deviation sharply increases, decreasing the average accuracy. This is because large ϵ propagates more information at each diffusion, decreasing the stability. In terms of the loss coefficient, $\beta = 0.5$ gives the best balance between the two losses. It is notable that larger deviations are observed when $\beta > 0.5$, since in that case l_d starts to be minimized before f is fully optimized for the labeled nodes by minimizing l_c.

4.5 Summary

In this chapter, we have proposed BPN (Belief Propagation Network), a novel approach for hard inductive semi-supervised learning. BPN takes a differentiable classifier as an input and trains it efficiently by minimizing two types of loss functions, improving both its classification performance and robustness to the non-existence of neighborhood. As a result, the trained classifier outperforms the state-of-the-art approaches in four datasets for text classification. Moreover, the separable structure of BPN makes the classifier interpretable in terms of the relationship between features and labels, which is difficult in previous approaches for soft inductive learning. Future works include designing an improved message passing operation for heterogeneous or edge-attributed networks.

Chapter 5

Node Feature Estimation

Given a graph with partial observations of node features, how can we estimate the missing features accurately? Missing feature estimation is a crucial problem for analyzing real-world graphs whose features are commonly missing during the data collection process. An accurate estimation not only provides diverse information of nodes but also supports the inference of graph neural networks that require the full observation of node features. In this chapter, we propose MGA (Markov Graph Autoencoder), a method for accurate feature estimation. MGA generates high-dimensional features with variational inference that changes the problem of maximizing the intractable likelihood into maximizing the evidence lower bound. MGA then models the prior distribution of latent variables with Gaussian Markov random field, making the KL divergence term of variational inference into a strong graph-based regularizer that considers the correlations between variables. Extensive experiments show that MGA makes state-of-the-art performance in feature estimation and node classification problems.

5.1 Motivation

Given a graph with partial observations of node features, how can we estimate the missing features accurately? Many real-world data are represented as graphs to model the relationships between entities. Social networks, seller-item graphs in electronic commerce, and user-movie graphs in a streaming service are all

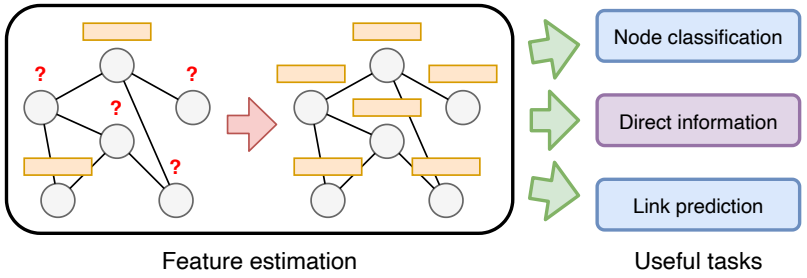


Figure 5.1: An illustration of the feature estimation problem. The generated features not only provide the missing information of node properties but also help graph-related tasks such as node classification and link prediction.

examples of graph data that have been studied widely in literature [1, 38, 39, 61, 62]. Such graphs become more powerful when combined with feature vectors that describe the diverse properties of nodes [63, 64].

However, features are commonly missing in real-world graphs, especially in large data. Users in an online social network set their profiles private, and sellers in electronic commerce often register items without an informative description. In such cases, even the observed features cannot be used properly due to the missing ones, since many graph algorithms assume the full observation of node features. Figure 7.1 illustrates the feature estimation problem in an example graph. An accurate estimation of missing features not only provides diverse information of node properties but also improves the performance of other essential tasks such as node classification since the generated features provide important evidence.

However, making an accurate estimation of missing features is challenging due to the following reasons. First, target nodes have no specific information that describes their properties. The main evidence for estimation is the graph structure, which provides only partial information of nodes based on

the relationships with the other nodes. Second, the target variables are high-dimensional vectors containing up to thousands of elements. This requires large representation power for accurate estimation, which involves a high risk of overfitting. Previous works [65, 66, 67] succeeded in increasing their representation power for predicting high-dimensional features but lack strong regularizers that effectively control the training process to avoid the overfitting problem.

We propose MGA (Markov Graph Autoencoder), a method for accurate estimation of missing features. MGA consists of encoder and decoder networks, where an encoder makes a latent variable for each node, and a decoder generates high-dimensional features from the latent variables. Our main idea is to model the target variables in a probabilistic way with variational inference, which allows us to maximize the intractable likelihood. We also model the prior distribution of variables as Gaussian Markov random field (GMRF), introducing a regularizer that forces the latent variables to have correlations with each other following the graph structure. As a result, MGA effectively addresses the challenges of feature estimation, making state-of-the-art performance.

Our contributions are summarized as follows:

- **Method.** We propose MGA, a method for accurate estimation of missing features. MGA provides a new way to run variational inference on graph-structured data by assuming the prior of latent variables as GMRF.
- **Theory.** We analyze the time and space complexities of our MGA, which are both linear with the number of nodes and edges of the given graph, showing its scalability.
- **Experiments.** Extensive experiments on real-world graph datasets show that MGA makes state-of-the-art performance with up to 16.3% higher

recall and 14.0% higher nDCG scores in feature estimation, and up to 7.0% higher accuracy in node classification than the best competitors.

The rest of this chapter is organized as follows. In Section 5.2, we introduce preliminaries. In Section 5.3, we propose our MGA and discuss its theoretical properties. We show experimental results in Section 5.4 and introduce related works in Section 7.5. We summarize in Section 7.6.

5.2 Preliminaries

We present the formal definition of the missing feature estimation problem and introduce Gaussian Markov random fields.

5.2.1 Missing Feature Estimation

The feature estimation problem is defined as follows. We have an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of nodes and edges, respectively. A feature vector $\mathbf{x}_i$ exists for every node i, but is observable only for a subset $\mathcal{V}_x \subset \mathcal{V}$ of nodes. Our goal is to predict the missing features of *test* nodes $\mathcal{V} \setminus \mathcal{V}_x$ using the structure of G and the observations for $\mathcal{V}_x$.

We also assume that the label y_i of each node i can be given as an additional input for a set $\mathcal{V}_y$ of nodes such that $\mathcal{V}_y \subseteq \mathcal{V}$. Such labels improve the accuracy of feature estimation, especially when they provide direct evidence for the test nodes: $\mathcal{V}_y \cap (\mathcal{V} \setminus \mathcal{V}_x) \neq \emptyset$. This is based on the idea that categorical labels are often easier to acquire than high-dimensional features, and knowing the labels of target nodes gives a meaningful advantage for estimation. Thus, we design our framework to be able to work with $\mathcal{V}_y \neq \emptyset$, although we consider $\mathcal{V}_y = \emptyset$ as a base setup of experiments for the consistency with previous approaches that

take only the observed features.

The problem differs from typical generative learning [68, 69, 70] in that the correct answers exist. Generative learning is typically an unsupervised problem whose goal is to make diverse samples that have similar properties with the true ones. On the other hand, our feature estimation is a supervised problem whose goal is to predict the exact feature $\mathbf{x}_i$ of each test node $i \in \mathcal{V} \setminus \mathcal{V}_x$. Such a difference comes from the existence of the graph, which allows us to identify each target node based on its neighborhood information.

Evaluation We evaluate the performance of feature estimation in two ways. First, we directly compare predictions with the true features that are unknown at the training time. Second, we solve the node classification problem utilizing the generated features to quantify how well the generated features model the relationships between nodes. This is based on the idea that generated features can be informative for solving other tasks regardless of the error from the true features. We describe the details of evaluation in Section 5.7.

5.2.2 Gaussian Markov Random Field

Gaussian Markov random field (GMRF) [6] is a graphical model that represents a multivariate Gaussian distribution. Given a graph $G = (\mathcal{V}, \mathcal{E})$ whose nodes have continuous signals that are correlated by the graph structure, GMRF represents the distribution of signals with two kinds of potential functions ψ_i and ψ_{ij} for every node i and edge (i, j), respectively. We assume the signal of each node i as a random variable Z_i with a possible value z_i.

Specifically, the node potential ψ_i for each node i and the edge potential

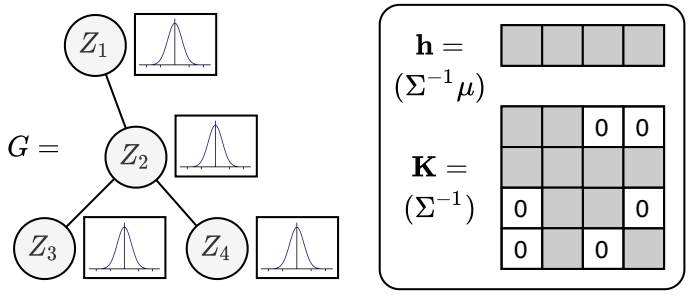


Figure 5.2: Gaussian Markov random field (GMRF) that describes a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with parameters $\mathbf{h}$ and $\mathbf{K}$. The nonzero entries in $\mathbf{K}$ correspond to the edges in G.

ψ_{ij} for each edge (i, j) are defined as follows:

$$\psi_i(z_i) = \exp(-0.5K_{ii}z_i^2 + h_i z_i) \tag{5.1}$$

$$\psi_{ij}(z_i, z_j) = \exp(-K_{ij}z_i z_j), \tag{5.2}$$

where $\mathbf{h} \in \mathbb{R}^n$ and $\mathbf{K} \in \mathbb{R}^{n \times n}$ are the parameters of the GMRF, and n is the number of nodes. The nonzero elements of $\mathbf{K}$ correspond to the edges of the graph as depicted in Figure 5.2.

Then, the joint probability $p(\mathbf{z})$ is given as the multiplication of all potential functions:

$$p(\mathbf{z}) = \frac{1}{C} \prod_{i \in \mathcal{V}} \psi_i(z_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(z_i, z_j), \tag{5.3}$$

where C is a normalization constant. Each potential measures how likely z_i or (z_i, z_j) appears with the current probabilistic assumption with the parameters $\mathbf{h}$ and $\mathbf{K}$, and the joint probability is computed by multiplying the potentials for all nodes and edges.

Representing a real-world graph as GMRF allows us to understand the re-

lationships between nodes in a probabilistic way. In other words, we can make from GMRF a multivariate Gaussian that models the correlations between variables following the graph structure. This is supported by Lemma 6.

Lemma 6. *The joint probability of Equation* (5.3) *is the same as the probability density function of a multivariate Gaussian distribution* $\mathcal{N}(\mu, \Sigma)$*, where* $\mu = \mathbf{K}^{-1}\mathbf{h}$ *and* $\Sigma = \mathbf{K}^{-1}$*.*

Proof. The probability density function of $\mathcal{N}(\mu, \Sigma)$ is

$$f(\mathbf{z}) = C' \exp(-(\mathbf{z} - \mu)^\top \Sigma^{-1}(\mathbf{z} - \mu)),$$

where $C' = (2\pi)^{-d/2}|\Sigma|^{-1/2}$ is a constant.

We rewrite f as follows with $\mathbf{K} = \Sigma^{-1}$ and $\mathbf{h} = \mathbf{K}\mu$:

$$\begin{aligned} f(\mathbf{z}) &= C' \exp(-\mathbf{z}^\top \mathbf{K}\mathbf{z} + 2\mu^\top \mathbf{K}\mathbf{z} - \mu^\top \mathbf{K}\mu) \\ &= C'' \exp(-\mathbf{z}^\top \mathbf{K}\mathbf{z} + 2\mu^\top \mathbf{K}\mathbf{z}) \\ &= C'' \exp\Big(-\sum_i \sum_j z_i K_{ij} z_j + 2\sum_i h_i z_i\Big). \end{aligned}$$

where $C'' = \exp(\mu^\top \mathbf{K}\mu) \cdot C'$ is also a constant.

By the definition of GMRF, $K_{ij} \neq 0$ only if edge (i, j) exists in the given graph $G = (\mathcal{V}, \mathcal{E})$. Then, we rewrite $f(\mathbf{z})$ as

$$f(\mathbf{z}) = C \exp\Big(\sum_{(i,j)\in\mathcal{E}} (-z_i K_{ij} z_j) + \sum_{i\in\mathcal{V}} (-0.5 K_{ii} z_i^2 + h_i z_i)\Big),$$

where $C = \exp(2) \cdot C''$. We prove the lemma by substituting $\psi_{ij}(z_{ij})$ and $\psi_i(z_i)$ for the first and second terms, respectively. □

GMRF is a powerful tool to incorporate a real-world graph in a probabilistic framework. The roles of parameters $\mathbf{K}$ and $\mathbf{h}$ can be understood with respect to the distribution that they represent. $\mathbf{K}$ is the inverse of the covariance Σ, and a pair of signals z_i and z_j is more likely to be observed if K_{ij} is small. $\mathbf{h}$ determines the mean of the signals if $\mathbf{K}$ is fixed, and is typically set to zero as we assume no initial bias of signals for the simplicity of computation.

5.3 Proposed Method: MGA

We propose MGA (Markov Graph Autoencoder), a method for an accurate estimation of missing features. We address the following challenges to achieve high accuracy of estimation:

- **Intractable optimization.** The direct maximization of the likelihood of observed variables involves intractable computation over all possible states of latent variables.
- **Non-trivial correlations.** The target variables are strongly correlated with each other in graph data, and such correlations need to be modeled properly in our framework.
- **Representation power.** An estimator should contain sufficient representation power to predict the high-dimensional feature $\mathbf{x}_i$ of each target node i from limited observations.

The main ideas of MGA are summarized as follows:

- **Variational inference (Sec. 5.3.1).** We formulate the objective function with variational inference to maximize the joint likelihood of observations in a tractable way.

- **GMRF prior (Sec. 5.3.2).** We assume the prior of latent variables as Gaussian Markov random field (GMRF), exploiting the graph structure for modeling the correlations between variables.
- **GNN with diagonal features (Sec. 5.3.3).** We learn an independent embedding for each node by using a graph neural network (GNN) with diagonal features, mapping each node into a unique representation.

Figure 5.3 shows the overall structure of MGA, which consists of an encoder f and two decoder networks g_x and g_y. The encoder f generates latent variables for all nodes in the graph, using a GMRF prior for modeling their distribution. The decoders g_x and g_y have different roles: the output of the feature decoder g_x makes the final estimation of missing features, while the label decoder g_y helps the training of g_x and is not used if no labels are observed.

5.3.1 Variational Inference for Joint Learning

Given the adjacency matrix $\mathbf{A}$ of a graph $G = (\mathcal{V}, \mathcal{E})$, our goal is to find optimal parameters $\Theta = \{\phi, \theta, \rho\}$ that maximize the likelihood $p_\Theta(\mathbf{X}, \mathbf{y} \mid \mathbf{A})$ of observed features $\mathbf{X}$ and labels $\mathbf{y}$. We introduce a latent variable $\mathbf{z}_i \in \mathbb{R}^d$ for each node i and denote the realization of all latent variables by $\mathbf{Z} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the size of variables. The latent variable $\mathbf{z}_i$ represents the characteristic of each node i for estimating its feature $\mathbf{x}_i$.

However, the direct maximization of the likelihood term $p_\Theta(\mathbf{X}, \mathbf{y} \mid \mathbf{A}) = \mathbb{E}_{\mathbf{Z} \sim p(\mathbf{Z} \mid \mathbf{A})}[p_\Theta(\mathbf{X}, \mathbf{y} \mid \mathbf{Z}, \mathbf{A})]$ involves the intractable expectation of $\mathbf{Z}$ over $p(\mathbf{Z} \mid \mathbf{A})$. Thus, we change the problem into maximizing the evidence lower bound

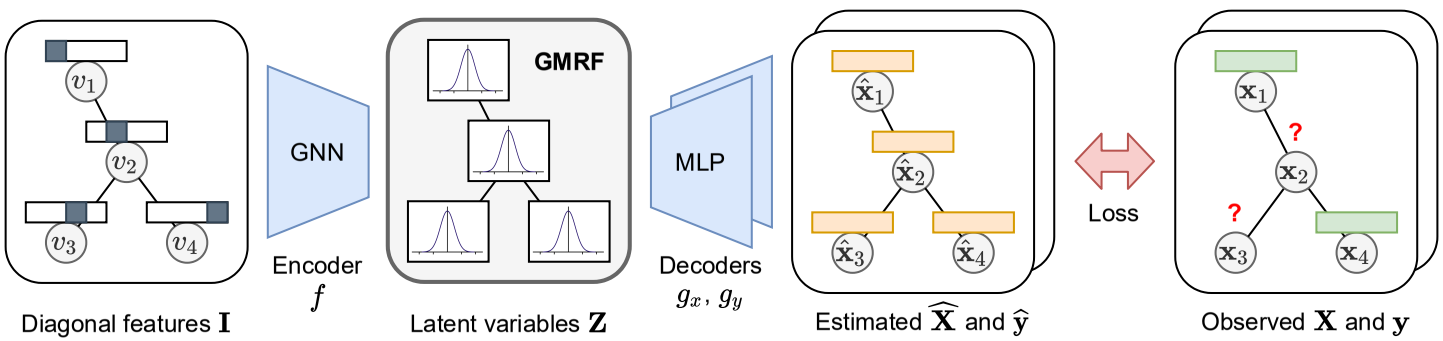


Figure 5.3: The structure of our MGA, which consists of an encoder network f and two decoder networks g_x and g_y for features and labels, respectively. We model the distribution of latent variables with GMRF, exploiting the graph structure for modeling the correlations between target variables. The label decoder g_y works as an auxiliary module that helps g_x.

(ELBO) with variational inference:

$$\begin{aligned}\log p_\Theta(\mathbf{X}, \mathbf{y} \mid \mathbf{A}) &\geq \mathcal{L}(\Theta) \\ &= \mathbb{E}_{\mathbf{Z}\sim q_\phi(\mathbf{Z}|\mathbf{X},\mathbf{y},\mathbf{A})}[\log p_{\theta,\rho}(\mathbf{X}, \mathbf{y} \mid \mathbf{Z}, \mathbf{A})] \\ &\qquad - D_{\mathrm{KL}}(q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A}) \,||\, p(\mathbf{Z} \mid \mathbf{A})),\end{aligned} \tag{5.4}$$

where $\mathcal{L}(\Theta)$ is the ELBO, q_ϕ is a parameterized distribution of $\mathbf{Z}$, and $p_{\theta,\rho}$ is a parameterized distribution of $\mathbf{X}$ and $\mathbf{y}$.

We assume the conditional independence between $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{A}$ given $\mathbf{Z}$, expecting that each variable $\mathbf{z}_i$ has sufficient information of node i to generate its feature $\mathbf{x}_i$ and label y_i. Then, the first term of $\mathcal{L}(\Theta)$ in Equation (5.4) is rewritten as follows:

$$\begin{aligned}&\mathbb{E}_{\mathbf{Z}\sim q_\phi(\mathbf{Z}|\mathbf{X},\mathbf{y},\mathbf{A})}[\log p_{\theta,\rho}(\mathbf{X}, \mathbf{y} \mid \mathbf{Z}, \mathbf{A})] \\ &\qquad = \mathbb{E}_{\mathbf{Z}\sim q_\phi(\mathbf{Z}|\cdot)}\Big[\sum_{i\in\mathcal{V}_x} \log p_\theta(\mathbf{x}_i \mid \mathbf{z}_i) + \sum_{i\in\mathcal{V}_y} \log p_\rho(y_i \mid \mathbf{z}_i)\Big],\end{aligned} \tag{5.5}$$

where $\mathcal{V}_x$ and $\mathcal{V}_y$ are the sets of nodes whose features and labels are observed, respectively, and $q_\phi(\mathbf{Z} \mid \cdot)$ denotes $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$.

Equation (5.5) represents the conditional likelihood of observed features and labels given $\mathbf{Z}$. Thus, maximizing Equation (5.5) is the same as minimizing the reconstruction error of observed variables in typical autoencoders. On the other hand, the KL divergence term in Equation (5.4) works as a regularizer that forces the distribution $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ of latent variables to be close to the prior $p(\mathbf{Z} \mid \mathbf{A})$. The characteristic of regularization depends on how we model the prior $p(\mathbf{Z} \mid \mathbf{A})$, which plays an essential role in our framework.

5.3.2 Regularization with the GMRF Prior

Previous works utilizing variational inference [68, 71] assume the prior of latent variables as a multivariate Gaussian distribution whose variables are independent of each other. Instead, we model the prior $p(\mathbf{Z} \mid \mathbf{A})$ as GMRF to consider the correlations between variables in the probabilistic modeling. We present two kinds of parameterization to model the distributions of latent variables: a basic way and a deterministic modeling that we propose.

Basic Parameterization

We model $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ as a multivariate Gaussian distribution $\mathcal{N}(\mathbf{U}, \Sigma)$, where $\mathbf{U}$ and Σ are the mean and covariance matrices of size $n \times d$ and $n \times n$, respectively. We assume that all d elements at each node share the same covariance matrix. $\mathbf{U}$ and Σ are generated from encoder functions f_μ and f_σ, respectively, which contain the set ϕ of learnable parameters.

We then model the prior $p(\mathbf{Z} \mid \mathbf{A})$ as GMRF $\mathcal{N}(\mathbf{0}, \mathbf{K}^{-1})$ with parameters $\mathbf{h} = \mathbf{0}$ and $\mathbf{K}$. We make the information matrix $\mathbf{K}$ from $\mathbf{A}$ as a graph Laplacian matrix with symmetric normalization [72]: $\mathbf{K} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}$ is the identity matrix, and $\mathbf{D}$ is the degree matrix such that $D_{ii} = \sum_j A_{ij}$. The resulting $\mathbf{K}$ preserves the structural information of the graph G as a positive-semidefinite matrix that satisfies the constraint of GMRF; the nonzero entries of $\mathbf{K}$ except the diagonal ones correspond to those of $\mathbf{A}$. Note that $\mathbf{K}$ is a constant, as it represents the fixed prior distribution.

Given the Gaussian modeling of $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ and $p(\mathbf{Z} \mid \mathbf{A})$, the KL

divergence is formulated as follows:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A}) \mid\mid p(\mathbf{Z} \mid \mathbf{A})) \\ = 0.5[\mathrm{tr}(\mathbf{U}^\top \mathbf{K} \mathbf{U}) + \mathrm{tr}(\mathbf{K}\Sigma) - \log|\Sigma|] + C, \quad (5.6)$$

where C is a constant related to $\mathbf{K}$ and $|\mathcal{V}|$.

The computational bottleneck of Equation (5.6) is $\log|\Sigma|$, whose computation is $O(n^3)$ [73]. Thus, we decompose the covariance as $\Sigma = \beta\mathbf{I} + \mathbf{V}\mathbf{V}^\top$ with a rectangular matrix $\mathbf{V} \in \mathbb{R}^{n \times r}$, where β and r are hyperparameters such that $r \ll n$ [74]. As a result, $\log|\Sigma|$ is computed efficiently by the matrix determinant lemma [75]:

$$\log|\Sigma| = \log|\mathbf{I}_r + \beta^{-1}\mathbf{V}^\top\mathbf{V}| + \log|\beta\mathbf{I}_n|, \quad (5.7)$$

where $\mathbf{I}_r$ and $\mathbf{I}_n$ are the identity matrices of sizes $r \times r$ and $n \times n$, respectively. The computation of Equation (5.7) is $O(r^2 n + r^3)$, which is more efficient than $O(n^3)$ of the full covariance matrix.

For each inference, we sample $\mathbf{Z}$ randomly from q_ϕ based on $\mathbf{U}$ and $\mathbf{V}$ generated from f_μ and f_σ, respectively. Since the gradient-based update is not possible with the direct sampling of $\mathbf{Z}$, we use the reparametrization trick of variational autoencoders [68, 74]:

$$\mathbf{Z} = \mathbf{U} + \sqrt{\beta}\mathbf{M}_1 + \mathbf{V}\mathbf{M}_2, \quad (5.8)$$

where $\mathbf{M}_1 \in \mathbb{R}^{n \times d}$ and $\mathbf{M}_2 \in \mathbb{R}^{r \times d}$ are matrices of standard normal variables, which are sampled randomly at each time to simulate the sampling of $\mathbf{Z}$ while supporting the backpropagation.

We verify that the variables $\mathbf{Z}$ sampled from Equation (5.8) follow the target distribution $\mathcal{N}(\mathbf{U}, \Sigma)$ by Lemma 7 and 8.

Lemma 7. *Let $\mathbf{z}_i$ be a latent variable sampled from Equation* (5.8) *for node i, and $\mathbf{u}_i$ be the i-th row of $\mathbf{U}$. Then, $\mathbb{E}[\mathbf{z}_i] = \mathbf{u}_i$.*

Proof. The random variables included in Equation (5.8) are $\mathbf{M}_1$ and $\mathbf{M}_2$. Since $\mathbf{M}_1$ and $\mathbf{M}_2$ are filled with standard normal values, it is satisfied that $\mathbb{E}[\sqrt{\beta}\mathbf{M}_1] = \mathbf{0}$ and $\mathbb{E}[\mathbf{V}\mathbf{M}_2] = \mathbf{0}$, regardless of the actual values of β and $\mathbf{V}$. Thus, $\mathbb{E}(\mathbf{Z}) = \mathbb{E}(\mathbf{U}) = \mathbf{U}$. □

Lemma 8. *Assume that the size d of latent variables is one. Let z_i and z_j be latent variables sampled from Equation* (5.8) *for nodes i and j, respectively. Then, $\mathbb{E}[(z_i - \mathbb{E}[z_i])(z_j - \mathbb{E}[z_j])] = \Sigma_{ij}$.*

Proof. The following is satisfied for both $k = i$ and $k = j$ based on Lemma 7:

$$z_k - \mathbb{E}[z_k] = \sqrt{\beta} m_{1k} + \mathbf{v}_k^\top \mathbf{m}_2,$$

where $\mathbf{v}_k$ and $\mathbf{m}_2$ are r-dimensional vectors.

Then, the covariance between z_i and z_j is given as

$$\begin{aligned}\mathbb{E}[(z_i - \mathbb{E}[z_i])(z_j - \mathbb{E}[z_j])] &= \beta \mathbb{E}[m_{1i} m_{1j}] \\ &+ \sqrt{\beta}\mathbf{v}_j^\top \mathbb{E}[m_{1i}\mathbf{m}_2] + \sqrt{\beta}\mathbf{v}_i^\top \mathbb{E}[m_{1j}\mathbf{m}_2] + \mathbb{E}[\mathbf{v}_i^\top \mathbf{m}_2 \mathbf{v}_j^\top \mathbf{m}_2].\end{aligned}$$

Recall that every element of $\mathbf{M}_1$ and $\mathbf{M}_2$ follows the standard normal distribution. This results in the following. First, $\mathbb{E}[m_{1i} m_{ij}] = 1$ if $i = j$ and zero otherwise. Second, the second and third elements of the right hand side are

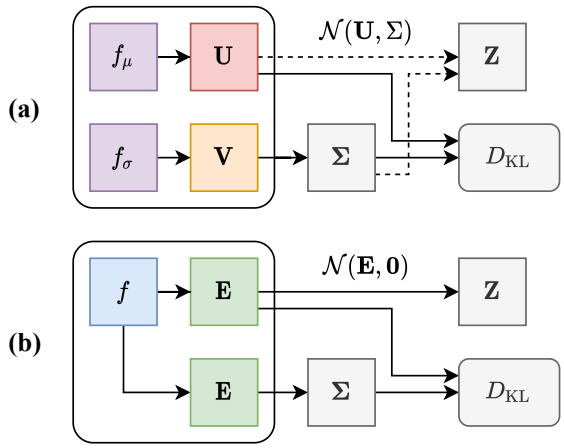


Figure 5.4: Comparison between the encoder structures of the (a) basic parameterization and (b) unified modeling. We use a single encoder f to deterministically generate $\mathbf{Z}$ while utilizing the strong regularization of the KL divergence.

zero. Third, $\mathbb{E}[\mathbf{v}_i^\top \mathbf{m}_2 \mathbf{v}_j^\top \mathbf{m}_2] = \mathbf{v}_i^\top \mathbf{v}_j$. We prove the lemma based on the definition of Σ. □

Unified Deterministic Modeling

The reparametrization trick allows us to sample different $\mathbf{Z}$ at each inference to approximate the expectation term of Equation (5.5). However, the sampling process can make the training unstable, considering the characteristics of our feature estimation problem where a) the inference needs to be done for all nodes at once, not for each node independently, and b) only a part of target variables have meaningful observations. These are notable differences from typical generative learning [68] where the sampling with reparametrization is effective.

We propose two ideas for improving the basic parameterization. First, we change q_ϕ into the Dirac delta function $\mathcal{N}(\mathbf{U}, \mathbf{0})$, where $\mathbf{0}$ is the zero matrix of size $n \times n$. This replaces the sampling process of $\mathbf{Z}$ with a deterministic

function that returns $\mathbf{U}$ at every inference, improving the stability of training. Then, we unify the two parameter matrices $\mathbf{U}$ and $\mathbf{V}$ as a single matrix $\mathbf{E}$, and generate it from a unified encoder f. This preserves the regularization effect of the KL divergence term even with the deterministic modeling, since the regularization for both $\mathbf{U}$ and $\mathbf{V}$ is applied to $\mathbf{E}$. Figure 5.4 compares the basic parameterization and the unified modeling.

As we assume $\mathbf{E} = \mathbf{U} = \mathbf{V}$ by the unified modeling, the first two terms of D_{KL} of Equation (5.6) become equivalent.

Lemma 9. *Let* $\mathbf{K} \in \mathbb{R}^{n \times n}$, $\mathbf{E} \in \mathbb{R}^{n \times d}$, *and* $\Sigma = \beta\mathbf{I} + \mathbf{E}\mathbf{E}^\top$. *Then,* $\mathrm{tr}(\mathbf{K}\Sigma) = \mathrm{tr}(\mathbf{E}^\top\mathbf{K}\mathbf{E}) + C$, *where* C *is a constant unrelated to* E.

Proof. $\mathrm{tr}(\mathbf{K}\Sigma) = \beta\mathrm{tr}(\mathbf{K}) + \mathrm{tr}(\mathbf{K}\mathbf{E}\mathbf{E}^\top)$ due to the definition of $\mathbf{K}$. The cyclic property of a trace makes $\mathrm{tr}(\mathbf{K}\mathbf{E}\mathbf{E}^\top) = \mathrm{tr}(\mathbf{E}^\top\mathbf{K}\mathbf{E})$. Thus, $\mathrm{tr}(\mathbf{K}\Sigma) = \beta\mathrm{tr}(\mathbf{K}) + \mathrm{tr}(\mathbf{E}^\top\mathbf{K}\mathbf{E})$, and $\mathrm{tr}(\mathbf{K})$ is a constant. □

Given our unified modeling and Lemma 9, the KL divergence of Equation (5.6) changes into a general regularizer function:

$$l_{\mathrm{GMRF}}(\mathbf{E}, \mathbf{A}) = \mathrm{tr}(\mathbf{E}^\top\mathbf{K}\mathbf{E}) - 0.5 \log |\mathbf{I} + \beta^{-1}\mathbf{E}^\top\mathbf{E}| + C, \tag{5.9}$$

where C is the same constant as in Equation (5.6).

The first term of Equation (5.9) is called the graph Laplacian regularizer and has been widely used in graph learning [76, 77]:

$$\mathrm{tr}(\mathbf{E}^\top\mathbf{K}\mathbf{E}) = \sum_{(i,j)\in\mathcal{E}} \left\| \frac{\mathbf{e}_i}{\sqrt{d_i}} - \frac{\mathbf{e}_j}{\sqrt{d_j}} \right\|_2^2, \tag{5.10}$$

where $\mathbf{e}_i$ and $\mathbf{e}_j$ are the i-th and the j-th row of $\mathbf{E}$, and d_i and d_j are the degrees

of nodes i and j, respectively. The minimization of Equation (5.10) makes adjacent nodes have similar representations in $\mathbf{E}$, and the symmetric normalization of $\mathbf{K}$ alleviates the effect of different node degrees in the regularization.

On the other hand, the second term of Equation (5.9) can be considered as measuring the amount of space that $\mathbf{E}$ occupies in the latent space. In other words, its maximization makes $\mathbf{e}_1, \cdots \mathbf{e}_n$ distributed sparsely, alleviating the effect of $\mathrm{tr}(\mathbf{E}^\top \mathbf{K}\mathbf{E})$ that squeezes the embeddings into a small space. The hyperparameter β controls the balance between the two terms.

5.3.3 Modeling by Deep Neural Networks

We propose an autoencoder structure to represent the target distributions $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$, $p_\theta(\mathbf{x}_i \mid \mathbf{z}_i)$, and $p_\rho(y_i \mid \mathbf{z}_i)$ by deep neural networks. We then describe how to train the networks.

Encoder Network

Our encoder network f aims to represent $q_\phi(\mathbf{Z} \mid \mathbf{X}, \mathbf{y}, \mathbf{A})$ with parameters ϕ. We use a graph convolutional network (GCN) [38] to model f, although a graph neural network of any structure can be used instead. The output of f is used as the parameter $\mathbf{E}$ of the distribution q_ϕ, which then becomes the latent variables $\mathbf{Z}$ due to our deterministic modeling.

If we use a GCN having two layers [38], the encoder function f is defined as $f(\mathbf{X}, \mathbf{A}; \phi) = \hat{\mathbf{A}}(\sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}_1))\mathbf{W}_2$, where $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2}$ is the normalized adjacency matrix, $\mathbf{D}$ is the degree matrix such that $D_{ii} = \sum_j A_{ij}$, σ is an activation function, $\mathbf{X}$ is the node feature matrix, and $\mathbf{W}_i$ is the weight matrix for layer i. We do not represent the bias terms for brevity.

Diagonal features The problem of using the GCN encoder is that we have only partial observations of node features, and thus the input matrix $\mathbf{X}$ is incomplete. There are ways to assign initial features to the nodes having no features, such as making random or one-hot vectors [38]. However, such partial imputation makes an imbalance between nodes based on whether the features are given or not, increasing the risk of overfitting. At the same time, in our feature estimation problem, it is essential to provide discriminative node features to learn the unique property of each target node i to accurately predict its unknown feature $\mathbf{x}_i$.

Thus, we adopt the identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ as $\mathbf{X}$ dropping all observations from the input of the encoder network f. This forces f to learn an independent embedding vector $\mathbf{w}_i$ for each node i at the i-th row of the weight matrix $\mathbf{W}_1$ whether $i \in \mathcal{V}_x$ or not.

Unit normalization A possible limitation of introducing the identity feature matrix is the large number of parameters in $\mathbf{W}_1$, which can make the training process unstable. We thus project the node representations $\mathbf{Z}$ generated from the encoder f into a unit hypersphere by normalizing each vector of node i as $\mathbf{z}_i / \|\mathbf{z}_i\|_2$. This does not alter the main functionality of making diverse representations of nodes that provide sufficient information for generating high-dimensional features, but improves the stability of training by restricting the output space [78].

Decoder Networks

We utilize two decoder networks g_x and g_y to model $p_\theta(\mathbf{x}_i \mid \mathbf{z}_i)$ and $p_\rho(y_i \mid \mathbf{z}_i)$, respectively. We assume that latent variables $\mathbf{Z}$ have sufficient information to construct the observed features and labels. Thus, we minimize the complexity of decoder networks by adopting the simplest linear transformation as $g_x(\mathbf{z}_i) = \mathbf{W}_x\mathbf{z}_i + \mathbf{b}_x$ and $g_y(\mathbf{z}_i) = \mathbf{W}_y\mathbf{z}_i + \mathbf{b}_y$, where $\mathbf{W}_x \in \mathbb{R}^{m\times d}$, $\mathbf{W}_y \in \mathbb{R}^{c\times d}$, $\mathbf{b}_x \in \mathbb{R}^m$ and $\mathbf{b}_y \in \mathbb{R}^c$ are learnable weights and biases, m is the number of features, and c is the number of classes.

How to model the distributions $p_\theta(\mathbf{x}_i \mid \mathbf{z}_i)$ and $p_\rho(y_i \mid \mathbf{z}_i)$ with the outputs of decoder networks depends on the property of the dataset. Many graph datasets [37, 61] for node classification have binary features and a single label for each node. Thus, we assume $p_\theta(\mathbf{x}_i \mid \mathbf{z}_i)$ as a multivariate Bernoulli distribution and $p_\rho(y_i \mid \mathbf{z}_i)$ as a categorical distribution. This makes the following loss terms that we aim to minimize in the training process.

$$l_{\text{BCE}}(\mathbf{x}_i, \hat{\mathbf{x}}_i) = -\sum_k (x_{ik} \log \hat{x}_{ik} + (1 - x_{ik}) \log(1 - \hat{x}_{ik})) \tag{5.11}$$

$$l_{\text{CE}}(y_i, \hat{\mathbf{y}}_i) = -\sum_k \mathbb{I}(y_i = k) \log \hat{y}_{ik}, \tag{5.12}$$

where $\hat{\mathbf{x}}_i = \sigma(g_x(\mathbf{z}_i))$ and $\hat{\mathbf{y}}_i = \text{softmax}(g_y(\mathbf{z}_i))$ are the outputs of decoder networks, σ is the logistic sigmoid function, and $\mathbb{I}$ returns one if the condition holds and zero otherwise.

Training

We update all three networks f, g_x, and g_y in an end-to-end way using a gradient-based optimizer. We summarize the overall objective function to min-

imize as

$$l(\Theta) = \sum_{i \in \mathcal{V}_x} l_{\mathrm{BCE}}(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \sum_{i \in \mathcal{V}_y} l_{\mathrm{CE}}(y_i, \hat{\mathbf{y}}_i) + \lambda l_{\mathrm{GMRF}}(\mathbf{Z}, \mathbf{A}), \tag{5.13}$$

where l_{GMRF} is our proposed regularizer shown in Equation (5.9). We make two changes from the ELBO of Equation (5.4). First, we use a hyperparameter λ to adjust the amount of regularization. Second, we modify the binary cross entropy (BCE) loss l_{BCE} to balance the effects of zero and nonzero entries of the true features by adjusting the class weights based on occurrences [67].

5.3.4 Complexity Analysis

We analyze the time and space complexities of MGA, assuming the GCN encoder f having two layers. We define space complexity as the amount of space required to store intermediate data during each inference. Let d, m, and c be the size of latent variables, the number of features, and the number of labels, respectively.

Lemma 10. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, the time complexity of MGA is $O((d^2 + md + cd)|\mathcal{V}| + d|\mathcal{E}|)$ for each inference.*

Proof. MGA consists of an encoder f and two decoders g_x and g_y. The complexity of f is $O(d^2|\mathcal{V}| + d|\mathcal{E}|)$ assuming the identity feature matrix. The complexities of g_x and g_y are $O(md|\mathcal{V}|)$ and $O(cd|\mathcal{V}|)$, respectively. □

Lemma 11. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, the space complexity of MGA is $O(d|\mathcal{V}| + |\mathcal{E}| + d^2 + md + cd)$ for each inference.*

Proof. MGA consists of an encoder f and two decoders g_x and g_y. The com-

plexity of f is $O(d|\mathcal{V}| + |\mathcal{E}| + d^2)$, and the complexities g_x and g_y are $O(md)$ and $O(cd)$, respectively. □

Lemma 10 and 11 show that MGA is an efficient method whose complexity is linear with both the numbers of nodes and edges of the given graph. The GMRF regularizer does not affect the inference of MGA, because it is used only at the training time. Still, the time and space complexities of the GMRF loss l_{GMRF} of Equation (5.9) are $O(d^2|\mathcal{V}| + d|\mathcal{E}| + d^3)$ and $O(d|\mathcal{V}| + |\mathcal{E}| + d^2)$, respectively, which are linear with both the numbers of nodes and edges.

5.4 Experiments

We perform experiments to answer the following questions:

Q1. **Feature estimation (Section 5.4.2).** Does MGA show higher accuracy in feature estimation than those of baselines?

Q2. **Node classification (Section 5.4.3).** Are the features generated by MGA meaningful for node classification?

Q3. **Effect of observed labels (Section 5.4.4).** Does the observation of labels help generating more accurate features?

Q4. **Scalability (Section 5.4.5).** How does the computational time of MGA increase with the number of edges?

Q5. **Ablation study (Section 5.4.6).** How does the performance of MGA for feature estimation change by the GMRF regularizer and the unified modeling of the encoder function?

Table 5.1: Summary of datasets.

Dataset	Nodes	Edges	Density	Features	Classes
Cora[1]	2,708	5,429	0.0007	1,433	7
Citeseer[1]	3,327	4,732	0.0004	3,703	6
Photo[2]	7,650	119,081	0.0020	745	8
Computers[2]	13,752	245,861	0.0013	767	10
Steam[3]	9,944	266,981	0.0027	352	1

[1] `https://github.com/kimiyoung/planetoid`
[2] `https://github.com/shchur/gnn-benchmark`
[3] `https://github.com/xuChenSJTU/SAT-master-online`

5.4.1 Experimental Setup

We introduce our experimental setup including datasets, baseline approaches, evaluation metrics, and training processes.

Datasets We use graph datasets used in previous work [67] as summarized in Table 7.2. Node features in all of these datasets are zero-one binary vectors, and each node has a single discrete label. Cora and Citeseer [37] are citation graphs consisting of research articles whose features are textual contents and classes are research areas. Photo and Computers [61] are product graphs collected from Amazon [56], whose features are product reviews and classes are product categories. Steam [67] is a graph of games whose features are game descriptions. All nodes in Steam have the same class, and thus the dataset cannot be used for node classification.

Baselines We compare MGA with existing models for feature estimation. NeighAggre [79] is a simple approach that aggregates the features of neighboring nodes through mean pooling. VAE [68] is a generative model that learns latent

representations of examples. GCN [38], GraphSAGE [40], and GAT [39] are popular graph neural networks that have been used in various domains. We report the best performance among the three models as GNN* for brevity.

GraphRNA [65] and ARWMF [66] are recent methods for representation learning, which can be applied for generating features. SAT [67] is the state-of-the-art model for missing feature estimation, which trains separate autoencoders with a shared latent space for the features and graphical structure, respectively. We include two versions of SAT in experiments, which adopt GraphSAGE and GAT as the backbone networks, respectively.

Evaluation metrics We evaluate the performance of feature estimation with two evaluation metrics: recall and the normalized discounted cumulative gain (nDCG). Given a binary feature vector, we treat each nonzero entry as a target item, considering the task as a ranking problem to find all nonzero entries. Recall at k measures the ratio of true entries contained in the top k predictions for each node, while nDCG at k measures the overall quality of a ranking in terms of information retrieval. We vary k in $\{3, 5, 10\}$ in the Steam dataset and $\{10, 20, 50\}$ in the other datasets, because Steam has fewer features and thus a prediction is generally easier.

Experimental process We take different processes of experiments for feature estimation and node classification. For feature estimation, we split all nodes at each dataset into the training, validation, and test sets by the 4:1:5 ratio as in previous work [67]. We train each model based on the observed features of training nodes and find the parameters that maximize the validation performance. We run each experiment ten times and report the average.

For node classification, we take only the test nodes of feature estimation, whose features are generated by our MGA or baseline models. Then, we perform the 5-fold cross-validation in the target nodes, evaluating the quality of generated features with respect to the accuracy of node classification. We use a multilayer perceptron (MLP) and a GCN as classifiers. We also use the induced subgraph of target nodes for the training and evaluation of GCNs.

Even though our MGA can utilize observed labels as additional evidence, we do not assume the observation of labels unless otherwise noted. This is to make a fair comparison between MGA and baseline models that assume only the observation of features. We perform experiments in Section 5.4.4 to show how observed labels affect the performance of our MGA.

Hyperparameters We set the hyperparameters of our MGA as $d = 256$, $p = 0.5$, $l = 2$, where d is the size of latent variables, p is the dropout probability, and l is the number of GCN layers. We search both λ and β in $\{0.1, 1.0\}$ based on the validation data and use the Adam [60] optimizer with the learning rate $r = 0.001$. We use a different hyperparameter setting only in the Steam dataset, whose characteristics are different from those of the other datasets: $p = 0$, $r = 0.005$, and no unit normalization of the latent variables. We take the experimental results of baselines from previous work [67] that optimized the hyperparameters for our datasets. All of our experiments were done at a workstation with RTX 2080 based on PyTorch [80]. We report in Section 5.8 the performance of MGA with different values of λ and β as a parameter study.

Table 5.2: Evaluation of MGA and baseline approaches for missing feature estimation with respect to (top) recall and (bottom) nDCG. The best is in bold, and the second best is underlined. Our MGA outperforms all baselines in most cases.

Evaluation by recall at k

Model	**Cora**			**Citeseer**			**Computers**			**Photo**			**Steam**		
	@10	@20	@50	@10	@20	@50	@10	@20	@50	@10	@20	@50	@3	@5	@10
NeighAggre	.0906	.1413	.1961	.0511	.0908	.1501	.0321	.0593	.1306	.0329	.0616	.1361	.0603	.0881	.1446
VAE	.0887	.1228	.2116	.0382	.0668	.1296	.0255	.0502	.1196	.0276	.0538	.1279	.0564	.0820	.1251
GNN*	.1350	.1812	.2972	.0620	.1097	.2058	.0273	.0533	.1278	.0295	.0573	.1324	.2395	.3431	.4575
GraphRNA	.1395	.2043	.3142	.0777	.1272	.2271	.0386	.0690	.1465	.0390	.0703	.1508	.2490	.3208	.4372
ARWMF	.1291	.1813	.2960	.0552	.1015	.1952	.0280	.0544	.1289	.0294	.0568	.1327	.2104	.3201	.4512
SAT-SAGE	.1356	.1981	.3165	.0704	.1163	.2174	.0419	.0738	.1562	**.0483**	.0766	.1601	.2518	.3470	.4845
SAT-GAT	.1653	.2345	.3612	.0811	.1349	.2431	.0421	.0746	.1577	.0427	.0765	.1635	.2536	**.3620**	.4965
MGA	**.1718**	**.2486**	**.3814**	**.0943**	**.1539**	**.2782**	**.0437**	**.0769**	**.1602**	.0446	**.0798**	**.1670**	**.2565**	**.3620**	**.4996**

Evaluation by nDCG at k

Model	**Cora**			**Citeseer**			**Computers**			**Photo**			**Steam**		
	@10	@20	@50	@10	@20	@50	@10	@20	@50	@10	@20	@50	@3	@5	@10
NeighAggre	.1217	.1548	.1850	.0823	.1155	.1560	.0788	.1156	.1923	.0813	.1196	.1998	.0955	.1204	.1620
VAE	.1224	.1452	.1924	.0601	.0839	.1251	.0632	.0970	.1721	.0675	.1031	.1830	.0902	.1133	.1437
GNN*	.1791	.2099	.2711	.1026	.1423	.2049	.0673	.1028	.1830	.0712	.1083	.1896	.3366	.4138	.4912
GraphRNA	.1934	.2362	.2938	.1291	.1703	.2358	.0931	.1333	.2155	.0959	.1377	.2232	.3437	.4023	.4755
ARWMF	.1824	.2182	.2776	.0859	.1245	.1858	.0694	.1053	.1851	.0727	.1098	.1915	.3066	.3877	.4704
SAT-SAGE	.1905	.2320	.2947	.1179	.1563	.2227	.1030	.1457	.2333	.1082	.1475	.2402	.3529	.4271	.5133
SAT-GAT	.2250	.2723	.3394	.1385	.1834	.2545	.1030	.1463	.2346	.1047	.1498	.2421	**.3585**	**.4400**	.5272
MGA	**.2381**	**.2894**	**.3601**	**.1579**	**.2076**	**.2892**	**.1068**	**.1509**	**.2397**	**.1084**	**.1549**	**.2472**	.3567	.4391	**.5299**

5.4.2 Performance on Feature Estimation

Table 5.2 compares MGA and baseline models for feature estimation. MGA outperforms all baselines by a significant margin in most cases; MGA shows up to 16.3% and 14.0% higher recall and nDCG, respectively, compared to the best competitors.

The difference in performance between models depends on the datasets. The improvement of MGA over baselines is the largest in Cora and Citeseer, which are citation graphs, and the smallest in the Steam dataset. This difference comes from various characteristics of datasets, but the main factors are the number of features and the density of a graph. The difficulty of feature estimation is affected by these two factors, since a) it is difficult to accurately predict high-dimensional features and b) sparse graphs are likely to provide less evidence to correlate the target variables. Thus, the improvement in the citation graphs, which are the most sparse and have the largest number of features, is important for evaluation.

5.4.3 Performance on Node Classification

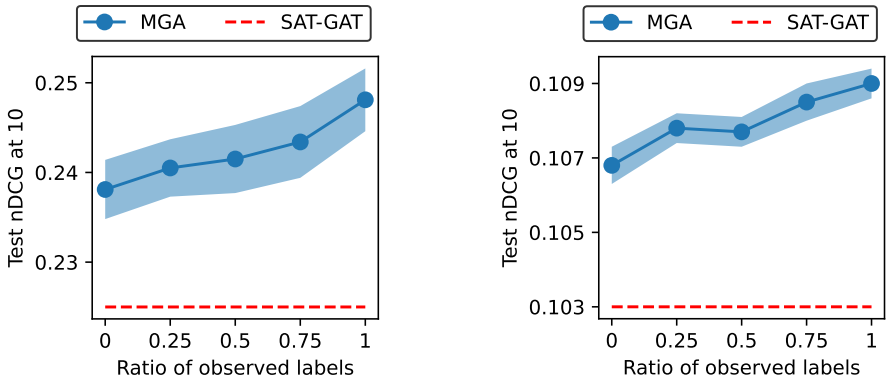
Table 5.3 shows the accuracy of node classification with two classifiers: an MLP and a GCN. MGA outperforms all baseline models in most cases, making a consistency with the result in Table 5.2; MGA shows up to 7.0% higher classification accuracy compared to the best competitors. The Steam dataset is not included in Table 5.3, since it has the same label for all nodes.

The improvement of MGA from the baseline methods is much larger with the MLP classifier than with the GCN. This is because MGA successfully embeds the structural information of each graph in the generated features, making

Table 5.3: Comparison between MGA and baselines by node classification accuracy, where each classifier is trained with the generated features. MGA outperforms all baseline methods in most cases, especially with an MLP classifier.

Model	**Classifier**	**Cora**	**Cite.**	**Comp.**	**Photo**
NeighAggre	MLP	.6248	.5549	.8365	.8846
VAE	MLP	.2826	.2551	.3747	.2598
GNN*	MLP	.4852	.3933	.3747	.2598
GraphRNA	MLP	.7581	.6320	.6968	.8407
ARWMF	MLP	.7769	.2267	.5608	.4675
SAT-SAGE	MLP	.7032	.5936	<u>.8396</u>	<u>.9035</u>
SAT-GAT	MLP	<u>.7937</u>	<u>.6475</u>	.8201	.8976
MGA (proposed)	MLP	**.8493**	**.6757**	**.8806**	**.9209**

Model	**Classifier**	**Cora**	**Cite.**	**Comp.**	**Photo**
NeighAggre	GCN	.6494	.5413	.8715	.9010
VAE	GCN	.3011	.2663	.4023	.3781
GNN*	GCN	.5779	.4278	.4034	.3789
GraphRNA	GCN	.8198	.6394	.8650	.9207
ARWMF	GCN	.8205	.2764	.7400	.6146
SAT-SAGE	GCN	.8255	.6547	<u>.8834</u>	.9234
SAT-GAT	GCN	**.8579**	<u>.6767</u>	.8766	<u>.9260</u>
MGA (proposed)	GCN	<u>.8533</u>	**.6879**	**.8854**	**.9264**



(a) Cora (b) Computers

Figure 5.5: Accuracy of MGA for feature estimation with additional labels. MGA effectively uses the given labels, making more accurate predictions.

the MLP utilize the relationships between examples. Another observation is that SAT-SAGE makes a higher accuracy than that of SAT-GAT in Computers, although SAT-GAT makes more accurate features as in Table 5.2. This implies that the high accuracy of feature estimation does not guarantee the high accuracy of node classification, and only a carefully designed approach performs well in both experiments.

5.4.4 Effect of Observed Labels

Figure 5.5 evaluates the performance of MGA for feature estimation with different ratios of observed labels. For instance, if the ratio is 0.5, half of all nodes have observed labels: $|\mathcal{V}_y| = 0.5|\mathcal{V}|$. Note that the experiments for Tables 5.2 and 5.3 are done with no observed labels for a fair comparison with the baseline models; the results of these experiments correspond to the leftmost points in Figure 5.5. We also report the performance of SAT-GAT for comparison.

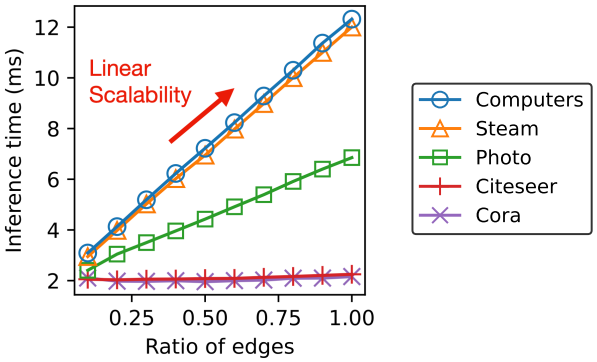
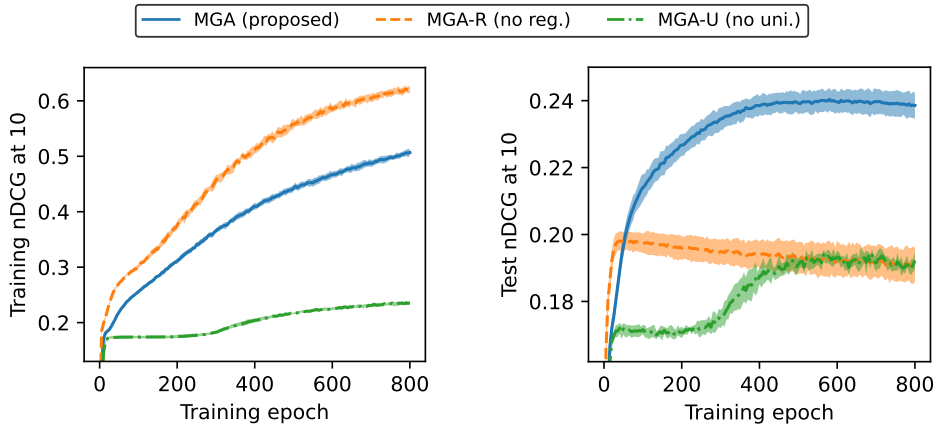


Figure 5.6: The inference time of MGA in graphs of different sizes. We randomly sample nine subgraphs for each dataset changing the number of edges. MGA shows the linear scalability in all datasets.

MGA shows higher accuracy with more observations of labels in both datasets, demonstrating its ability to utilize labels to improve the performance of feature estimation. Since the parameters need to be optimized to predict both features and labels accurately, the observed labels work as an additional regularizer that guides the training of latent variables. Still, the improvement is limited even with the full observation of labels, since the information provided by the labels is insufficient to perfectly predict high-dimensional features, which are determined also by other factors.

5.4.5 Scalability

Figure 7.5 shows the scalability of MGA with respect to the number of edges. For each dataset, we sample nine random subgraphs having different sizes from $0.1|\mathcal{E}|$ to $0.9|\mathcal{E}|$, where $|\mathcal{E}|$ denotes the number of original edges. We measure the inference time of MGA in each graph ten times and report the average. The figure shows that MGA has linear scalability with respect to the number



(a) Training (b) Test

Figure 5.7: Ablation study for MGA on the Cora dataset for the feature estimation problem. MGA-R denotes MGA without the KL divergence regularizer in Equation (5.6), while MGA-U denotes MGA without the unified modeling.

of edges in all datasets, supporting our claim in Lemma 10. The trend is not clearly visible in Cora and Citeseer, whose sizes are too small and thus the inference time makes a negligible difference.

5.4.6 Ablation Study

Figure 5.7 shows an ablation study that compares MGA with two base methods MGA-R and MGA-U. MGA-R uses the deterministic modeling as in our MGA but removes the KL divergence term of Equation (5.6) from the objective function. MGA-U uses the regularizer but without our unified deterministic modeling; we sample the latent variables for MGA-U following Equation (5.8), where f_μ and f_σ are both GCNs having the same structure as our f. We visualize the mean and standard deviation of ten runs.

MGA shows the best test accuracy throughout the training with a stable

curve. The training accuracy is the best at MGA-R, since it overfits to training nodes without the regularizer term. The test accuracy of MGA-U is similar to that of MGA-R at the later epochs but the training accuracy of MGA-U is the lowest among the three methods. This is because MGA-U has an unstable training process, which makes it fail even at maximizing the training accuracy. The low stability of MGA-U is shown also by the large standard deviation of its test accuracy, which increases as the training proceeds.

5.5 Related Works

Node classification Graph neural networks (GNN) [40, 39, 81, 82] refer to deep neural networks designed for graph-structured data. Given a graph representing an explicit manifold of examples, GNNs correlate adjacent nodes with various aggregation functions that are often called graph convolutions [38]. Recent works improve the performance of GNNs by considering various aspects such as graph isomorphism [83], position-awareness [84, 85], curriculum learning [86], structure estimation [87], or contrastive learning [88].

GNNs require the feature vectors of all nodes. Thus, one needs to generate artificial features before running a GNN over a graph with missing features. Derr et al. [89] and Cui et al. [90] make features from the graph structure information. Kipf and Welling [38] model the missing features as one-hot vectors to learn an independent embedding vector for each node. Zhao and Akoglu [91] leave the missing features as zero vectors and propose a new regularizer that propagates the observed features effectively.

Our MGA enables a GNN to be applied to graphs with partial observations by generating pseudo features. The main advantage of feature estimation is that

the modification of a GNN classifier is not required, regardless of the number of observations given in the original graph. Previous works that directly deal with partially observed graphs require finding new hyperparameters [91] or even making a new weight matrix [38] when the number of observations changes, making it difficult to reuse a trained model.

Node representation learning Unsupervised node representation learning [92] is to represent each node as a low-dimensional vector that summarizes its properties embedded in the graph structure and node features. Traditional methods [48, 93, 94, 47] use only the structural information, while recent ones utilize both the node features and graph structure to maximize the performance [92, 81]. Such methods make embeddings in a latent space, while we aim to learn the representations of nodes in the high-dimensional feature space; the generated features from our MGA are interpretable in the feature domain, unlike the embedding vectors that have arbitrary meanings.

There are recent works that can be used directly for the feature estimation problem [65, 66, 67], which are adopted as the main competitors in our experiments. The main advantage of MGA over the previous approaches is the existence of a strong regularizer that allows us to effectively propagate the partial observations to the entire graph, avoiding the overfitting problem even with sufficient representation power to generate high-dimensional features.

5.6 Summary

We have proposed MGA (Markov Graph Autoencoder), an accurate method for missing feature estimation. MGA generates high-dimensional features of nodes

from a graph with partial observations, and its structure is carefully designed by variational inference to maximize the joint likelihood of observations. The core idea of MGA is the structural regularizer that assumes the prior of latent variables as Gaussian Markov random field, which considers the graph structure as the main evidence for modeling the correlations between variables. MGA outperforms previous methods for both feature estimation and node classification, achieving the state-of-the-art accuracy in five benchmark datasets. Future works include improving MGA to also generate edge attributes or extending the domain of MGA into heterogeneous or temporal graphs.

5.7 Appendix: Problem Formulation

We formally define our feature estimation problem and discuss how to evaluate the generated features with node classification.

Problem 1. *The feature estimation problem is defined as follows. We are given the following:*

- *An undirected graph* $G = (\mathcal{V}, \mathcal{E})$.
- *Sets* $\mathcal{V}_x$ *and* $\mathcal{V}_y$ *of nodes such that* $\mathcal{V}_x \subset \mathcal{V}$ *and* $\mathcal{V}_y \subseteq \mathcal{V}$.
- *A set* $\mathcal{X} = \{\mathbf{x}_i \mid i \in \mathcal{V}_x\}$ *of features for* $\mathcal{V}_x$.
- *A set* $\mathcal{Y} = \{y_i \mid i \in \mathcal{V}_y\}$ *of labels for* $\mathcal{V}_y$.

Then, the problem is to predict the unknown feature $\mathbf{x}_i$ *of each test node* $i \in \mathcal{V} \setminus \mathcal{V}_x$ *based on the observations of* $\mathcal{X}$ *and* $\mathcal{Y}$.

The performance of feature estimation is mainly evaluated by the direct comparison between predictions and true features. Still, generated features can be evaluated with respect to other problems that benefit from the generated

features. We utilize the node classification problem in this regard, which is a popular problem in the graph domain where the quality of node features plays a crucial role for the overall performance. Note that the observed labels of $\mathcal{V}_y$ are *optional* information that helps feature estimation. We use $\mathcal{V}_y = \emptyset$ as the default setting of experiments, especially when the generated features are evaluated by node classification.

Problem 2. *The feature estimation problem for node classification is defined as follows. We are given the following:*

- *An undirected graph* $G = (\mathcal{V}, \mathcal{E})$.
- *Sets* $\mathcal{V}_x$ *and* $\mathcal{V}_t$ *of nodes such that* $\mathcal{V}_x \cap \mathcal{V}_t = \emptyset$.
- *A set* $\mathcal{X} = \{\mathbf{x}_i \mid i \in \mathcal{V}_x\}$ *of features for* $\mathcal{V}_x$.
- *A set* $\mathcal{Y}_t = \{y_i \mid i \in \mathcal{V}_t\}$ *of labels for* $\mathcal{V}_t$.
- *A classifier* f *that predicts the labels of nodes as* $\hat{\mathbf{y}} = f(G, \mathbf{X})$ *given a graph* G *and a node feature matrix* $\mathbf{X}$.

Then, the problem is to generate artificial features $\hat{\mathbf{X}}$ *for* $\mathcal{V} \setminus \mathcal{V}_x$ *to maximize the classification accuracy of* f *for the test nodes, which are given as* $\mathcal{V} \setminus (\mathcal{V}_x \cup \mathcal{V}_y)$*, that contain no true features.*

The baseline of Problem 2 is to perform node classification without utilizing any features, solely based on the graph structure and the observed labels of $\mathcal{V}_t$. We expect to improve the accuracy of classification if we generate node features that are meaningful for modeling the relationships between nodes for the target classes. Problem 2 evaluates the generated features from a perspective different from Problem 1, based on the motivation that we are often interested in how well the features describe the relationships between nodes, rather than their

Table 5.4: The performance of our MGA with different values of λ and β. We report the recall @ k scores with $k = 10$.

λ	β	Cora	Cite.	Comp.	Photo	Steam
0.1	0.1	.1635	**.0943**	.0431	.0438	.2524
0.1	1.0	.1622	.0932	.0430	.0437	.2528
1.0	0.1	**.1718**	.0925	**.0437**	**.0446**	**.2565**
1.0	1.0	.1715	.0917	.0436	**.0446**	.2557

exact values.

To summarize, we aim to design an approach that satisfies both goals together: a) to accurately predict the unknown features and b) to generate features that are helpful for node classification. This evaluation scheme is done also in previous work [67].

5.8 Appendix: Parameter Sensitivity

Table 5.4 shows the parameter sensitivity of our MGA with respect to λ and β. We report only the recall @ k scores with $k = 10$ since other evaluation metrics show similar patterns. MGA works well in all settings of λ and β, compared with the baseline approaches in Table 5.2, demonstrating its robustness for the choice of hyperparameters. The setting with $\lambda = 1.0$ and $\beta = 0.1$ works the best in four of the five datasets. We report the test performance in Table 5.4, but the trend is the same in the validation data, allowing us to find the best hyperparameters during the training.

Part II

Structural Modification

Chapter 6

Graph Sampling for Efficient Inference

How can we run graphical inference on large graphs efficiently and accurately? Many real-world networks are modeled as graphical models, and graphical inference is fundamental to understand the properties of those networks. We propose a novel approach for fast and accurate inference, which first samples a small subgraph and then runs inference over the subgraph instead of the given graph. This is done by the bounded treewidth (BTW) sampling, our novel algorithm that generates a subgraph with guaranteed bounded treewidth while retaining as many edges as possible. We first analyze the properties of BTW theoretically. Then, we evaluate our approach on node classification and compare it with the baseline which is to run loopy belief propagation (LBP) on the original graph. Our approach can be coupled with various inference algorithms: it shows higher accuracy up to 13.7% with the junction tree algorithm, and allows faster inference up to 23.8 times with LBP. We further compare BTW with previous graph sampling algorithms and show that it gives the best accuracy.

6.1 Motivation

Given a large graph whose nodes represent discrete random variables, how can we compute their posterior marginals efficiently? Graphical inference is a cru-

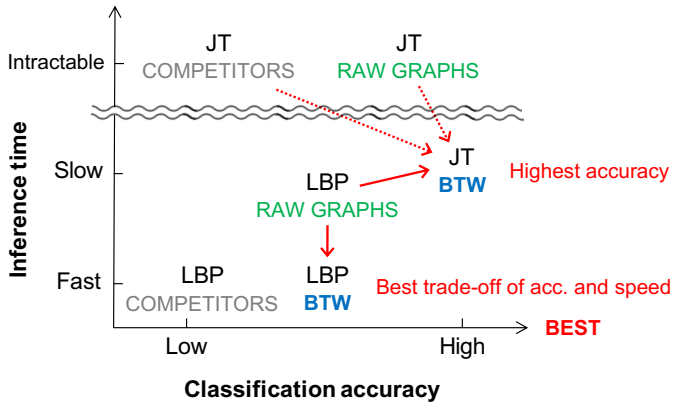
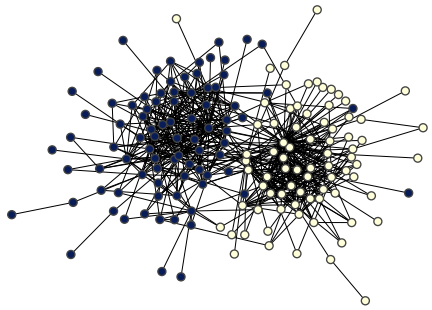


Figure 6.1: Advantages of BTW with two kinds of inference algorithms. BTW a) gives the best accuracy when the junction tree (JT) algorithm is used and b) speeds up the inference without hurting accuracy when LBP is used.

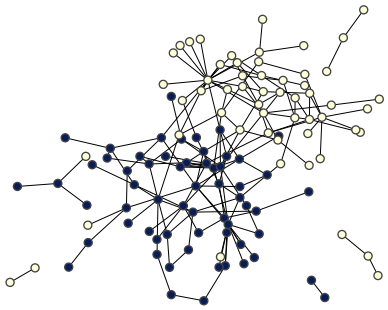
cial task in data mining and machine learning, which has been applied to solve various node classification problems such as malware detection [16], social network analysis [19, 18, 32, 1, 95], and recommender systems [33].

Loopy belief propagation (LBP) [5] is an inference algorithm which has been used widely for node classification. Yet, LBP has crucial limitations such that 1) it performs approximate inference rather than exact inference, and 2) its convergence is not guaranteed for general graphs; the conditions that LBP converges have been found only for restricted settings [96, 97]. These limitations make LBP difficult to be applied when the stability of inference is required. There are previous works [98, 99] that aim to solve the convergence problem by approximating LBP by a series of linear operations, but the amount of approximations is still not bounded.

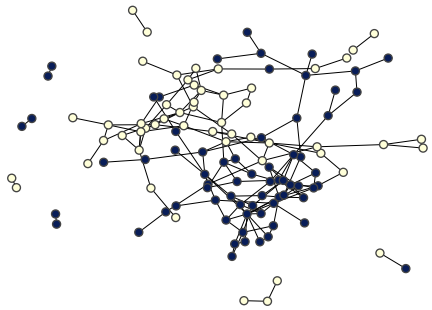
The junction tree algorithm [6] is an exact inference algorithm that does not suffer from the unstable convergence. However, this approach requires expo-



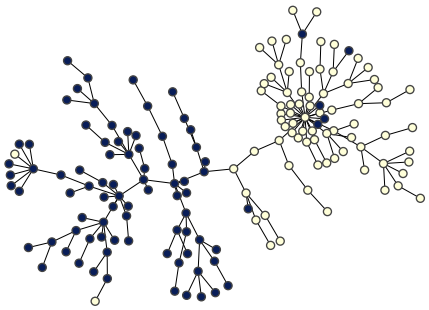
(a) Original graph.
(TW $\approx$ 24)



(b) Sampled subgraph by RE.
(TW $\approx$ 8)



(c) Sampled subgraph by FF.
(TW $\approx$ 7)



(d) Sampled subgraph by BTW.
(TW = 1)

Figure 6.2: Guaranteed treewidth by BTW. (a) A web network of 163 blogs which are divided into two political groups. (b) The random edge sampling and (c) the forest fire sampling [100] generate subgraphs that have disconnected components and large treewidth ($\geq$ 7). (d) Our proposed BTW generates a connected subgraph with the same number of edges as in (b) and (c), but with bounded treewidth of 1 while preserving the separate political groups.

nential time and space with the treewidth of a graph [6]; thus, efficient inference requires graphs with *bounded treewidth*. Although it is NP-compete to compute the treewidth of a graph [101], recent advances [102, 103, 104] in structure learning of Bayesian networks are able to learn from data a high-scoring graph that respects a treewidth bound; this allows exact inference to be tractable. However, there are many applications where the graph is given, rather than learned. In such cases, exact inference is intractable if there is no control of treewidth.

In this chapter, we propose a novel approach which allows efficient inference on large graphs. Our idea comprises two steps. First, we sample the original graph obtaining a bounded-treewidth subgraph. This is done by our proposed bounded treewidth (BTW) sampling, a novel algorithm to generate a subgraph with bounded treewidth while retaining as many edges as possible. Once we obtain the subgraph, we run on it a) exact inference by the junction tree algorithm or b) approximate inference by LBP based on the purpose. If we run the junction tree algorithm, our approach is more accurate than running LBP on the original graph, despite its slow speed. We also have guaranteed convergence as the algorithm runs exact inference. If we run LBP, our approach shows similar accuracy with much faster computational time. These results are due to the success of our BTW sampling that maintains essential edges of the original graph with guaranteed bounded treewidth.

To the best of our knowledge, BTW is the first graph sampling algorithm that controls the treewidth of subgraphs and allows efficient graphical inference. The problem of generating subgraphs similar to an original graph has been studied widely in the data mining community [100, 105, 106], but none of

the previous methods controls the treewidth of subgraphs. Furthermore, BTW considers cliques as minimal units of sampling, unlike other approaches that sample an edge at each iteration until the desired size is achieved. This gives every node a chance to preserve its comprehensive context which is essential for accurate graphical inference.

We evaluate our approach with the junction tree algorithm and LBP, which are the representative inference algorithms, on various real-world networks. As a result of extensive experiments, we demonstrate the following strengths of our approach:

- The junction tree algorithm on the subgraph generated from BTW achieves the highest accuracy, which is up to 13.7% higher than that from LBP on the original graph.
- LBP on the subgraph generated from BTW is up to 23.8× faster than LBP on the original graph, providing similar or even higher accuracy in some datasets.
- When we run LBP on subgraphs generated from BTW and other sampling algorithms, BTW shows up to 5.6% higher accuracy than the best competitors do.

These observations are summarized as Figure 6.1. BTW enables the junction tree algorithm which is intractable in raw graphs, leading to the highest accuracy. BTW also allows faster inference when LBP is used while maintaining the original accuracy. The detailed results are presented in Section 6.4.

Figure 6.2 shows that BTW generates a subgraph with guaranteed treewidth on a web network of 163 blogs with two political groups. The previous random edge (RE) sampling and forest fire (FF) sampling generate subgraphs

of Figures 6.2b and 6.2c, respectively, which contain disconnected components, noisy connections between the different groups, and large treewidth (≥ 7). On the other hand, BTW generates a connected subgraph of Figure 6.2d, which has low treewidth of 1 and preserves the separate political groups with the same number of edges as in Figures 6.2b and 6.2c. As a result, the junction tree algorithm is tractable only in the graph of Figure 6.2d, and LBP shows the best performance in that graph. The approximate treewidth is reported in (a) to (c) by the *min-fill-in* and *min-degree* heuristics, since the exact computation is intractable [107].

6.2 Related Works

We introduce related works for our proposed approach, which are categorized into treewidth, inference algorithms, and sampling algorithms. We denote an undirected graph by $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of nodes and edges, respectively.

6.2.1 Treewidth

Intuitively, the treewidth of a graph quantifies the extent to which it resembles a tree structure. We introduce the concepts of a clique and triangulation before defining formally the treewidth.

Definition 1 (Clique)**.** *Given a graph G, a set $\mathcal{C}$ of nodes is a* clique *if all of its nodes are pairwise connected. A clique of k nodes is called a k-clique and it contains $k(k-1)/2$ edges. A clique $\mathcal{C}$ is* maximal *if it is not a subset of a larger clique in G.*

Definition 2 (Triangulated graph)**.** *An undirected graph T is* triangulated *if all the chordless cycles existing in T are of length less than or equal to 3. Note that a chordless cycle is a cycle such that no two nodes of the cycle are connected by an edge that does not belong to the cycle.* Triangulation *is the process of generating a triangulated graph from a graph G by inserting additional edges cutting all the cycles of length greater than 3 in G.*

Definition 3 (Treewidth)**.** *The* treewidth *of a triangulated graph T is the number of nodes in its largest maximal clique minus one. The treewidth of a general graph G is the minimum treewidth among all possible triangulations.*

Given a graph, it is intractable to compute its treewidth because it requires all possible triangulations [107]. Instead, the treewidth can be bounded to k if the graph is shown as a subgraph of a k-tree.

Definition 4 (k-tree)**.** *An undirected graph $K = (\mathcal{V}, \mathcal{E})$ is a k-*tree *of treewidth k if the addition of any edge $(u, v) \notin \mathcal{E}$ among its nodes $u, v \in \mathcal{V}$ increases its treewidth k, or it is a $(k+1)$-clique. A subgraph of a k-tree is called a* partial *k-tree, and its treewidth is smaller than or equal to k.*

We can generate a k-tree by the following inductive process [108]:

- **Base case:** a clique with $k+1$ nodes is a k-tree.
- **Inductive step:** given a k-tree K_n on n nodes, a new k-tree K_{n+1} is obtained by connecting a new node u to a k-clique $\mathcal{C}$ in K_n and generating k additional edges between u and $\mathcal{C}$. This adds a new $(k+1)$-clique of nodes $\{u\} \cup \mathcal{C}$ to K_n.

Learning Markov networks that best represent given observations or probability distributions has been studied widely [109, 110]. On the other hand,

score-based learning of Bayesian networks with bounded treewidth has been recently addressed [104, 111]. However, none of the previous approaches has used the idea of bounded treewidth learning to sample a subgraph that is suitable for accurate and efficient graphical inference.

6.2.2 Inference Algorithms

Belief propagation (BP) is an inference algorithm for Markov networks [5], which performs variable elimination efficiently. Given potentials of the variables in a graph, BP efficiently computes their posterior marginals by passing messages between the variables. BP computes the exact marginals when the graph is a tree. Otherwise, we run its variations described below.

Loopy belief propagation (LBP) is an approximate inference algorithm for cyclic graphs [5]. Unlike BP that passes the messages sequentially and only once following the tree structure, LBP updates all messages iteratively until convergence. However, since its convergence is not guaranteed, one needs to adjust the parameters to stabilize the termination of the algorithm, which are represented as a potential or an affinity matrix [10, 52]. There exist various approaches to guarantee its convergence by linear approximations, but the amount of approximation is not bounded [98, 99].

The *junction tree algorithm* is an exact inference algorithm for cyclic graphs [6]. It triangulates a given graph G and generates a supergraph T. Then, it finds the maximal cliques on T and builds a maximum weight spanning tree over the cliques where the weight between two cliques is the number of shared variables. We get the exact marginals by running BP on the resulting tree. However, its time and space complexities are exponential with the size of the largest

maximal clique in T, which depends on how we triangulate G. Since finding an optimal triangulation is intractable, a typical approach is to resort to a heuristic approach [112]. As a result, a tractable computation of the junction tree algorithm is not guaranteed on a general graph.

6.2.3 Graph Sampling

Graph sampling is a task of generating subgraphs that preserve the properties of a graph. Given an undirected graph $G = (\mathcal{V}, \mathcal{E})$, we aim to generate a subgraph $U = (\mathcal{V}', \mathcal{E}')$ such that $\mathcal{V} = \mathcal{V}'$ and $\mathcal{E}' \subset \mathcal{E}$. In this chapter, we focus on the *edge-sampling* which samples only the edges not changing the nodes. This is because our objective is to run graphical inference efficiently. Each node is an essential variable which cannot be removed from the graph.

Graph sampling has been studied widely. Leskovec and Faloutsos [100] compared different graph sampling algorithms and showed that the forest fire sampling [113] and the random walk sampling work well in maintaining the original properties. Li et al. [105] showed drawbacks of existing random walk based algorithms and proposed two novel methods. Wang et al. [114] compared various sampling algorithms on directed social networks. Voudigari et al. [115] proposed a sampling algorithm on social networks. Several sampling methods for triangle counting [116, 117, 118] have been proposed as well.

However, there has been no sampling algorithm that focuses on graphical inference. Most of the previous approaches sample subgraphs by selecting individual edges iteratively until the desired size is achieved, missing complex relationships between multiple nodes. On the other hand, we select cliques as minimal units to guarantee the bounded treewidth of subgraphs, maintaining

the comprehensive neighborhood of each node. This leads to improved accuracy of graphical inference on the generated subgraphs.

6.3 Proposed Method: BTW

We propose the bounded treewidth (BTW) sampling, a novel algorithm for generating subgraphs with bounded treewidth. BTW selects cliques as minimal units instead of edges to bound the treewidth, supporting tractable and accurate inference by the junction tree algorithm and preserving the context of each node.

Figure 6.2a shows a simple graph of 163 nodes with binary states, which is generated by slicing randomly a principal submatrix from the adjacency matrix of the PolBlogs network which we use in our experiments (Section 6.4.1). The two groups of nodes are separated clearly satisfying the property of homophily. BTW generates the subgraph in Figure 6.2d bounding its treewidth to 1, enabling accurate and efficient inference with only 39% of the edges.

6.3.1 Detailed Algorithm

The main objective of BTW is to guarantee the bounded treewidth of sampled subgraphs, which in turn leads to accurate and efficient inference. In other words, given a treewidth bound k, the treewidth of a subgraph U generated from BTW should be smaller than or equal to k. To achieve the objective, we keep track of a k-tree K along with U such that U is a subgraph of K. Thus, it is possible to bound its treewidth without exact computations.

BTW first selects $k+1$ nodes and creates a $(k+1)$-clique, which is an initial k-tree K, and creates the induced subgraph U from the same set of nodes. Then, BTW takes iterations of selecting a new node u by a score-based

Algorithm 6: Bounded treewidth (BTW) sampling

Input: a graph $G = (\mathcal{V}, \mathcal{E})$ and a bound k
Output: a sampled subgraph U and a k-tree K
1: $K, U, H \leftarrow \text{initialize}(G, k)$
2: **while** $|\mathcal{V}_U| < |\mathcal{V}|$ **do**
3: $\quad u, \mathcal{C} \leftarrow \text{next\_node}(H)$
4: $\quad U \leftarrow (\mathcal{V}_U \cup \{u\}, \mathcal{E}_U \cup \{(u, v) | v \in \mathcal{N}_G(u) \cap \mathcal{C}\})$
5: $\quad K \leftarrow (\mathcal{V}_K \cup \{u\}, \mathcal{E}_K \cup \{(u, v) | v \in \mathcal{C}\})$
6: $\quad H \leftarrow \text{update\_heap}(H, u, \mathcal{C})$
7: **end while**
8: **return** U, K

approach: given a score function $m(v, \mathcal{C})$ between a candidate node v and a clique $\mathcal{C}$ in K, it selects the node with the maximum score. BTW adds the selected node u to both K and U until all nodes are included in U.

Algorithm 6 describes the whole process of BTW. It initializes a k-tree K, a subgraph U, and a max-heap H in line 1. The max-heap H is introduced to speed up the process and is discussed in Section 6.3.2. Then, BTW adds a new node u iteratively until all nodes are included in U, in lines 2 to 7. At each iteration, it selects u in line 3 and then updates U, K, and H in lines 4 to 6. BTW returns K along with U as a result in line 8, since K is a junction tree that is needed when running the junction tree algorithm over U.

Score function The simplest choice of the score function m is the uniform random function:

$$m(v, \mathcal{C}) = \text{uniform}(0, 1). \tag{6.1}$$

Since it scores randomly all candidate nodes, using this function equals to first ordering all nodes randomly and then adding one node at a time following the order. However, it generates a sparse subgraph that maintains very few edges,

because it does not take into account the structure of the original graph G.

Our main objective is to generate subgraphs that preserve the properties of G to support efficient graphical inference. Thus, we propose the following score function that is designed to maximize the number of edges in the sampled subgraphs:

$$m(v, \mathcal{C}) = \sum_{x \in \mathcal{C} \cap \mathcal{N}_G(v)} \left(w(v, x) + \text{uniform}(0, \alpha) \right), \tag{6.2}$$

where $\mathcal{N}_G(u)$ is the set of neighbors of u in G, α is a small constant, and $w(v, x)$ is a positive weight of the edge (v, x).

This score function maximizes greedily the sum of edge weights in the subgraph U at every iteration. It simply maximizes the number of edges if w is a constant function, while one can design w to treat the edges differently. The uniform random function with the parameter α gives a randomness to the sampling process when the scores of multiple nodes are similar. For our experiments, we set w to a constant function as we deal with simple graphs, and α to a small value 10^{-4} to use it as a tie-breaker.

Subgraph generation BTW chooses randomly the first variable $u_1 \in \mathcal{V}$ and initializes the set $\mathcal{I}$ of sampled nodes as follows:

$$\mathcal{I} = \{u_1\}. \tag{6.3}$$

Then, until $\mathcal{I}$ contains $k + 1$ variables we add to it a new node $u_i \in \mathcal{V} \setminus \mathcal{I}$ that maximizes the score as in Equation (6.4). Since we initially do not have a k-tree, we use $\mathcal{I}$ instead of a clique: the score function m can be used generally

with a node set.

$$u_i = \arg\max_{v \in \mathcal{V} \setminus \mathcal{I}} m(v, \mathcal{I}). \tag{6.4}$$

After that, BTW generates a complete graph K_{k+1} over the sampled $k+1$ nodes, which is a k-tree. BTW generates also the induced subgraph U_{k+1} by selecting all the original edges whose incident nodes are both in $\mathcal{I}$. U_{k+1} is the initial subgraph which we update in the following iterations, and its treewidth is at most k since it is a partial k-tree; it is a subgraph of K_{k+1}.

Then, BTW takes iterations of adding a subsequent node u_{i+1} to the current subgraph U_i and k-tree K_i of i nodes, generating new graphs U_{i+1} and K_{i+1} of $i+1$ nodes. As in the initialization, BTW selects a node that maximizes the score function m as follows:

$$u_{i+1}, \mathcal{C}_k^* = \arg\max_{v, \mathcal{C}_k \in K_i} m(v, \mathcal{C}_k), \tag{6.5}$$

where $\mathcal{C}_k$ is a k-clique contained in K_i.

Given u_{i+1} and $\mathcal{C}_k^*$, BTW first updates K_i by connecting u_{i+1} to $\mathcal{C}_k^*$. This is the same as adding a new clique $\mathcal{C}_k^* \cup \{u_{i+1}\}$ to K_i by considering a clique as a minimal sampling unit. However, it is not possible to add the same clique to U_i since some of the edges in the clique may not be included in the original graph G. Thus, we instead connect u_{i+1} only to $\mathcal{N}_G(u_{i+1}) \cap \mathcal{C}_k^*$ to guarantee that U is a subgraph of G. Note that K is not necessarily a subgraph of G as it is used only for bounding the treewidth of U. We repeat the iteration until all nodes are sampled and return K and U.

Number of sampled edges BTW aims to maximize greedily the number of edges in U while bounding its treewidth, using the score function of Equation (6.2). The function counts the number of edges that are added to U if u is connected to $\mathcal{C}$. Thus, the number of edges in U_{i+1} is maximized as we find the maximum score for all pairs of nodes and cliques. The number of sampled edges will be much less if the score function of Equation (6.1) is used.

It is possible to further limit the number of edges in U with any sampling algorithm, since U already has bounded treewidth. On the other hand, it is very difficult to increase the number of sampled edges, since it is likely that more edges lead to increased treewidth. Recall that it takes exponential time even to simply compute the treewidth of a graph. Thus, we leave it as a future work to increase the number of sampled edges from our subgraph U. It can be done from designing a new score function or by implementing a whole new algorithm coupled with more advanced techniques.

6.3.2 Further Optimization

It is redundant to compute the scores of all candidate nodes at each iteration since the scores of most nodes remain unchanged. Thus, we store in a max-heap the maximum score $s(v) = \max_{\mathcal{C} \in K} m(v, \mathcal{C})$ of every candidate node v and optimize the algorithm by selecting the next node from the max-heap.

We initialize $s(v)$ of every node v as zero. Then, at each iteration where a new clique $\mathcal{C}_{k+1}$ is added to the current k-tree K, we update the score $s(v)$ of every candidate node v as follows:

$$s(v) \leftarrow \max \left\{ s(v), \max_{\mathcal{C}_k \subset \mathcal{C}_{k+1}} m(v, \mathcal{C}_k) \right\}. \tag{6.6}$$

If $s(v)$ is updated, we store also the clique $\mathcal{C}_k$ with the maximum score since it is needed when adding v to the subgraph.

A naive approach is to update the scores of all nodes neighboring in $\mathcal{C}_{k+1}$, since such nodes' scores with regard to $\mathcal{C}_{k+1}$ are greater than zero. The new scores need to be compared with their previous maximum scores. To expedite the update process, our optimization updates the scores of *only the neighbors of a new node* u which has been added to the current subgraph U. This results in the same update as in the naive approach, but is more efficient in orders of magnitude: it makes BTW run in linear time with the number of edges and thus scalable to large graphs.

Lemma 12 (Optimization)**.** *When a new node u and a clique $\mathcal{C}_{k+1}$ of $k+1$ nodes are added to the current subgraph U and k-tree K, respectively, the score $m(v, \mathcal{C}_k)$ of a node $v \notin \mathcal{N}_G(u)$ and a clique $\mathcal{C}_k \subset \mathcal{C}_{k+1}$ is less than or equal to the current score $s(v)$.*

Proof. We assume that α in Equation (6.2) is used only as a tie-breaker. The following holds because we assume $v \notin \mathcal{N}_G(u)$:

$$\max_{\mathcal{C}_k \subset \mathcal{C}_{k+1}} m(v, \mathcal{C}_k) = m(v, \mathcal{C}_{k+1} \setminus \{u\}). \tag{6.7}$$

Then, Equation (6.6) for computing the new maximum score $s(v)$ of node v is given as follows with regard to $\mathcal{C}_{k+1}$:

$$s(v) \leftarrow \max\left\{s(v), m(v, \mathcal{C}_{k+1} \setminus \{u\})\right\}. \tag{6.8}$$

Recall that $\mathcal{C}_{k+1}$ is formed by connecting u to a k-clique $\mathcal{C}_k = \mathcal{C}_{k+1} \setminus \{u\}$,

existing already in K. Since all cliques in K have been used already to update the score of every candidate node which is not included in the current K, $\mathcal{C}_k$ has also been considered for updating $s(v)$ before we add u to K in this step. Thus, Equation (6.8) is a redundant operation that does not change $s(v)$. □

6.3.3 Theoretical Analysis

We further present theoretical analyses of BTW about its time complexity and preservation of connectivity. Lemma 13 shows that BTW has a linear scalability with the number of edges of the given graph. The effect of k^2 is negligible since we normally use a small treewidth bound k such as 2 or 4. Lemma 16 shows that BTW preserves the connectivity of the given graph in sampled subgraphs, which is especially useful for graphical inference, because disconnected nodes are considered independent from each other.

Lemma 13 (Time complexity)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E})$ and a bound k, the time complexity of BTW is $O(|\mathcal{E}|(k^2 + \log|\mathcal{V}|))$ with the optimization.*

Proof. BTW selects a new node u at each iteration. The number of heap updates after selecting u is $O(|\mathcal{N}_G(u)|)$, and each update is $O(\log|\mathcal{V}|)$. The computation of Equation (6.6) to update each score is $O(k^2)$. Thus, the complexity of all heap updates is given as

$$O(\sum_{u\in\mathcal{V}} |\mathcal{N}_G(u)|(k^2 + \log|\mathcal{V}|)) = O(|\mathcal{E}|(k^2 + \log|\mathcal{V}|)).$$

The computational cost for heap deletions is safely ignored with a natural assumption of $|\mathcal{V}| \leq |\mathcal{E}|$, since it is $O(|\mathcal{V}|\log|\mathcal{V}|)$. □

Lemma 14 (Connectivity preservation)**.** *When a given graph G is connected,*

all intermediate subgraphs during the iterations of BTW are connected, including the one to be returned.

Proof. We prove the lemma by mathematical induction:

1. **Base case.** The initial nodes in $\mathcal{I}$ are guaranteed to be connected because every node is selected for maximizing the number of edges of the subgraph U when added to $\mathcal{I}$.

2. **Inductive step.** We assume that the subgraph U_i of i nodes is connected. Then, the next subgraph U_{i+1} with an additional node u_{i+1} becomes disconnected only if there exists no edge between U_i and the rest of G, since u_{i+1} has been selected for its maximum score. However, that cannot happen since G is connected. Thus, U_{i+1} is connected.

Thus, every subgraph generated by BTW is connected. □

6.4 Experiments

We present experimental results to answer the following questions:

Q1. **Accuracy and inference time (Section 6.4.2).** What are the accuracy and inference time on the subgraphs from BTW compared with those on the original graphs?

Q2. **Comparison with other sampling algorithms (Section 6.4.3).** Is BTW more suitable for the graphical inference compared with the other sampling algorithms?

Q3. **Optimization of BTW (Section 6.4.4).** How much speedup can be

Table 6.1: A summary of four real-world networks that we use in our experiments. All datasets are publicly available.

Network	Domain	Nodes	Edges	Density	Labels
Wikipedia[1]	Web	35,579	495,357	0.0391%	16
CoRA[1]	Citation	23,567	91,965	0.0167%	10
PubMed[2]	Citation	19,717	44,324	0.0114%	3
PolBlogs[3]	Web	1,222	16,714	1.1193%	2

[1] `https://github.com/sharadnandanwar/snbc`
[2] `https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz`
[3] `http://www-personal.umich.edu/~mejn/netdata/polblogs.zip`

obtained by the optimization of BTW? How does it scale with the number of edges and the treewidth k?

6.4.1 Experimental Settings

In this section, we present experimental settings: datasets, competitors, etc. Our experiments were done by a workstation with Intel Xeon E5-2630 2.20GHz CPU and 50GB memory.

Datasets We use four real-world networks summarized in Table 6.1. Wikipedia is a crawled dump of Wikipedia pages from 16 categories [119]. The nodes represent pages of certain categories, and the edges represent hyperlinks. CoRA and PubMed are citation networks whose nodes represent research articles, and edges represent citations [55, 119]. Each node is labeled according to the research topic. PolBlogs is a webgraph whose nodes represent political blogs, which are liberal or conservative, and edges represent hyperlinks [120].

We remove the edge directions from directed graphs as in [9] to model symmetric relationships between variables. We also remove the isolated compo-

nents since they are independent in terms of inference. The numbers of nodes and edges in Table 6.1 are counted after these preprocessing steps.

Graph sampling algorithms We compare BTW with other graph sampling algorithms. We focus on edge sampling approaches that sample only edges, since every node is a target variable which we aim to classify.

Random edge (RE) sampling is the simplest algorithm that samples repeatedly an edge uniformly at random. However, it selects most edges from high-degree nodes, ignoring the importance of low-degree nodes. Random node-edge (RNE) sampling solves the problem by selecting a node uniformly at random and sampling one of its adjacent edges. Hybrid (HYB) approach [121] combines the two approaches: it performs either a step of RE or RNE randomly with a probability p at each iteration. p is set to 0.8 as in [121].

Random walk (RW) sampling uniformly at random picks a node and then simulates a random walk with restart. It selects all edges that its random walker passes through. Random jump (RJ) sampling is similar to RW, but the random walker restarts randomly in any node instead of the starting one. Frontier sampling (FS) [122] uses multiple random walkers simultaneously to generate more stable subgraphs. Forest fire (FF) sampling [113] selects n neighbors instead of one at each step, where n follows a probability distribution.

Because none of these methods guarantees to keep the original connectivity in subgraphs, we propose two additional algorithms BTW-W and BTW-J by modifying RW and RJ, respectively. They guarantee the connectivity of a sampled subgraph by generating a spanning tree of nodes using BTW of $k = 1$ and then selecting the remaining edges by their original algorithms (RW and

RJ). We use them to demonstrate that the superior performance of BTW comes from not only its ability to keep the connectivity, but also its idea of sampling cliques as minimal units instead of edges.

Node classification by inference The real-world networks in Table 6.1 are not graphical models and thus contain no probabilistic information. For solving node classification by graphical inference, we model each dataset as a pairwise Markov random field (MRF), an undirected graphical model that has been widely used for node classification [52]. Compared with a standard MRF that requires a potential for every clique existing in the graph, a pairwise MRF uses only pairwise potentials, which we call *edge potentials*.

We generate an edge potential matrix ψ of size $n \times n$ based on the observed evidence of each dataset, where n is the number of states. First, we initialize ψ as a zero matrix. Then, for each connected pair of observed nodes in the graph, we add one to the corresponding element of ϕ based on their labels. After that, we normalize ψ by the sum of its elements to make it represent a probability distribution. The resulting potentials are shown to follow the homophily (details in Section 6.6). However, we have found that LBP does not converge with the computed potentials since they are too skewed. For this reason, we introduce a new parameter d that alleviates the skewness, and use it in all experiments where LBP is used (details in Section 6.7). We use the original potentials when applying the junction tree algorithm since it has no convergence issue.

For each network and sampling algorithm, we generate 10 subgraphs with different random seeds. Then, we adopt the 3-fold cross validation for each subgraph: we hide the labels of 1/3 of the nodes and predict them by run-

Table 6.2: The average numbers of edges in subgraphs generated from BTW, with respect to the treewidth bound k.

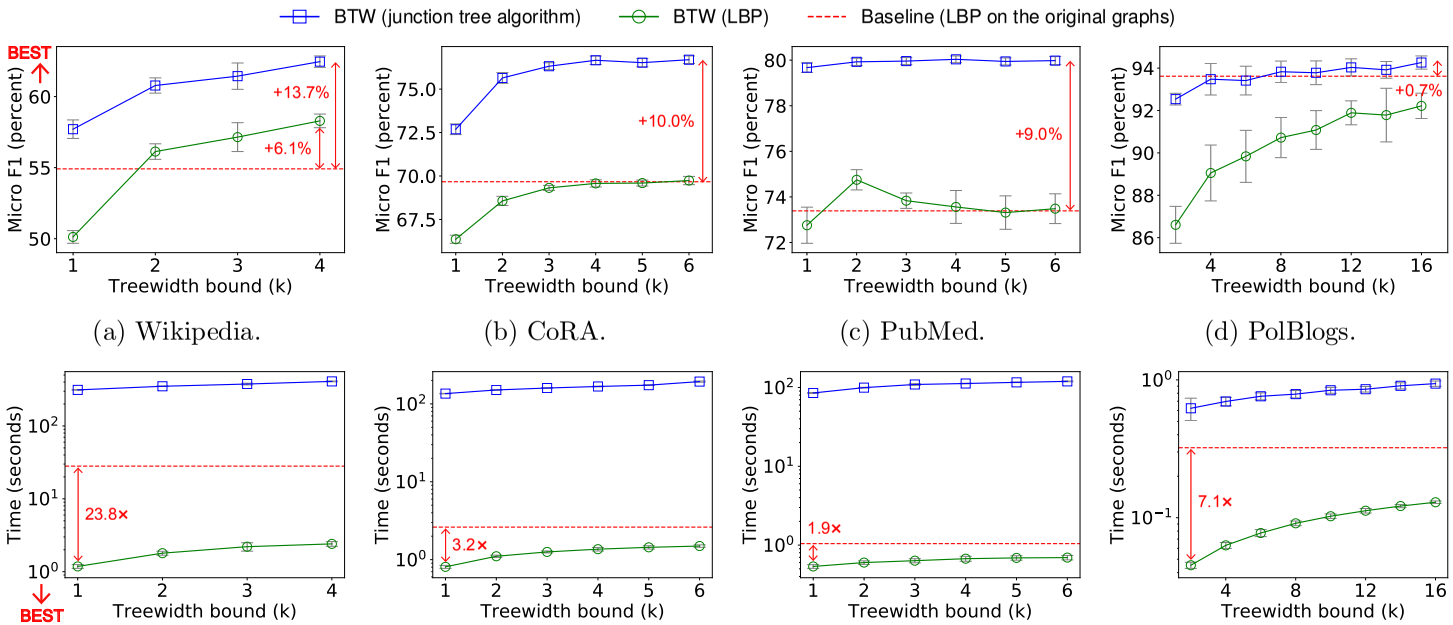
Network	$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$
Wikipedia	35,578	57,896	84,213	126,234	165,620
Cora	23,566	36,307	46,158	51,310	54,479
PubMed	19,716	23,109	25,169	26,641	27,619
PolBlogs	1,221	2,004	3,086	4,477	6,374

ning graphical inference using the rest as evidence. We give the maximum-a-posteriori (MAP) assignments as predicted labels. We evaluate the result using micro and macro F1 scores as done in [119] but report only the micro F1 due to the lack of space. We report the averages in all experiments.

6.4.2 Accuracy and Inference Time

Figure 6.3 shows how the classification accuracy and inference time change if we run the inference algorithms on the subgraphs from BTW instead of the original graphs. The junction tree algorithm achieves up to 13.7% higher accuracy than the baseline (LBP on the original graph), showing consistent improvements in most cases; the exception is PolBlogs with small k, but the accuracy improves with k and outperforms the baseline from $k = 8$. Although the inference is slower than the baseline due to the overhead of computing the joint probability of each clique by the junction tree algorithm, the inference is still tractable because BTW guarantees to generate subgraphs with bounded treewidth.

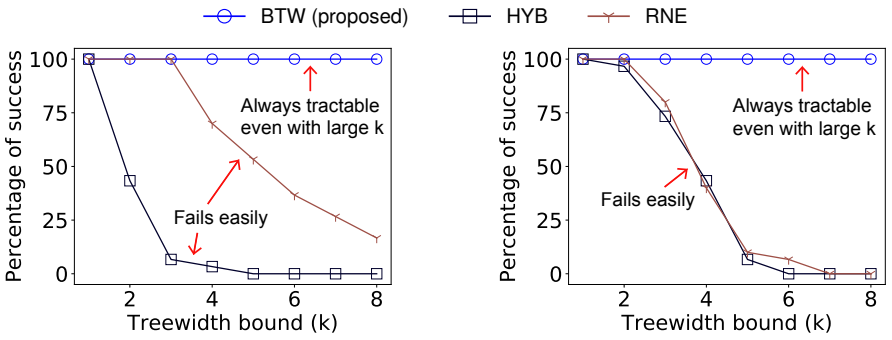
The accuracy of LBP on the subgraphs is generally similar to the baseline. This implies that BTW successfully captures essential relationships between



(a) Wikipedia. (b) CoRA. (c) PubMed. (d) PolBlogs.

(e) Wikipedia. (f) CoRA. (g) PubMed. (h) PolBlogs.

Figure 6.3: Micro F1 of node classification and inference time by the junction tree algorithm (the solid blue lines) and LBP (the solid green lines) on the subgraphs sampled by BTW, and the baseline (LBP on the original graph, shown by the red dashed lines). BTW with the junction tree algorithm achieves the highest accuracy, despite its slow inference time. On the other hand, BTW with LBP shows comparable accuracy to the baseline with much faster inference.



(a) PubMed. (b) PolBlogs.

Figure 6.4: Ratios of successful inference (no out of memory) of the junction tree algorithm for the subgraphs from BTW and major competitors (HYB and RNE). BTW maintains the success ratio of 100% regardless of k, while the others easily fall into memory shortages as k increases.

nodes, and thus LBP can recover the original accuracy with fewer edges: the subgraphs of Wikipedia when $k = 2$ contain only 11.7% of the original edges as shown in Table 6.2. At the same time, LBP on the subgraphs is consistently faster than the baseline from 1.5× to 23.8× based on the value of k.

We use small treewidth bounds in CoRA and Wikipedia (up to 6 and 4, respectively) in order to keep the junction tree algorithm tractable, which requires large memory of $O(n^k)$, where n represents the number of states [6]. Recall that n in both graphs are much larger than in the other graphs. However even in this case BTW yields an important improvement over the baseline.

6.4.3 Comparison with Other Approaches

We compare BTW and the other graph sampling algorithms by the classification accuracy on generated subgraphs. Most competitors require to know in advance the number of edges that the subgraph should contain, while BTW finds the

Table 6.3: Micro F1 scores of LBP on the subgraphs from BTW and the competitors when the treewidth bound k is 2. BTW shows the best accuracy.

Method	**Wikipedia**	**CoRA**	**PubMed**	**PolBlogs**
RE	35.3 ± 0.2	57.9 ± 0.3	61.8 ± 0.4	75.8 ± 1.0
RNE	51.3 ± 0.3	65.2 ± 0.2	71.1 ± 0.1	84.4 ± 0.6
HYB	49.4 ± 0.2	64.5 ± 0.2	69.8 ± 0.3	83.4 ± 0.4
RW	26.5 ± 2.7	43.0 ± 1.7	56.5 ± 1.2	65.2 ± 3.4
RJ	36.6 ± 0.4	55.4 ± 0.3	63.2 ± 0.4	75.6 ± 0.5
FS	29.8 ± 0.2	47.9 ± 0.2	56.2 ± 0.5	72.4 ± 0.8
FF	49.4 ± 0.2	63.7 ± 0.3	62.8 ± 0.4	79.8 ± 0.9
BTW-W	50.9 ± 1.4	66.8 ± 2.4	69.7 ± 0.7	82.4 ± 1.7
BTW-J	53.0 ± 0.3	67.5 ± 0.2	73.6 ± 0.8	85.0 ± 1.3
BTW	**56.1 ± 0.5**	**68.6 ± 0.3**	**74.8 ± 0.4**	**86.6 ± 0.9**

subgraph automatically once k is specified. Thus, we first run BTW with a given bound k, record the number of edges that are present in the subgraph, and let the others generate subgraphs with the same numbers of edges.

Recall that the time and space complexities of the junction tree algorithm are both exponential with k. Usually, it is not possible to run exact inference on subgraphs generated without any control of treewidth, since it causes the junction tree algorithm to fail with an out-of-memory error. The percentages of successful inference (no out-of-memory) on PubMed and PolBlogs are shown in Figure 6.4. They decrease sharply for all methods other than BTW, especially when $k > 3$, while the inference is always successful on the subgraphs returned by BTW due to the bounded treewidth.

We then perform node classification on such subgraphs reverting to LBP and obtain the accuracy in Table 6.3. BTW consistently shows the best accuracy compared with all competitors. It is notable that BTW performs better than BTW-J and BTW-W that keep the original connectivity in subgraphs

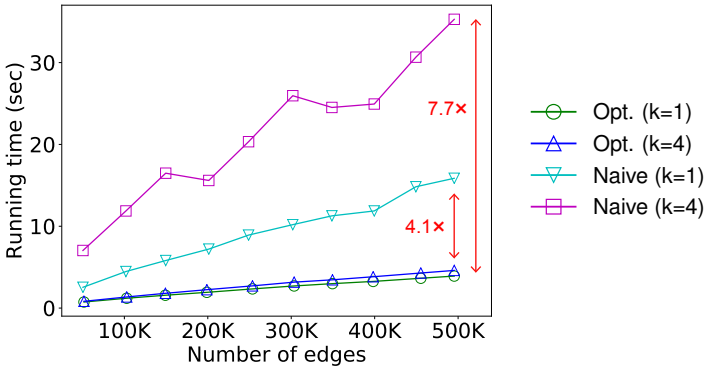
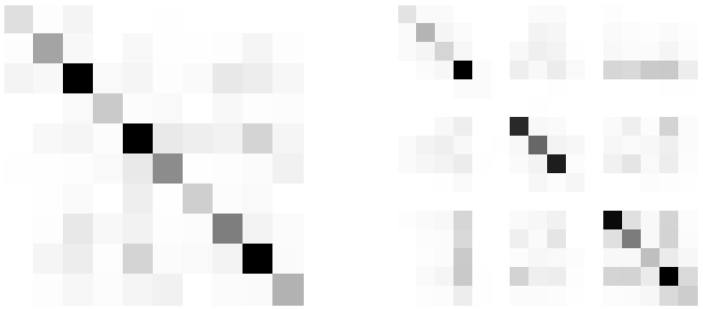


Figure 6.5: Sampling time of BTW on graphs of different numbers of edges and treewidth bound k. *Opt.* and *Naive* represent BTW with and without the optimization, respectively. BTW runs up to 7.7 $\times$ faster with the optimization, showing near-linear scalability as presented in Lemma 13.

by running BTW of $k = 1$ before RW and RJ. This implies that the superior performance of BTW does not rely on its property of keeping the connectivity, but on its main ideas of bounding the treewidth and providing each node a chance of having a neighborhood of size k in sampled subgraphs: every node keeps essential relationships with the others.

6.4.4 Optimization of BTW

Figure 6.5 shows the running time of BTW on graphs with different numbers of edges, with and without the optimization in Lemma 12. We have sampled nine principle submatrices from the adjacency matrix of Wikipedia for creating graphs of different sizes. BTW shows near-linear scalability with the number of edges as discussed in Lemma 13. It also shows the effect of optimization: introducing it makes BTW up to 7.7 times faster than before. Thus, BTW can be used efficiently for running inference in large graphs.



(a) CoRA (b) Wikipedia

Figure 6.6: The potential matrices of CoRA and Wikipedia. The nodes in those graphs have the homophily as the diagonal elements are significantly larger than the others.

6.5 Summary

In this chapter, we propose the bounded treewidth (BTW) sampling, a novel graph sampling algorithm that generates subgraphs with guaranteed bounded treewidth. BTW samples as many edges as possible while bounding the treewidth, selecting cliques as minimal units of sampling and thus preserving the comprehensive neighborhood of each node. The subgraphs generated from BTW make the junction tree algorithm tractable by the bounded treewidth, which shows up to 13.7% higher micro F1 in node classification than loopy belief propagation (LBP) does on the original graph. Moreover, BTW speeds up LBP up to 23.8 times with accuracy comparable to that on the original graph, and consistently higher than those from the other sampling algorithms. We expect that subgraphs generated from BTW are suited to other applications which can exploit the low treewidth. Future works include extending BTW to heterogeneous networks by introducing new score functions that consider various types of nodes, edges, and additional attributes.

6.6 Appendix: Potential Matrices

The following matrices ψ_B and ψ_M show the edge potentials used for PolBlogs and PubMed, respectively, that are generated from the process of modeling MRFs in Section 6.4.1:

$$\psi_B = \begin{bmatrix} 0.437 & 0.047 \\ 0.047 & 0.469 \end{bmatrix}, \qquad \psi_M = \begin{bmatrix} 0.118 & 0.042 & 0.019 \\ 0.042 & 0.356 & 0.038 \\ 0.019 & 0.038 & 0.329 \end{bmatrix}.$$

Figure 6.6 shows the potentials of CoRA and Wikipedia by representing the values as colors (the larger, the darker). The diagonal elements in all these matrices are significantly larger than the others, implying that the nodes are positively correlated and follow the assumption of homophily in social networks.

6.7 Appendix: Convergence of LBP

An exact condition that LBP converges has not been presented in the literature. Thus, we empirically show that LBP converges more often when the potential matrix is less skewed. We introduce a new parameter d which controls the skewness of a potential: a potential matrix ψ is replaced by a weighted average $\frac{1}{d}(\psi + (d-1)M)$ with a constant matrix M, whose elements are uniform and sum to one. Thus, the potentials become less skewed as d increases.

For each dataset, we run LBP ten times with different sets of observed nodes and show in Figure 6.7 the percentage of runs that converge. We conclude it as a convergence when the maximum difference between the messages in two consecutive iterations is within 10^{-4} before 1000 iterations. As a result, LBP

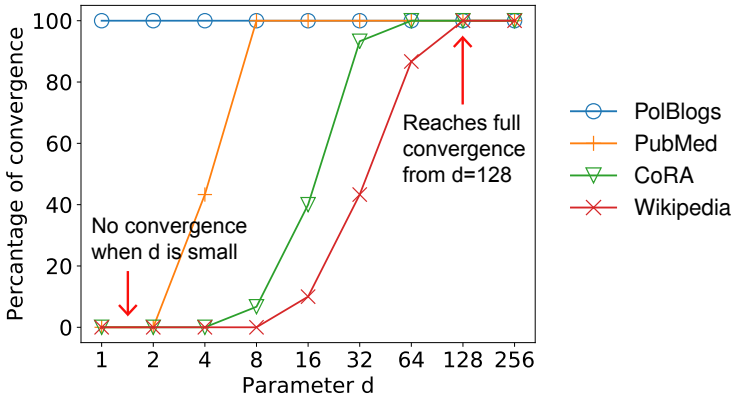


Figure 6.7: Percentages of runs that LBP converges by different values of d. LBP hardly converges with small d on three of the four datasets, and the convergence ratios increase as d increases: all ratios become 100% if $d \geq 128$.

hardly converges in three of the four datasets when d is a small value such as 2 or 4. The convergence seems especially difficult on CoRA and Wikipedia, which contain larger numbers of labels compared with the other datasets. The convergence ratios become 100% for all datasets when d is at least 128.

Based on this result, we set d to 256 when LBP is used, to guarantee a full convergence. On the contrary, it is not necessary to use d when the junction tree algorithm is used because the convergence is guaranteed: it stops after one iteration of fully propagating the observed information. This is an important advantage of running exact inference on subgraphs from BTW because it is not necessary to modify the relationships arbitrarily, which leads to different marginals from the ones that can be inferred from a dataset.

Chapter 7

Graph Augmentation for Classification

Given a graph dataset, how can we augment it for accurate graph classification? Graph augmentation is an essential strategy to improve the performance of graph-based tasks by enlarging the distribution of training data. However, previous works for graph augmentation either a) involve the target model in the process of augmentation, losing the generalizability to other tasks, or b) rely on simple heuristics that lead to unreliable results. In this chapter, we introduce five desired properties for effective augmentation. Then, we propose NodeSam (Node Split and Merge) and SubMix (Subgraph Mix), two model-agnostic approaches for graph augmentation that satisfy all desired properties with different motivations. NodeSam makes a balanced change of the graph structure to minimize the risk of semantic change, while SubMix mixes random subgraphs of multiple graphs to create rich soft labels combining the evidence for different classes. Our experiments on seven benchmark datasets show that NodeSam and SubMix consistently outperform existing approaches, making the highest accuracy in graph classification.

7.1 Motivation

How can we augment graphs for accurate graph classification? Data augmentation is an essential strategy to maximize the performance of estimators by enlarging the distribution covered by training data. The technique has been used widely in various data domains such as images [123], time series [124], and language processing [125]. The problem of graph augmentation has also attracted wide attention for graph-related tasks such as graph classification [126] with the advancement of graph neural networks (GNN) [127, 38, 83].

Previous approaches on graph augmentation are categorized to model-specific [128, 129] and model-agnostic ones [130, 126, 131]. Model-specific approaches often make a better performance than model-agnostic ones, because they are designed for specific target models. However, their performance is not generalized to other settings of models and problems, and a careful tuning of hyperparameters is required even with a small change of experimental setups. On the other hand, model-agnostic approaches work generally well with various models and problems, even though their best performance can be worse than that of model-specific ones. Figure 7.1 illustrates how model-agnostic augmentation works for a graph classifier f that classifies each graph into the red or the blue class.

However, previous works on model-agnostic augmentation [130, 126, 131] rely on simple heuristics such as removing random edges or changing node attributes rather than carefully designed operations. Such heuristics provide no theoretical guarantee for essential properties of data augmentation such as the unbiasedness or linear scalability. As a result, previous approaches often make unreliable results, losing the main advantage over model-specific approaches.

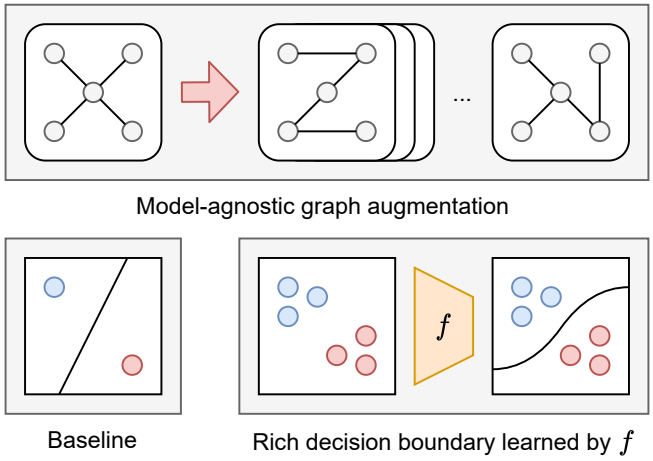


Figure 7.1: Illustration of how model-agnostic augmentation works. Each circle at the bottom represents a graph, whose label is represented as color. The augmented graphs allow a classifier f to learn a rich decision boundary.

Moreover, our experiments show that they often decrease the accuracy of target models in graph classification, where the structural characteristic of each graph plays an essential role for predicting its label (details are in Section 7.4).

In this chapter, we first propose five properties that an augmentation algorithm should satisfy to maximize its effectiveness. These properties are designed carefully to include the degree of augmentation, the preservation of graph size, and the scalability to large graphs. We then propose two novel algorithms, NodeSam and SubMix, which satisfy all these properties. NodeSam (Node Split and Merge) performs split and merge operations on nodes, minimizing the degree of structural change while augmenting both the node- and edge-level information. SubMix (Subgraph Mix) combines multiple graphs by swapping random subgraphs to maximize the degree of augmentation, generating rich soft labels for classification.

Our contributions are summarized as follows:

- **Objective formulation.** We propose desired properties that are essential for effective graph augmentation, providing a clear objective for augmentation algorithms.
- **Algorithms.** We propose NodeSam and SubMix, effective model-agnostic algorithms for graph augmentation. NodeSam is a stable and balanced approach that makes a minimal change of the graph structure, while SubMix generates more diverse samples through abundant augmentation.
- **Theory.** We theoretically analyze the characteristics of our proposed approaches and demonstrate that they satisfy all the desired properties even in the worst cases.
- **Experiments.** We perform experiments on seven datasets to show the effectiveness of our methods for improving graph classifiers. Our methods make up to 2.1× larger improvement of accuracy compared with the best competitors.

The rest of this chapter is organized as follows. In Section 7.2, we define the problem of graph augmentation and present the desired properties. In Section 7.3, we propose our NodeSam and SubMix and discuss their theoretical properties. We show experimental results in Section 7.4 and introduce related works in Section 7.5. We summarize in Section 7.6.

7.2 Problem and Desired Properties

We formally define the graph augmentation problem and present desired properties for effective augmentation. Table 7.1 compares various methods for graph

Table 7.1: Comparison between various approaches for graph augmentation with respect to desired properties. Pi refers to Property i (see Section 7.2.2). Our proposed methods satisfy all the desired properties, while the baselines do not.

Method	**P1**	**P2**	**P3**	**P4**	**P5**
DropEdge [132]				✓	✓
GraphCrop [126]			✓	✓	✓
NodeAug [130]			✓	✓	✓
MotifSwap [131]	✓	✓			
NodeSam (proposed)	✓	✓	✓	✓	✓
SubMix (proposed)	✓	✓	✓	✓	✓

augmentation regarding the desired properties that we propose in this section.

7.2.1 Problem Definition

Given a set of graphs, graph augmentation is to generate a new set of graphs that have similar characteristics to the given graphs. We give the formal definition as Problem 3.

Problem 3 (Graph augmentation)**.** *We have a set $\mathcal{G}$ of graphs. Each graph $G \in \mathcal{G}$ consists of a set $\mathcal{V}$ of nodes, a set $\mathcal{E}$ of edges, and a feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where d is the number of features. Then, the problem is to make a set $\bar{\mathcal{G}}$ of new graphs that are more suitable than $\mathcal{G}$ for the training of a model f, improving its performance.*

Although any task can benefit from graph augmentation, we use graph classification as the target task to solve by a classifier f. This is because graph classification is more sensitive to the quality of augmentation than node-level tasks are, such as node classification or link prediction. In graph classification, naive augmentation can easily decrease the accuracy of f if it changes the char-

acteristic of a graph that is essential for its classification (details are in Section 7.4). On the other hand, in node-level tasks such as node classification, even a simple heuristic algorithm can improve the accuracy of models by changing the local neighborhood of each target node [130, 132]. Thus, the accuracy of graph classification is a suitable measure for comparing different approaches for graph augmentation.

7.2.2 Desired Properties

Our goal is to generate a set of augmented graphs that maximize the performance of a graph classifier f as presented in Problem 3. The main difficulty of augmentation is that the *semantic information* of a graph, which means the unique characteristic that determines its label, is not given clearly. For example, in the classification task of molecular graphs, it is difficult even for domain experts to check whether an augmented graph has the same chemical property as in the original graph. This makes it difficult for an augmentation algorithm to safely enlarge the data distribution.

We propose five desired properties for an effective augmentation algorithm to maximize the degree of augmentation while minimizing the risk of changing semantic information. Property 1 and 2 are for preserving the basic structural information of a graph in terms of the size and connectivity, respectively.

Property 1 (Preserving size)**.** *Given a graph* $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ *and an augmentation function* h*, let* $\bar{G} = h(G)$*. Then,* h *should make an unbiased change of the graph size with* $\mathbb{E}[|\bar{\mathcal{V}}| - |\mathcal{V}|] = 0$ *and* $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$*, where* $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$*.*

Property 2 (Preserving connectivity)**.** *Given a graph* $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ *and an augmentation function* h*, let* $\bar{G} = h(G)$*. Then,* $\bar{G}$ *should follow the connectivity*

information of G, i.e., $\bar{G}$ should be connected if and only if G is connected.

At the same time, it is necessary for an augmentation algorithm to make meaningful changes to the given graph; Property 1 and 2 are satisfied even with the identity function. In this regard, we introduce Property 3 and 4 that force an augmentation algorithm to make meaningful changes.

Property 3 (Changing nodes)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, h should make a change of nodes in $\mathcal{V}$ by satisfying either $\mathbb{E}[(|\bar{\mathcal{V}}| - |\mathcal{V}|)^2] > 0$ or $\mathbb{E}[\|\bar{\mathbf{X}} - \mathbf{X}\|_{\mathrm{F}}^2] > 0$, where $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$, and $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm of a matrix.*

Property 4 (Changing edges)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, h should make a change of $|\mathcal{E}|$, i.e., $\mathbb{E}[(|\bar{\mathcal{E}}| - |\mathcal{E}|)^2] > 0$, where $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$.*

Lastly, we require the augmentation to be done in linear time with the graph size to support scalability in large graphs, which is essential for real-world applications [78, 95].

Property 5 (Linear complexity)**.** *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and an augmentation function h, let $\bar{G} = h(G)$. Then, the time and space complexities of h for generating $\bar{G}$ should be $O(d|\mathcal{V}| + |\mathcal{E}|)$, where d is the number of features, i.e., $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$.*

We briefly review the previous approaches in Table 7.1 in terms of the desired properties. DropEdge [132] and GraphCrop [126] make a subgraph of the given graph as a result of augmentation. This makes a high risk of semantic change, as we have no clue for the essential part that determines the characteristic of the graph. NodeAug [130] also changes the graph properties by adding

Algorithm 7: NodeSam (Node Split and Merge)

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$
Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$
1: $G', v_i, v_j, v_k \leftarrow \text{Split}(G)$ `// Algorithm 8`
2: $G'' \leftarrow \text{Adjust}(G, G', v_i, v_j, v_k)$ `// Algorithm 10`
3: $\bar{G} \leftarrow \text{Merge}(G'')$ `// Algorithm 9`

and removing edges. MotifSwap [131] preserves the properties of the given graph with respect to both nodes and edges, but fails to make a sufficient amount of change. Moreover, MotifSwap is not scalable to large graphs, as its running time is not linear with the number of edges due to the global enumeration to find all open triangles existing in the graph.

7.3 Proposed Methods

In this chapter, we propose two effective algorithms for graph augmentation, which satisfy all the desired properties in Table 7.1. NodeSam (Node Split and Merge) performs opposite split and merge operations over nodes to make a balanced change of graph properties, while SubMix (Subgraph Mix) combines multiple graphs by mixing random subgraphs to enlarge the space of augmentation.

7.3.1 NodeSam: Node Split and Merge

The main idea of NodeSam is to perform opposite operations at once to make a balanced change of the graph properties. NodeSam first splits a random node into a pair of adjacent nodes, increasing the number of nodes by one. NodeSam then merges a random pair of nodes into a single node, making the generated

Algorithm 8: Split in NodeSam

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$

Output: Intermediate graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$, target node v_i, and generated nodes v_j and v_k

1: $v_i \leftarrow$ Select a node from $\mathcal{V}$ uniformly at random
2: $v_j, v_k \leftarrow$ Make new nodes to insert to G
3: $\mathbf{x}_j, \mathbf{x}_k \leftarrow$ Make new features such that $\mathbf{x}_i = \mathbf{x}_j = \mathbf{x}_k$
4: $h \leftarrow$ Make a function that randomly returns v_j or v_k
5: $\mathcal{V}' \leftarrow (\mathcal{V} \setminus \{v_i\}) \cup \{v_j, v_k\}$
6: $\mathcal{E}' \leftarrow \{(a, b) \mid (a, b) \in \mathcal{E} \wedge a \neq v_i \wedge b \neq v_i\} \cup \{(v_j, v_k)\}$
7: $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(h(a), b) \mid (a, b) \in \mathcal{E} \wedge a = v_i\}$
8: $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{(a, h(b)) \mid (a, b) \in \mathcal{E} \wedge b = v_i\}$
9: $\mathbf{X}' \leftarrow$ Remove $\mathbf{x}_i$ from $\mathbf{X}$, and add $\mathbf{x}_j$ and $\mathbf{x}_k$ to it

graph have the same number of nodes as in the original graph. The combination of these opposite operations allows it to change the node- and edge-level information at the same time while preserving the structure of the original graph such as the connectivity.

Algorithm 7 describes the process of NodeSam, where the split and merge operations are done at lines 1 and 3, respectively. The adjustment operation in line 2 is introduced to make an unbiased change of the number of edges by inserting additional edges to the graph. We first discuss the split and merge operations in detail and then present the adjustment operation in Section 7.3.1.

Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, NodeSam performs the split and merge operations following Algorithms 8 and 9, respectively. The split operation selects a random node v_i and splits it into nodes v_j and v_k, making an edge (v_j, v_k) and copying its feature as $\mathbf{x}_i = \mathbf{x}_j = \mathbf{x}_k$. The edges attached to v_i are split into v_j and v_k following the binomial distribution $\mathcal{B}(|\mathcal{N}_i|, 0.5)$, where $\mathcal{N}_i$ is the set of neighbors of v_i. The merge operation selects a random pair of adjacent nodes

Algorithm 9: Merge in NodeSam

Input: Graph $G'' = (\mathcal{V}'', \mathcal{E}'', \mathbf{X}'')$ generated from Adjust
Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$

1: $v_o, v_p \leftarrow$ Select adjacent nodes from $\mathcal{V}''$ uniformly at random
2: $v_q \leftarrow$ Make a new node to insert to G''
3: $\mathbf{x}_q \leftarrow$ Make a new feature such that $\mathbf{x}_q = (\mathbf{x}_o + \mathbf{x}_p)/2$
4: $\bar{\mathcal{V}} \leftarrow (\mathcal{V}'' \setminus \{v_o, v_p\}) \cup \{v_q\}$
5: $\bar{\mathcal{E}} \leftarrow$ Remove all edges from $\mathcal{E}''$ connected to either v_o or v_p
6: $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \cup \{(v_q, b) \mid (a, b) \in \mathcal{E}'' \wedge a \in \{v_o, v_p\}\}$
7: $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \cup \{(a, v_q) \mid (a, b) \in \mathcal{E}'' \wedge b \in \{v_o, v_p\}\}$
8: $\bar{\mathbf{X}} \leftarrow$ Remove $\mathbf{x}_o$ and $\mathbf{x}_p$ from $\mathbf{X}''$, and add $\mathbf{x}_q$ to it

v_o and v_p and merges them into a new single node v_q with a feature vector $\mathbf{x}_q = (\mathbf{x}_o + \mathbf{x}_p)/2$. The edges connected to either v_o or v_p are connected to v_q, while the edge (v_o, v_p) is removed.

Adjustment Operation

The basic version of NodeSam with only the split and merge operations have two limitations. First, the split operation weakens the relationships between the nodes in $\mathcal{N}_i$. That is, the number of common neighbors between any two nodes in $\mathcal{N}_i$ is likely to decrease in the graph G' generated from the split. This happens if v_i forms triangles with its neighbors, since the split eliminates these triangles by changing them into loops of length four. Second, the number of edges tends to decrease in augmented graphs, since the merge operation can remove more than one edge. This happens if there are triangles containing both of the target nodes v_o and v_p, since the other two edges except (v_o, v_p) in each triangle are combined into a single one.

To address the two limitations, we propose an adjustment operation of Algorithm 10 that inserts additional edges to nodes v_j and v_k, which are generated

from the split. First, we randomly select a subset $\mathcal{S}$ of nodes from $\mathcal{T}_i$, which is the set of all nodes that form triangles with v_i in the original graph G. Then, we add $|\mathcal{S}|$ edges to the graph by the following process: for each node $u \in \mathcal{S}$, we insert edge (u, s) to the graph, where s is v_j (or v_k) if u is connected with v_k (or v_j). Note that all nodes in $\mathcal{S}$ have an edge with either v_j or v_k before the adjustment since they are neighbors of v_i in G.

Figure 7.2 illustrates how NodeSam works in an example graph of seven nodes. NodeSam first splits a random node v_i into a pair of nodes v_j and v_k, decreasing the number of triangles from six to four. Then, the adjustment operation selects a random subset $\mathcal{S} = \{u\}$ of nodes from $\mathcal{T}_i$, which is $\mathcal{V} \setminus \{v_i\}$ in this example, and connects u with v_j. Lastly, the merge operation combines a random pair of nodes v_o and v_p into node v_q. Note that the numbers of edges and triangles of G are preserved in the augmented graph $\bar{G}$ even with a different structure due to edge (u, v_j) made by the adjustment.

The size of $\mathcal{S}$ is determined randomly in line 7 of Algorithm 10 following a binomial distribution whose mean is given as $\mathbb{E}[|\mathcal{S}|] = h_i$, where h_i is a number chosen to estimate the number of edges removed by the merge operation as follows:

$$h_i = \frac{1}{2}((c_i^2 + 4t_i|\mathcal{V}| - 6t_i)^{1/2} - c_i), \tag{7.1}$$

where t_i is the number of triangles containing v_i in G, d_i is the node degree of v_i in G, and $c_i = |\mathcal{E}| - 3t_i/d_i - 2$ (see Lemma 15).

Local estimation All variables that compose h_i in Equation (7.1) are computed from the direct neighborhood of v_i, except for $|\mathcal{V}|$ and $|\mathcal{E}|$ that are known in advance. This allows NodeSam to be run in linear time with the number

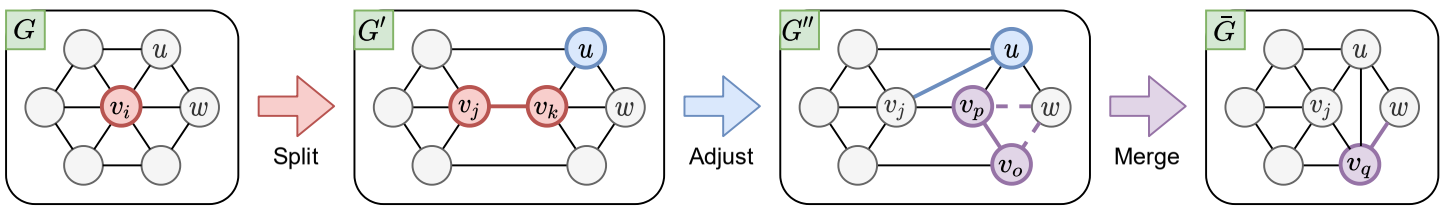


Figure 7.2: Illustration of how the three operations of NodeSam work in an example graph: Split, Adjust, and Merge. The adjustment operation preserves the numbers of edges and triangles of G by inserting an edge between nodes u and v_j.

Algorithm 10: Adjust in NodeSam

Input: Original graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$,
graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ generated from Split,
target node $v_i \in \mathcal{V}$ of Split, and
nodes $v_j \in \mathcal{V}'$ and $v_j \in \mathcal{V}'$ generated from Split

Output: Intermediate graph $G'' = (\mathcal{V}'', \mathcal{E}'', \mathbf{X}'')$

$t_i \leftarrow$ Count the number of triangles in G containing v_i
$d_i \leftarrow$ Get the degree of v_i in G
$c_i \leftarrow |\mathcal{E}| - 3t_i/d_i - 2$
$h_i \leftarrow ((c_i^2 + 4t_i|\mathcal{V}| - 6t_i)^{1/2} - c_i)/2$
$b \leftarrow$ Make a function that returns a random value in $[0, 1)$
$\mathcal{T}_i \leftarrow$ Take all nodes included in the triangles containing v_i
$\mathcal{S} \leftarrow \{u \mid (u \in \mathcal{T}_i \setminus \{v_i\}) \wedge (b(u) < h_i/(|\mathcal{T}_i| - 1))\}$
$\mathcal{V}'', \mathbf{X}'' \leftarrow \mathcal{V}', \mathbf{X}'$
$\mathcal{E}'' \leftarrow \mathcal{E}' \cup \{(u, v_j) \mid u \in \mathcal{S}\} \cup \{(u, v_k) \mid u \in \mathcal{S}\}$

of edges, supporting scalability to large real-world graphs. At the same time, since the value of h_i is positively correlated with the number t_i of the triangles containing v_i, the adjustment effectively compensates for the triangles that are removed during the split; more triangles are likely to be removed with large t_i, but it tends to insert more edges in the adjustment. As a result, we address the two limitations of the naive version of NodeSam effectively.

Satisfying Desired Properties

NodeSam satisfies all the desired properties in Table 7.1. It is straightforward that Property 3 and 4 are satisfied since NodeSam changes the node features and the set of edges at every augmentation. We show in Lemma 15, 16, and 17 that NodeSam also satisfies the rest of the properties.

Lemma 15. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ be the result of NodeSam. Then, $\mathbb{E}[|\bar{\mathcal{V}}| - |\mathcal{V}|] = 0$ and $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$.*

Proof. The proof is straightforward for $\mathcal{V}$, since the number of nodes does not change by NodeSam. The proof for $\mathcal{E}$ requires a series of estimations for the properties of intermediate graphs, and thus the full proof is given in Section 7.7. The idea is that the expected number of edges removed by the merge operation is the same as h_i, which is the expected number of edges added by the adjustment operation as shown in line 4 of Algorithm 10. □

Lemma 16. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \mathbf{X})$ be a graph generated by NodeSam. Then, $\bar{G}$ is connected if and only if G is connected.*

Proof. The split and merge operations preserve the connectivity, since they are transformations between a single node and a pair of adjacent nodes. The adjustment also preserves the connectivity since it makes new edges only between two-hop neighbors. □

Lemma 17. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, the time and space complexities of NodeSam are both $O(d|\mathcal{V}| + |\mathcal{E}|)$, where d is the number of features.*

Proof. The time complexity of the split operation is $O(d|\mathcal{V}| + |\mathcal{E}|)$, and the complexity of the merge is the same. The time complexity of the adjustment operation is $O(\mathcal{V}| + |\mathcal{E}|)$ since it does not change $\mathbf{X}$. The space complexities are always smaller than or equal to the time complexities [133]. We prove the lemma by combining the complexities of all these three operations. □

7.3.2 SubMix: Subgraph Mix

SubMix aims to make a large degree of augmentation by combining multiple graphs. This is done by swapping subgraphs of different graphs, motivated by

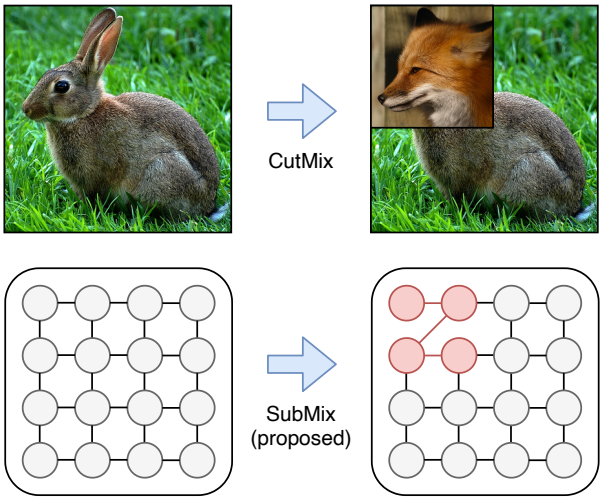


Figure 7.3: Comparison between image and graph mixing augmentation. SubMix, our proposed approach, generalizes CutMix [134] to graphs. The image patch in the top row corresponds to the red inserted subgraph in the bottom.

mixing approaches in the image domain [135, 136, 137]. The main idea is to treat the adjacency matrix of each graph like an image and replace a random patch of the matrix with that of another graph, as depicted in Figure 7.3 that compares our SubMix to CutMix [136] in the image domain. The red subgraph corresponds to the head of the fox in the augmented image.

The overall process of SubMix is shown as Algorithm 11. Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and a set $\mathcal{G}$ of all available graphs, SubMix makes an augmented graph by replacing a subgraph of G with that of another graph G' chosen from $\mathcal{G} \setminus \{G\}$. First, SubMix samples ordered sets S and S' of nodes from G and G', respectively, where $|S| = |S'|$. Then, SubMix makes a one-to-one mapping ϕ from the nodes in S' to those in S based on their order in each ordered set, and uses it to transfer the induced subgraph of S' into G, replacing the induced subgraph of S. The edges that connect S to the rest of the graph G are then

Algorithm 11: SubMix (Subgraph Mix)

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with its label y,
set $\mathcal{G}$ of all available graphs with labels $\mathcal{Y}$, and
target ratio $p \in (0, 1)$ of augmentation

Output: Augmented graph $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ and its label $\bar{\mathbf{y}}$

$G', y' \leftarrow$ Pick a random graph from $\mathcal{G} \setminus \{G\}$ and its label
$S, S' \leftarrow \text{Sample}(G, G', p)$ `// Algorithm 12`
$\phi \leftarrow$ Make the one-to-one mapping from S' to S
$\mathcal{E}_1 \leftarrow \{(u, v) \mid (u, v) \in \mathcal{E} \wedge \neg(u \in S \wedge v \in S)\}$
$\mathcal{E}_2 \leftarrow \{(\phi(u), \phi(v)) \mid (u, v) \in \mathcal{E}' \wedge (u \in S' \wedge v \in S')\}$
$\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}} \leftarrow \mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2, \mathbf{X}$
$\bar{\mathbf{X}}[\phi(S')] \leftarrow \mathbf{X}'[S']$ `// Replace a subset of features`
$\mathbf{y}, \mathbf{y}' \leftarrow$ Represent y and y' as one-hot vectors, respectively
$\bar{\mathbf{y}} \leftarrow (|\mathcal{E}_1|/|\bar{\mathcal{E}}|)\mathbf{y} + (1 - |\mathcal{E}_1|/|\bar{\mathcal{E}}|)\mathbf{y}'$

connected to the new nodes.

As a result, SubMix makes the set $\bar{\mathcal{E}} = \mathcal{E}_1 \cup \mathcal{E}_2$ of edges for the augmented graph $\bar{G}$, where $\mathcal{E}_1$ and $\mathcal{E}_2$ are extracted from G and G', respectively. The difference between $\mathcal{E}_1$ and $\mathcal{E}_2$ is that $\mathcal{E}_1$ contains the edges incident to at least one node in S, while $\mathcal{E}_2$ contains only the edges whose two connected nodes are both in S'. The reason is because the main target of augmentation is G. The subgraph of G' is inserted into G without changing the edges that connect S with $\mathcal{V} \setminus S$. See Figure 7.3 for a visual illustration.

One notable difference from NodeSam is that SubMix takes the label of G as an additional input and changes it. The label $\bar{\mathbf{y}}$ of $\bar{G}$ is softly determined as $\bar{\mathbf{y}} = q\mathbf{y} + (1 - q)\mathbf{y}'$, where $\mathbf{y}$ and $\mathbf{y}'$ are the one-hot labels of G and G', respectively, and $q = |\mathcal{E}_1|/|\bar{\mathcal{E}}|$. This is based on an assumption that edges are crucial factors to determine the label of a graph, and thus the ratio of included edges quantifies how much it contributes to making $\bar{\mathbf{y}}$.

Selecting Subgraphs with Diffusion

The core part of SubMix is the choice of ordered sets S and S'. The list of nodes included in each set determines the shape of the subgraph, and their order determines how the nodes in S' are connected to the nodes in $\mathcal{V} \setminus S$. A naive approach is to select a random subset from each graph without considering the structural information, but this is likely to make disconnected subgraphs containing few edges.

Instead, we utilize graph diffusion to select connected and clustered subgraphs from G and G'. The process of subgraph sampling is shown as Algorithm 12. Given a root node r, a diffusion operator propagates a signal from r to all other nodes following the graph structure. This assigns large affinity scores to the nodes close to r or having many common neighbors with r. Thus, selecting the top k nodes having the largest affinity scores provides a meaningful substructure around r containing a sufficient number of edges, and the chosen nodes can be used directly as the subset S.

We use personalized PageRank (PPR) as the diffusion function, which is a popular approach to measure the personalized scores of nodes. PPR [138] converts the adjacency matrix $\mathbf{A}$ of each graph to a matrix $\mathbf{S}$ of scores by diffusing the graph signals from all nodes: $\mathbf{S} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})^k$, where $\mathbf{D}$ is the degree matrix such that $D_{ii} = \sum_k A_{ik}$, and $\alpha = 0.15$ is the teleport probability. The i-th column of $\mathbf{S}$ contains the affinity scores of nodes with respect to node i. Thus, given a root node r, we return the r-th column of $\mathbf{S}$ as the result $\mathbf{c}$ of diffusion in line 3 of Algorithm 12.

Connected subgraphs In Algorithm 12, it is essential to guarantee the connectivity of nodes S and S' to allow SubMix to replace meaningful substructures. We guarantee the connectivity with two ideas in the algorithm. First, we bound the number of selected nodes by the size of the connected component containing each root node, since the input graphs can be disconnected. Second, if the nodes selected by PPR make disconnected subgraphs, we resample the nodes by the breadth-first search (BFS) that is guaranteed to make connected subgraphs. This is to prevent rare cases where PPR returns a disconnected graph, which has not occurred in all of our experiments but can happen theoretically.

Lemma 18. *Each set of nodes returned by Algorithm 12 contains only connected nodes for both G and G' and for any value of k.*

Proof. The proof is straightforward as we run BFS on the connected component of each root if S (or S') is not connected. □

Order of nodes The order of selected nodes determines how the nodes in S are matched with those in S', playing an essential role for the result of augmentation. One advantage of diffusion is that the selected nodes are ordered by their affinity scores, making nodes at the same relative position around the root nodes r and r' to be matched between S and S' in the replacement. The root r of S is always replaced with r' of S', and the replacement of all remaining nodes is determined by their affinity scores. This makes the generated graph $\bar{G}$ more plausible than in the naive approach that matches the nodes in S randomly with those in S'.

Algorithm 12: Sample in SubMix

Input: Target graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$,
another graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ selected from $\mathcal{G} \setminus \{G\}$, and
target ratio $p \in (0, 1)$ of augmentation
Output: Ordered sets $S \subseteq \mathcal{V}$ and $S' \subseteq \mathcal{V}'$ of connected nodes

$r, r' \leftarrow$ Pick random nodes from G and G', respectively
$\psi \leftarrow$ Make a function that finds the connected component
$k \leftarrow \text{uniform}(0, p) \cdot \min(|\psi(G, r)|, |\psi(G', r')|)$
for $(G_t, r_t) \in \{(G, r), (G', r')\}$ **do**
  $\mathbf{c}_t \leftarrow$ Compute the scores of nodes by Diffuse(G_t, r_t)
  $S_t \leftarrow$ Select k nodes having the largest scores in $\mathbf{c}_t$
  if S_t contains a disconnected node **then**
    $S_t \leftarrow$ Take the first k nodes from BFS(G_t, r_t)
  end if
  $S \leftarrow S_t$ if $G_t = G$ otherwise $S' \leftarrow S_t$
end for

Satisfying Desired Properties

SubMix satisfies all the desired properties in Table 7.1. It is straightforward that Property 3 and 4 are satisfied since SubMix changes the node features and the set of edges at every augmentation. We show in Lemma 15, 16, and 17 that SubMix also satisfies the rest of the properties.

Preserving size Since SubMix combines multiple graphs, it is not possible to directly show the satisfaction of Property 1 for any given graph G. For instance, the number of edges always increases if G is a chain graph and all other graphs in $\mathcal{G}$ are cliques. Thus, we assume that the target graph G is selected uniformly at random from $\mathcal{G}$ and prove the unbiasedness with $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$.

Lemma 19. *Given a set $\mathcal{G}$ of graphs, we sample different graphs $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ from $\mathcal{G}$ uniformly at random. Let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ be an augmented graph from SubMix. Then, $\mathbb{E}[|\mathcal{V}| - |\bar{\mathcal{V}}|] = 0$ and $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$.*

Proof. The proof is straightforward for $\mathcal{V}$, since SubMix does not change the number of nodes. For $\mathcal{E}$, let $\mathcal{E}_s$ and $\mathcal{E}'_s$ be the edges of induced subgraphs of S and S' in graphs G and G', respectively. Then, the following equations hold, since a) SubMix replaces only the edges in $\mathcal{E}_s$, b) G and G' are selected independently, and c) S and S' are selected by the same diffusion algorithm:

$$\begin{aligned}\mathbb{E}_{G,\bar{G}}[|\mathcal{E}| - |\bar{\mathcal{E}}|] &= \mathbb{E}_{G,G'}[|\mathcal{E}_s| - |\mathcal{E}'_s|] \\ &= \mathbb{E}_G[|\mathcal{E}_s|] - \mathbb{E}_{G'}[|\mathcal{E}'_s|] = 0.\end{aligned}$$

The fact that $\mathbb{E}[|\mathcal{E}| - |\bar{\mathcal{E}}|] = 0$ proves the lemma. □

Other properties Lemma 20 shows that SubMix preserves the connectivity of the given graph, while Lemma 21 shows that SubMix runs in linear time with respect to the number of edges.

Lemma 20. *Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $\bar{G} = (\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathbf{X}})$ be an augmented graph from SubMix. Then, $\bar{G}$ is connected if and only if G is connected.*

Proof. Let G' be a graph chosen for the augmentation of G by SubMix. The sets S and S' of nodes selected for the replacement are connected due to Lemma 20. If we treat the induced subgraphs of S and S' as supernodes, the replacement of S with S' does not change the connectivity of G, proving the lemma. □

Lemma 21. *Given graphs $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$, the time and space complexities of SubMix are $O(pd|\mathcal{V}| + |\mathcal{E}| + |\mathcal{E}'|)$, where p is the ratio of sampling, and d is the number of features.*

Proof. The time complexity of Algorithm 11 without including the sampling function is $O(pd|\mathcal{V}| + |\mathcal{E}| + |\mathcal{E}'|)$. We assume that the number of labels is neg-

ligible. The time complexity of Algorithm 12 is $O(|\mathcal{E}| + |\mathcal{E}'|)$, since the PPR diffusion is $O(|\mathcal{E}|)$ and $O(|\mathcal{E}'|)$ for G and G', respectively. The space complexities are always smaller than or equal to the time complexities [133]. □

7.4 Experiments

We perform experiments to answer the following questions:

Q1. **Accuracy (Section 7.4.2).** Do NodeSam and SubMix improve the accuracy of graph classifiers? Are they better than previous approaches for graph augmentation?

Q2. **Desired properties (Section 7.4.3).** Do NodeSam and SubMix satisfy the desired properties of Table 7.1 in real-world graphs as we claim theoretically in Section 7.3?

Q3. **Ablation study (Section 7.4.4).** Do our ideas for improving NodeSam and SubMix, such as the adjustment or diffusion operation, increase the accuracy of graph classifiers?

7.4.1 Experimental Setup

We introduce our experimental setup including datasets, baseline approaches, hyperparameters, and graph classifiers.

Datasets We use seven molecular datasets [139] summarized in Table 7.2, which have been used in previous works for graph classification [83]: D&D, ENZYMES, MUTAG, NCI1, NCI109, PROTEINS, and PTC-MR. Each dataset consists of multiple undirected graphs each of which represents a chemical com-

Table 7.2: Summary of datasets.

Dataset	Graphs	Nodes	Edges	Features	Labels
D&D[1]	1,178	334,925	843,046	89	2
ENZYMES[1]	600	19,580	37,282	3	6
MUTAG[1]	188	3,371	3,721	7	2
NCI1[1]	4,110	122,747	132,753	37	2
NCI109[1]	4,127	122,494	132,604	38	2
PROTEINS[1]	1,113	43,471	81,044	3	2
PTC-MR[1]	344	4,915	5,054	18	2

[1] `https://chrsmrrs.github.io/datasets`

pound, and the feature of each node is a one-hot embedded atomic type. The numbers of nodes and edges in the table include all graphs of each dataset.

Baselines We consider the following approaches as baselines for graph augmentation. DropEdge [132] removes an edge uniformly at random, while DropNode removes a node and all connected edges. AddEdge inserts an edge between a random pair of nodes. ChangeAttr augments the one-hot feature vector of a random node by changing the nonzero index. GraphCrop [126] selects a subgraph by diffusion and uses it as an augmented graph. NodeAug [130] combines ChangeAttr, DropEdge, and AddEdge to change the local neighborhood of a random target node. MotifSwap [131] swaps the two edges in an open triangle to preserve the connectivity during augmentation.

Classifier We use GIN [83] as a graph classifier for evaluating the accuracy, which is one of the most popular models for graph classification and shows great performance in many domains. The hyperparameters are searched in the same space as in their original paper [83] to ensure that the improvement of accuracy

comes from the augmentation, not from hyperparameter tuning: batch size in $\{32, 128\}$ and the dropout probability in $\{0, 0.5\}$.

Training details Following the experimental process of GIN [83], which we use as the classifier, we run 10-fold cross-validation for evaluation. The indices of chosen graphs for each fold are included in the provided code repository. The Adam optimizer [140] is used, and the learning rate starts from 0.01 and decreases by half at every 50 epochs until it reaches 350 epochs. We set the ratio p of augmentation for SubMix, which is the only hyperparameter of our proposed approaches, to 0.4. All of our experiments were done at a workstation with Intel Core i7-8700 and RTX 2080.

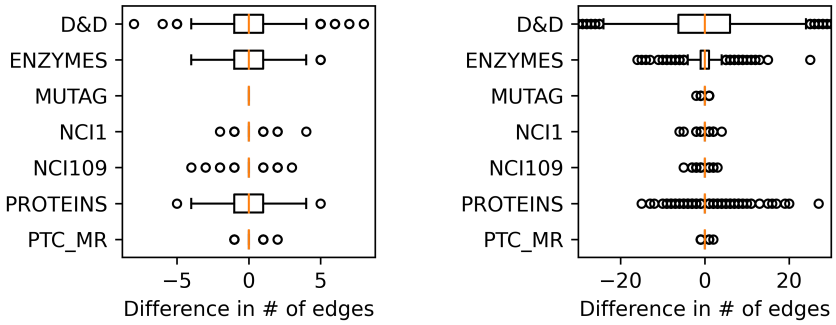
7.4.2 Accuracy of Graph Classification

Table 7.3 compares the accuracy of graph classification with various augmentation methods. The Rank column is made by getting the ranks of methods in each dataset and computing their average and standard deviation over all datasets. For example, if the rank of a method is 1, 3, and 3 in three datasets, respectively, its rank is reported as 2.3 ± 1.2. The rank measures the performance of each method differently from the average accuracy: one can achieve the highest rank even though its average accuracy is low.

NodeSam and SubMix improve the average accuracy of GIN by 2.2 and 2.0 percent points, which are $2.1\times$ and $1.8\times$ larger than the improvement of the best competitor, respectively. Note that all approaches except NodeSam and SubMix decrease the accuracy of GIN in some datasets; GraphCrop and DropEdge even decrease the average accuracy of GIN, implying that they give

Table 7.3: Accuracy of graph classification with various graph augmentation methods. The values in parentheses are the ranks of methods in each dataset, and the Rank column shows the average and standard deviation of ranks over all datasets. The proposed methods NodeSam and SubMix achieve the best average accuracy and the highest ranks at the same time.

Method	**D&D**	**ENZY.**	**MUTAG**	**NCI1**	**N109**	**PROT.**	**PTC**	**Average**	**Rank**
Baseline	76.40 (4)	50.33 (10)	89.94 (4)	82.68 (9)	81.80 (9)	75.38 (9)	63.94 (7)	74.35 (8)	7.43±2.51
GraphCrop	77.08 (2)	51.00 (9)	77.11 (10)	80.46 (10)	79.77 (10)	75.20 (10)	61.87 (10)	71.78 (10)	8.71±2.98
DropEdge	76.14 (6)	53.67 (6)	81.93 (9)	82.82 (7)	82.60 (7)	75.74 (4)	63.68 (8)	73.80 (9)	6.71±1.60
AddEdge	76.14 (7)	55.17 (3)	85.67 (8)	83.99 (2)	83.06 (5)	75.38 (8)	64.27 (5)	74.81 (7)	5.43±2.37
ChangeAttr	75.72 (9)	53.33 (8)	90.44 (3)	83.02 (5)	83.57 (2)	75.47 (7)	62.47 (9)	74.86 (6)	6.14±2.85
NodeAug	76.14 (8)	54.67 (5)	86.14 (7)	83.16 (4)	82.36 (8)	75.56 (6)	66.24 (2)	74.90 (5)	5.71±2.21
DropNode	75.55 (10)	55.17 (3)	87.28 (6)	82.85 (6)	83.04 (6)	75.65 (5)	**66.59 (1)**	75.16 (4)	5.29±2.81
MotifSwap	76.23 (5)	53.50 (7)	90.47 (2)	82.82 (7)	83.28 (4)	75.92 (3)	65.79 (3)	75.43 (3)	4.43±1.99
SubMix	**78.10 (1)**	57.50 (2)	89.94 (4)	**84.33 (1)**	**84.37 (1)**	**76.19 (1)**	63.97 (6)	76.34 (2)	**2.29±1.98**
NodeSam	76.57 (3)	**60.00 (1)**	**90.96 (1)**	83.33 (3)	83.52 (3)	76.10 (2)	65.48 (4)	**76.57 (1)**	2.43±1.13



(a) NodeSam (b) SubMix

Figure 7.4: Box plots representing the number of edges that our proposed methods change through augmentation. Both methods make unbiased augmentation, where the variance of changes depends on the characteristic of each dataset.

a wrong bias to the classifier by distorting the data distribution.

7.4.3 Preserving Desired Properties

The main motivation of our NodeSam and SubMix is to satisfy the desired properties of Table 7.1. We present empirical results that support our theoretical claims given in Section 7.3.1 and 7.3.2.

Figure 7.4 visualizes the change in the number of edges during augmentation as box plots. Both NodeSam and SubMix perform unbiased augmentation as shown in the orange lines that appear in the center of each plot. On the other hand, they make a sufficient change of edges through augmentation, generating diverse examples for the training of a classifier. SubMix makes a larger degree of augmentation than NodeSam, especially in the D&D dataset, as it combines multiple graphs of different structures.

Figure 7.5 shows the scalability of NodeSam and SubMix with the number of edges in a graph. We use the Reddit [141] dataset for this experiment, which

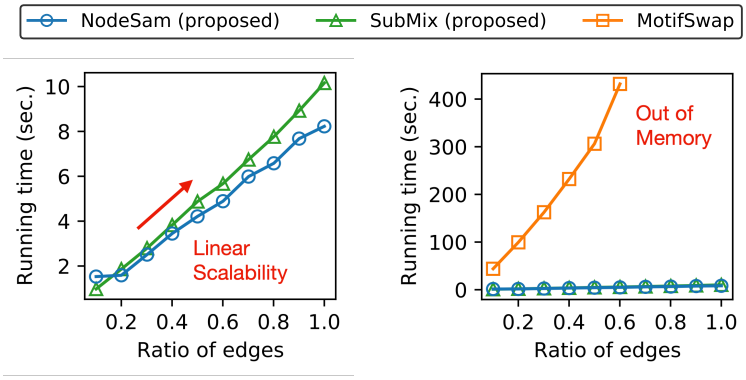
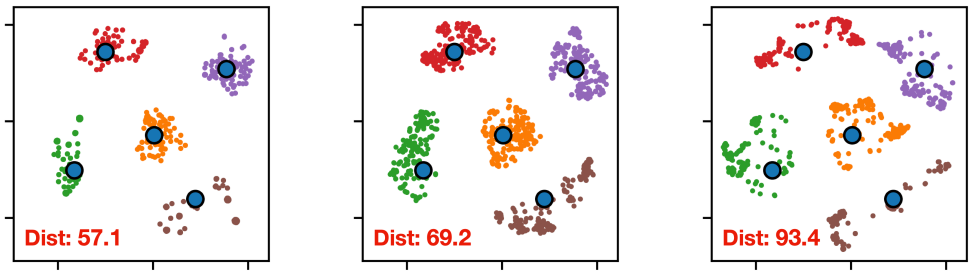


Figure 7.5: Computational time of NodeSam and SubMix, our proposed methods, and MotifSwap with respect to the number of edges. Our methods achieve linear scalability, while MotifSwap does not.

is large enough to cause scalability issues as it contains 232,965 nodes and 11,606,919 edges. We randomly make nine subgraphs with different sizes to measure the running time of methods. The figure shows that NodeSam and SubMix have linear scalability with similar patterns, while the computational time of MotifSwap increases much faster, causing out-of-memory errors. This supports our claims in Lemma 17 and 21.

Figure 7.6 visualizes the space of augmentation for our proposed algorithms and MotifSwap, the strongest baseline. We first use the Weisfeiler-Lehman (WL) kernel [142] to extract embedding vectors of graphs and then use the t-SNE algorithm [143] to visualize them in the 2D space. The distance value in each figure represents the average distance between each original graph and its augmented graphs in the 2D space. SubMix shows the largest space of augmentation by combining multiple graphs, effectively filling in the space between given examples. MotifSwap makes augmented graphs only near the original graphs, making the smallest average distance.



(a) MotifSwap (b) NodeSam (c) SubMix

Figure 7.6: Space of augmentation for various algorithms. The large blue points are original graphs, while the small points represent augmented graphs. SubMix has the largest space of augmentation, while MotifSwap has the smallest one.

7.4.4 Ablation Study

We perform an ablation study for both NodeSam and SubMix, and present the result in Figure 7.7. SplitOnly and MergeOnly refer to the split and merge operations of Algorithm 8 and 9, respectively. NodeSamBase refers to the naive version of NodeSam, which does not perform the adjustment operation of Algorithm 10. SubMixBase refers to the naive version of SubMix, which selects the target sets of nodes for the replacement uniformly at random, without using the diffusion operation of Algorithm 12. The vertical and horizontal axes of the figure are the average accuracy and rank over all datasets, corresponding to the last two columns of Table 7.3, respectively.

We have two observations from Figure 7.7. First, NodeSam and SubMix with all techniques achieve the best accuracy among all versions, showing that our proposed techniques are effective for improving their performance by satisfying the desired properties. Second, every basic version of our approaches still improves the baseline significantly, which is to use raw graphs without augmen-

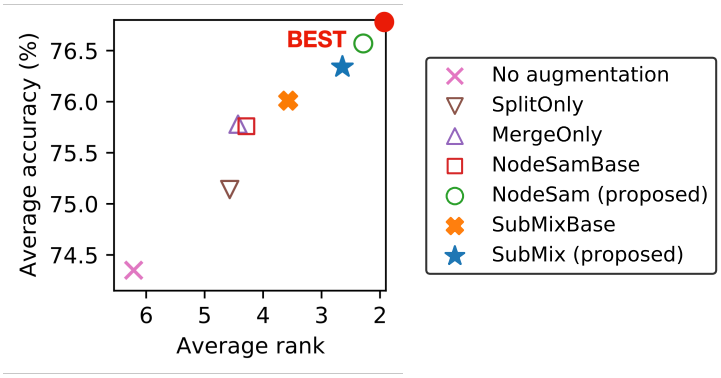


Figure 7.7: Ablation study for our NodeSam and SubMix. Both of them make the best average accuracy and rank based on our proposed ideas.

tation. This is clear when compared with existing methods shown in Table 7.3 that make marginal improvements or even decrease the baseline accuracy. Such improvement exhibits the effectiveness of our ideas, which are building blocks of NodeSam and SubMix.

We also perform an additional study for NodeSam and its basic versions in Section 7.8, which shows that NodeSamBase without the adjustment operation is likely to decrease the number of edges due to the merge operation that eliminates additional triangles.

7.5 Related Works

We review related works for graph augmentation, graph classification, and image augmentation.

Model-agnostic graph augmentation Our work focuses on model-agnostic augmentation, which is to perform augmentation independently from target models. Previous works can be categorized by the purpose of augmentation:

node-level tasks [130, 132, 144], graph-level tasks [126, 145, 131], or graph contrastive learning [146, 147]. Such methods rely on heuristic operations that make no theoretical guarantee for the degree of augmentation or the preservation of graph properties. We propose five desired properties for effective graph augmentation and show that our approaches satisfy all of them, resulting in the best accuracy in classification.

Model-specific graph augmentation Model-specific methods for graph augmentation make changes optimized for the target model. For example, changing the representation vectors of graphs learned by the target classifier by a mixup algorithm [148] is model-specific. Such model-specific approaches include manifold mixup [129], dynamic augmentation of edges [128, 149], and adversarial perturbations [150, 151]. Methods for graph adversarial training [152, 153, 154, 155] also belong to this category as they require the predictions of target models. Such approaches participate directly in the training process, and thus cannot be used generally in various settings.

Graph classification Graph classification is a core problem in the graph domain, which is to predict the label of each graph. Compared to node classification [38] that focuses on the properties of individual nodes, graph classification aims at extracting meaningful substructures or summarizing the information of multiple nodes to make a better representation of a graph. A traditional way to solve the problem is to utilize kernel methods [142, 156, 157, 158], while graph neural networks [38, 83, 138] have recently shown a better performance with advanced pooling methods [159, 160, 161] to aggregate node embeddings to the graph level.

We use graph classification as a downstream task for evaluating the quality of augmented graphs. Devising an augmentation algorithm for accurate graph classification requires separating the core information that determines the characteristic of each graph from the random and unimportant components.

Image augmentation Data augmentation is studied actively in the image domain. Basic approaches include color and geometric transformations [162, 163], masking [164, 165, 166], adversarial perturbations [167], pixel-based mixing [135, 168], and generative models [169, 170]. Such approaches utilize the characteristic of images where slight changes of pixel values do not change the semantic information of each image. On the other hand, patch-based mixing approaches that replace a random patch of an image to that of another image [136, 137, 171] give us a motivation to propose SubMix, which replaces an induced subgraph of a graph, instead of an image patch. This is based on the idea that images are grid-structured graphs whose pixels correspond to individual nodes.

7.6 Summary

We propose NodeSam (Node Split and Merge) and SubMix (Subgraph Mix), effective algorithms for model-agnostic graph augmentation. We first present desired properties for graph augmentation including the preservation of original properties, the guarantee of sufficient augmentation, and the linear scalability. Then, we show that both NodeSam and SubMix achieve all properties unlike all of the existing approaches. NodeSam aims to make a minimal change of the graph structure by performing opposite operations at once: splitting a node and

merging two nodes. On the other hand, SubMix aims to increase the degree of augmentation by combining random subgraphs of different graphs with different labels. Experiments on seven datasets show that NodeSam and SubMix achieve the best accuracy in graph classification with up to 2.1× larger improvement of accuracy compared with previous approaches.

7.7 Appendix: Unbiasedness of NodeSam

We have four graphs G, G', G'', and $\bar{G}$ presented in Algorithm 7, which denote the original graph, the graph after the split, the graph after the adjustment, and the final graph generated by NodeSam, respectively. Let v_i be the target node of the split operation, d_i be the degree of v_i in G, and h_i be the *expected* number of edges added in the adjustment operation. Let $T(\cdot)$ be the function that counts the number of triangles in a graph. Then, we analyze the properties of graphs generated during NodeSam in Lemma 22 to 25.

Lemma 22. $\mathbb{E}[T(G')] = T(G) - t_i/2$.

Proof. Each triangle in G containing v_i has a probability of 0.5 to be removed in the split operation, due to the random selection of the target node from v_j and v_k. Thus, it is expected that the half of all triangles containing v_i are removed in G'. □

Lemma 23. *In the adjustment operation of Algorithm 10, let l be the number of new triangles created in G'' by connecting an edge between a target node $u \in \mathcal{S}$ and either v_j or v_k. Then, $\mathbb{E}[l] = |\mathcal{N}_{ui}|/2 + 1$, where $\mathcal{N}_{ui}$ is the set of common neighbors between u and v_i in G.*

Proof. Node u is connected to either v_j or v_k in G', since it is a neighbor of v_i in the the original graph G before the split is done. Assume that u is connected to v_j in G' without loss of generality. Then, the adjustment for u makes an edge (u, v_k). A single triangle (u, v_j, v_k) is always created by this operation, since edges (u, v_j) and (v_j, v_k) already exist in G'. The number of additional triangles created by the adjustment is the same as the number of common neighbors between u and v_k in G', denoted by $|\mathcal{N}'_{uk}|$. Since we split $\mathcal{N}_i$ by the same probability into v_j and v_k during the split operation, $\mathbb{E}[|\mathcal{N}'_{uk}|] = |\mathcal{N}_{ui}|/2$. We prove the lemma by adding one. □

Lemma 24. $\mathbb{E}[T(G'')] = T(G) + h_i(t_i/d_i + 1) - t_i/2$.

Proof. Following Lemma 23, $|\mathcal{N}_{ui}|/2 + 1$ triangles are expected to be created for each target node $u \in \mathcal{S}$ during the adjustment. We estimate $|\mathcal{N}_{ui}|$ by $2t_i/d_i$ based on the relation

$$\sum_{k \in \mathcal{N}_i} |\mathcal{N}_{ki}| = 2t_i,$$

where $\mathcal{N}_i$ is the set of neighbors of v_i in G. We get

$$\mathbb{E}[T(G'') - T(G')] = h_i(t_i/d_i + 1)$$

by repeating the adjustment h_i times for every possible $u \in \mathcal{S}$. We prove the lemma by combining it with Lemma 22. □

Lemma 25. $\mathbb{E}[|\bar{\mathcal{E}}|] = |\mathcal{E}''| - 3T(G'')/|\mathcal{E}''| - 1$.

Proof. Let v_o and v_p be the target nodes of the merge operation, and let t''_{op} be the number of triangles containing v_o and v_p in G''. A single edge (v_o, v_p) is always removed by the merge operation as v_o and v_p are merged into one.

The number of additional edges removed by the merge operation is given as t''_{op}, which is estimated by $3T(G'')/|\mathcal{E}''|$ based on the relation

$$\sum_{(i,j)\in\mathcal{E}''} t''_{ij} = 3T(G'').$$

We prove the lemma by adding one to $3T(G'')/|\mathcal{E}''|$. □

Proof of Lemma 15 We now provide the full proof of Lemma 15 given in Section 7.3.1 and based on Lemma 22 to 25.

Proof. We prove only the edge part, since it is straightforward that NodeSam does not change the number of nodes. Let m be the expected number of edges removed by the merge operation:

$$m \equiv \mathbb{E}[|\mathcal{E}''| - |\bar{\mathcal{E}}|]. \tag{7.2}$$

Then, m is rewritten as follows by Lemma 25:

$$m = \mathbb{E}[3T(G'')/|\mathcal{E}''|] + 1. \tag{7.3}$$

If we ignore the dependence between $T(G'')$ and $|\mathcal{E}''|$, which is negligible in real-world graphs that sastify $|\mathcal{E}| \gg h_i$, Equation (7.3) changes into

$$m = \frac{3\mathbb{E}[T(G'')]}{\mathbb{E}[|\mathcal{E}''|]} + 1. \tag{7.4}$$

The first term $\mathbb{E}[T(G'')]$ is rewritten by Lemma 24:

$$\mathbb{E}[T(G'')] = |\mathcal{V}|t_i/3 + h_i(t_i/d_i + 1) - t_i/2, \tag{7.5}$$

where the global number $T(G)$ of triangles is replaced with a local estimation $|\mathcal{V}|t_i/3$, since we select v_i from G uniformly at random and the relation $\mathbb{E}_i[t_i] = 3T(G)$ holds between $T(G)$ and t_i.

The second term $\mathbb{E}[|\mathcal{E}''|]$ is given by the definition of h_i:

$$\mathbb{E}[|\mathcal{E}''|] = |\mathcal{E}| + h_i + 1. \tag{7.6}$$

We apply Equation (7.5) and (7.6) to Equation (7.4) to get

$$m = \frac{3(t_i/d_i + 1)h_i + (|\mathcal{V}| - 3/2)t_i}{h_i + |\mathcal{E}| + 1} + 1. \tag{7.7}$$

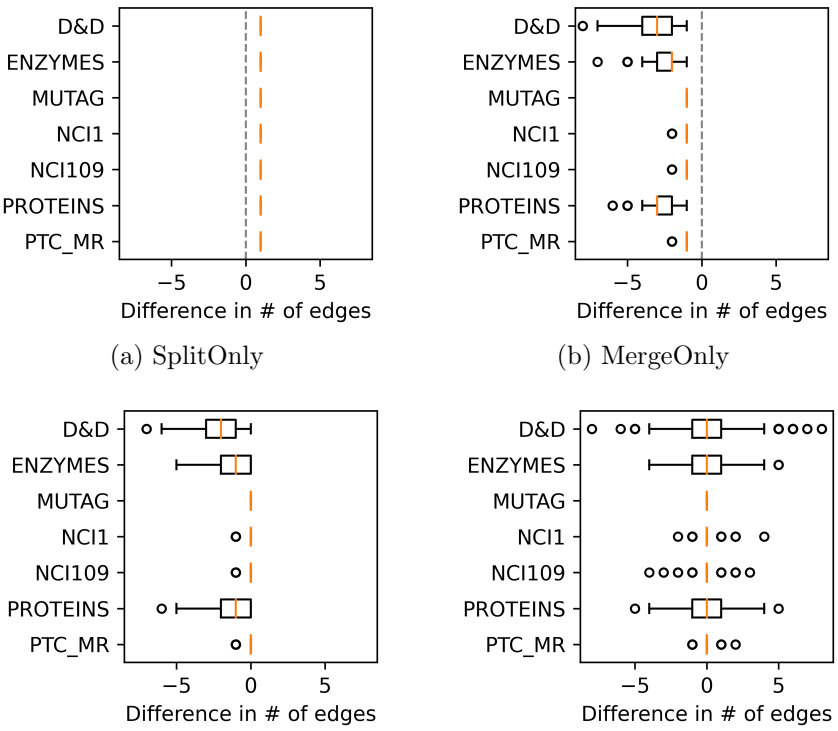
If we denote the first term of the right hand side of Equation (7.7) as A, we can verify that Equation (7.1) is the solution of $h_i = A$:

$$\begin{aligned} &h_i = A \\ &h_i(h_i + |\mathcal{E}| + 1) = 3(t_i/d_i + 1)h_i + (|\mathcal{V}| - 3/2)t_i \\ &h_i^2 + (|\mathcal{E}| - 3t_i/d_i - 2)h_i - (|\mathcal{V}| - 3/2)t_i = 0. \end{aligned} \tag{7.8}$$

Then, the right hand side of Equation (7.7) changes into $h_i + 1$. By the definition of m and h_i, we finally get

$$\mathbb{E}[|\mathcal{E}''| - |\bar{\mathcal{E}}|] = \mathbb{E}[|\mathcal{E}''|] - |\mathcal{E}| - 1 + 1,$$

which changes into $\mathbb{E}[|\bar{\mathcal{E}}| - |\mathcal{E}|] = 0$ and proves the lemma. □



(a) SplitOnly

(b) MergeOnly

(c) NodeSamBase

(d) NodeSam

Figure 7.8: Box plots illustrating the difference in the number of edges. NodeSamBase, which does not perform the adjustment operation, tends to decrease the number of edges due to the merge operation that removes additional edges.

7.8 Appendix: Ablation Study for NodeSam

We perform an additional ablation study for NodeSam to confirm the effect of the adjustment operation for making unbiased changes of the number of edges. Figure 7.8 shows box plots for the difference in the number of edges for all variants of NodeSam: (a) SplitOnly, (b) MergeOnly, (c) NodeSamBase, and (d) NodeSam. The detailed information of such variants is discussed in Section 7.4.4. The figure shows that the split operation increases the number of edges by one in all cases, since it splits a node into two by inserting a single edge, while the merge operation decreases the number of edges by more than one. This is because the merge operation eliminates the triangles that contain both of the target nodes. Our NodeSam with the adjustment operation makes unbiased changes, compensating for the removed edges, compared with NodeSamBase.

Part III

Conclusion and Future Works

Chapter 8

Conclusion

8.1 Conclusion

In this thesis, we introduce our works for solving various graph-related tasks with probabilistic modeling. The proposed approaches study the relationships between features, labels, and the structure of a graph, extending the coverage of graph algorithms into challenging scenarios with limited observations. They show great performance in various problem settings based on consistent, robust and interpretable decision processes made by probabilistic modeling.

The contributions of this thesis are categorized into two parts: structural exploitation (Part I) and structural modification (Part II). In Part I, we design various approaches for exploiting the structures of graphs to solve challenging graph-related problems. This is the basic way done by most graph algorithms, and is effective if a graph structure gives reliable evidence for estimation. Our contributions for the structural exploitation are presented as follows:

- Node classification in an edge-attributed graph (Chapter 3): We propose SBP, an approach to accurately classify nodes in edge-attributed graphs. SBP models the propagation strength as a function of edge attributes to control the degree of propagation via supervised learning.
- Hard inductive node classification (Chapter 4): We propose BPN, a deep learning-based approach to accurately classify unseen isolated nodes at the

test time. BPN shows a superior performance on hard inductive learning based on the separable structure of classification and diffusion.

- Node feature estimation (Chapter 5): We propose MGA, an autoencoder-like model that generates missing features from only partial observations existing in a graph. MGA presents a strong structural regularizer to avoid overfitting while estimating high-dimensional numerical features.

In Part 2, we propose two approaches that modify the structure of a graph with different objectives and assumptions. Structural modification has a great potential for many graph-related tasks, since it can change a given structure to match the target problem. On the other hand, it involves a risk of changing the essential properties of a graph, and thus an algorithm needs to be designed carefully. Our contributions are presented as follows:

- Subgraph sampling (Chapter 6): We propose BTW, an algorithm to sample subgraphs with bounded treewidth. The generated subgraphs can be used to a) speed up approximate inference or b) support tractable exact inference for higher accuracy of node classification.

- Graph augmentation (Chapter 7): We propose NodeSam and SubMix to augment graph structures for accurate graph classification. They satisfy five desired properties for effective augmentation by making balanced and sufficient changes of the graph structure in a model-agnostic way.

8.2 Overall Impact

In addition to the contributions above, our works have a broad impact on the field of graph mining. We highlight the impact of our work as follows:

- Our work on transductive node classification [1] was awarded the Google Conference Scholarship for attending ICDM 2017.
- Our work on subgraph sampling [2] was awarded the Qualcomm Innovation Fellowship Korea and the Samsung HumanTech Paper Award.
- Our two works on node classification [1, 3] were presented as invited talks at Korea Software Congress 2017 and 2019, respectively.
- The works included in this thesis [1, 2, 3, 4] were supported by the Google PhD Fellowship and the Yulchon AI Star Award.
- We have a registered patent on node classification [1] and two filed patents on node feature estimation and graph augmentation [4], respectively.

8.3 Observations and Discussion

Our studies for various graph-related tasks provide observations that enhance our understanding toward graph-structured data. We summarize the observations and insights derived from our works and discuss them in detail.

Properties of Feature Estimation

The feature estimation problem has not been studied actively in literature and can be understood in two ways. First, the problem is to generate intermediate evidence for improving node classification. The generated features are fed into node classifiers, giving additional information for nodes whose true features are unknown. Second, the problem is a generalization of node classification in that node labels are special cases of node features, i.e., they are features that follow a categorical distribution. In this perspective, there is no fundamental difference

Table 8.1: Comparison between SBP and graph neural networks in the MovieLens dataset [22]. The accuracy of SBP is taken from its original paper. SBP performs well and its accuracy is comparable to those of GNNs, even though it contains only five learnable parameters.

Method	**AUC**	**Parameters**
SGC [39]	78.91 ± 6.11	19,848
GCN [71]	69.67 ± 9.86	317,634
GraphSAGE [40]	81.60 ± 5.53	635,234
GAT [39]	80.12 ± 6.76	635,398
SBP (proposed)	80.65	5

between node classification and feature estimation, except that feature estimation is more difficult because a) the target nodes have no information that help the estimation and b) the target variables are high-dimensional vectors.

Various types of node features can be generated from our MGA based on the choice of a distribution for modeling feature variables. We introduce three promising options that can be applied to most real-world datasets. First, binary features are generated by modeling a Bernoulli distribution. This includes bag-of-words representations of texts or the membership of users to social groups. Second, categorial features are generated by modeling a categorical distribution. This includes node classes or any types of discrete information, e.g., the current location of a user. Third, any continuous features can be generated by modeling a Gaussian distribution. This is basically the same as making no assumption for the distribution of features, and includes most types of numerical information, such as RGB values in images or TF-IDF scores.

Comparison with Graph Neural Networks

Graph neural networks (GNN) are powerful tools that achieve state-of-the-art performance in many graph-related tasks, especially semi-supervised learning. We compare our SBP, designed for node classification in edge-attributed graphs, with various GNNs to analyze its strengths and weaknesses. The result is shown in Table 8.1. Since the nodes in the MovieLens dataset contain no features, we introduced one-hot node features to make GNNs identify each node. The result shows that SBP performs very well and its accuracy is comparable to those of various GNNs, even though SBP contains only five learnable parameters. At the same time, SGC with 20K parameters outperforms GCN and GAT, which learn stronger representations with at least 300K parameters.

We have the following observations from the result. First, a large number of parameters in GNNs are not used effectively as the dataset has no features for describing target nodes. The main strength of GNNs is the ability to fuse high-dimensional node features and a graph structure in a single learnable operation. Second, SBP performs well even with few parameters by focusing on modeling the heterogeneous relationships between target nodes by learning an edge potential function. This makes it effectively use the edge attributes, which play an essential role in achieving high accuracy of node classification.

Comparing Approaches for Structural Modification

We compare BTW and NS (NodeSam and SubMix), our proposed approaches for structural modification having different objectives. BTW aims to sample a simple tree-like graph to improve the efficiency of node classification, removing most of the original edges. Thus, BTW is highly likely to destroy communities

of nodes and change many properties of the original graph. On the other hand, NS is designed to create a new graph that preserves as many original properties as possible by making balanced and stable changes in the graph structure. NS makes unbiased changes in the numbers of nodes and edges.

BTW is designed for scenarios where target graphs are too large to handle and thus the removal of edges is necessary. BTW preserves the connectivity of nodes while removing edges, and thus is useful for node-level tasks where connectivity is more important than community information. On the other hand, BTW has a risk of reducing the performance of classifier models as it changes structural properties by removing most of the original edges. The risk increases when a graph contains edge attributes, each of which conveys unique information to represent the relationships between adjacent nodes.

NS is designed to increase the number of training graphs when the number of available graphs is not sufficient to train accurate classifiers. Since NS makes unbiased and balanced changes, it is effective when a graph structure is sensitive and its structural properties need to be preserved through augmentation. On the other hand, if the size of each graph is small, even minimal augmentation can cause a significant change of its properties and reduce the performance of target models. Thus, it needs to be checked that whether augmentation can be done safely without causing a shift of domain distributions.

Chapter 9

Future Works

All our approaches introduced in this thesis are based on the assumption that a graph structure is given for describing the target variables. They are mainly categorized by whether the given graph is modified or not, since it determines how the structural information should be treated by our algorithms. A possible way to extend our approaches to general domains is to learn a graph structure from any given dataset, e.g., a tabular dataset containing multivariate features. This allows us to utilize unknown correlations between target variables and to apply graph-based algorithms to any types of datasets.

There are our previous works that study the problem of learning unknown correlations from time series variables [172, 173]. Specifically, the proposed approaches first map each time series variable into a low-dimensional embedding vector and then compare the embeddings to measure the similarities between target variables. We found from experiments that the learned correlations improve the accuracy of forecasting even with simple forecasting models that contain fewer parameters than competitors, because they provide a new dimension of information for correlating multiple variables.

However, our previous works on the time series domain are not sufficient to be applied to general data domains due to the following reasons. First, they make full dense correlation matrices considering all pairs of relationships, i.e., matrices of size $m \times m$ when the number of target variables is m. This makes

it inefficient to apply graph algorithms to the learned correlations, because the structure itself provides little information for understanding the true relationships between examples. Second, our previous approaches focus on time series datasets, where each variable has multiple observations over time and thus it is relatively easy to capture the accurate correlations between variables. In most datasets where target variables are not observed enough, learning correlations is a more difficult problem that can easily cause overfitting.

Based on our previous studies in the time series domain for learning correlations between variables, we aim to develop an effective way to learn a sparse graph structure from any given dataset. This requires one to properly choose a distant metric for comparing different samples, a way to automatically sparsify the learned correlations, and a strong regularizer to avoid overfitting. This is a challenging problem, but can create a new perspective of understanding graph and non-graph data in a single parameterized model.

Another way of future works is to apply our previous works in the other fields of data mining into the graph domain. For example, our work on graph-based PU learning [174] is motivated by of our study on zero-shot knowledge distillation [175], which aims at addressing the shortage of observed data. Our works on interpretable tree classifiers [176, 177, 178] can also be applied to the graph domain, especially with probabilistic approaches, which provide inherent interpretability through probabilistic decision processes.

References

[1] J. Yoo, S. Jo, and U. Kang, "Supervised belief propagation: Scalable supervised inference on attributed networks," in *ICDM*, pp. 595–604, 2017.

[2] J. Yoo, U. Kang, M. Scanagatta, G. Corani, and M. Zaffalon, "Sampling subgraphs with guaranteed treewidth for accurate and efficient graphical inference," in *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pp. 708–716, ACM, 2020.

[3] J. Yoo, H. Jeon, and U. Kang, "Belief propagation network for hard inductive semi-supervised learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 4178–4184, ijcai.org, 2019.

[4] J. Yoo, S. Shim, and U. Kang, "Model-agnostic augmentation for accurate graph classification," in *WWW '22: The Web Conference 2022*, 2022.

[5] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring Artificial Intelligence in the New Millennium*, pp. 239–269, 2003.

[6] D. Koller and N. Friedman, *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[7] D. H. Chau, S. Pandit, and C. Faloutsos, "Detecting fraudulent personalities in networks of online auctioneers," in *Knowledge Discovery in Databases: PKDD 2006*, pp. 103–114, Springer, 2006.

[8] D. H. P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-Scale Graph Mining and Inference for Malware Detection," in *Proceedings of the 2011 SIAM International Conference on Data Mining*, pp. 131–142, Philadelphia, PA: Society for Industrial and Applied Mathematics, Dec. 2013.

[9] M. McGlohon, S. Bay, M. G. Anderle, D. M. Steier, and C. Faloutsos, "SNARE: a link analytic system for graph labeling and risk detection," in *KDD*, pp. 1265–1274, ACM, 2009.

[10] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, "Netprobe: a fast and scalable system for fraud detection in online auction networks," in *WWW*, pp. 201–210, 2007.

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

[12] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 635–644, ACM, 2011.

[13] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[14] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th international conference on World wide web*, 2010.

[15] J. Ha, S. Kwon, S. Kim, C. Faloutsos, and S. Park, "Top-n recommendation through belief propagation," in *CIKM*, pp. 2343–2346, 2012.

[16] A. Tamersoy, K. A. Roundy, and D. H. Chau, "Guilt by association: large scale malware detection by mining file-relation graphs," in *KDD*, pp. 1524–1533, 2014.

[17] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[18] L. Akoglu, "Quantifying political polarity based on bipartite opinion networks," in *ICWSM*, 2014.

[19] U. Kang, D. H. Chau, and C. Faloutsos, "Mining large graphs: Algorithms, inference, and discoveries," in *ICDE*, pp. 243–254, 2011.

[20] L. Yan, R. H. Dodier, M. Mozer, and R. H. Wolniewicz, "Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic.," *ICML*, 2003.

[21] P. Massa and P. Avesani, "Trust-aware recommender systems," in *Proceedings of the 2007 ACM conference on Recommender systems*, pp. 17–24, ACM, 2007.

[22] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 1361–1370, ACM, 2010.

[23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2016.

[24] T. H. Haveliwala, "Topic-sensitive pagerank," in *Proceedings of the 11th international conference on World Wide Web*, pp. 517–526, ACM, 2002.

[25] D. Koutra, T.-Y. Ke, U. Kang, D. H. Chau, H.-K. K. Pao, and C. Faloutsos, "Unifying Guilt-by-Association Approaches - Theorems and Fast Algorithms.," *ECML/PKDD*, 2011.

[26] M. Shahriari and M. Jalili, "Ranking nodes in signed social networks," *Social Network Analysis and Mining*, vol. 4, no. 1, pp. 1–12, 2014.

[27] J. Gonzalez, Y. Low, and C. Guestrin, "Residual splash for optimally parallelizing belief propagation," in *International Conference on Artificial Intelligence and Statistics*, pp. 177–184, 2009.

[28] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," *arXiv preprint arXiv:1206.6837*, 2012.

[29] A. Chechetka and C. Guestrin, "Focused belief propagation for query-specific inference.," in *AISTATS*, pp. 89–96, 2010.

[30] U. Kang, D. Chau, and C. Faloutsos, "Inference of beliefs on billion-scale graphs," *The 2nd Workshop on Large-scale Data Mining: Theory and Applications*, 2010.

[31] L. Akoglu, R. Chandy, and C. Faloutsos, "Opinion fraud detection in online reviews by network effects.," *ICWSM*, vol. 13, pp. 2–11, 2013.

[32] M. Jang, C. Faloutsos, S. Kim, U. Kang, and J. Ha, "PIN-TRUST: fast trust propagation exploiting positive, implicit, and negative information," in *CIKM*, pp. 629–638, 2016.

[33] E. Ayday and F. Fekri, "A belief propagation based recommender system for online services," in *RecSys*, pp. 217–220, 2010.

[34] Q. Yang, L. Wang, and N. Ahuja, "A constant-space belief propagation algorithm for stereo matching," in *Computer vision and pattern recognition (CVPR), 2010 IEEE Conference on*, pp. 1458–1465, IEEE, 2010.

[35] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "Deltacon: A principled massive-graph similarity function," in *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 162–170, SIAM, 2013.

[36] J. Jung, W. Jin, L. Sael, and U. Kang, "Personalized ranking in signed networks using signed random walk with restart," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 973–978, IEEE, 2016.

[37] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 40–48, JMLR.org, 2016.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[39] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, OpenReview.net, 2018.

[40] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, pp. 1025–1035, 2017.

[41] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *NIPS*, pp. 1993–2001, 2016.

[42] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.

[43] K. Yoon, R. Liao, Y. Xiong, L. Zhang, E. Fetaya, R. Urtasun, R. S. Zemel, and X. Pitkow, "Inference in probabilistic graphical models by graph neural networks," in *ICLR (Workshop)*, 2018.

[44] Y. C. Ng, N. Colombo, and R. Silva, "Bayesian semi-supervised learning with graph gaussian processes," in *NeurIPS*, pp. 1690–1701, 2018.

[45] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. S. Zemel, "Graph partition neural networks for semi-supervised classification," in *ICLR (Workshop)*, 2018.

[46] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel, "Lanczosnet: Multi-scale deep graph convolutional networks," in *ICLR*, 2019.

[47] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, pp. 855–864, 2016.

[48] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *KDD*, pp. 701–710, ACM, 2014.

[49] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[50] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade - Second Edition*, pp. 639–655, 2012.

[51] J. Liang, P. Jacobs, J. Sun, and S. Parthasarathy, "Semi-supervised embedding in attributed networks with outliers," in *SDM*, pp. 153–161, 2018.

[52] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining for malware detection," in *SDM*, pp. 131–142, 2011.

[53] L. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun, "Learning deep structured models," in *ICML*, 2015.

[54] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[55] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[56] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *SIGIR*, pp. 43–52, ACM, 2015.

[57] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *WWW*, pp. 507–517, 2016.

[58] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *ICML*, pp. 412–420, 1997.

[59] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[61] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *CoRR*, vol. abs/1811.05868, 2018.

[62] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *NeurIPS*, 2020.

[63] C. T. Duong, T. D. Hoang, H. T. H. Dang, Q. V. H. Nguyen, and K. Aberer, "On node features for graph neural networks," *CoRR*, vol. abs/1911.08795, 2019.

[64] Y. Yang and D. Li, "NENN: incorporate node and edge features in graph neural networks," in *ACML*, vol. 129 of *Proceedings of Machine Learning Research*, pp. 593–608, PMLR, 2020.

[65] X. Huang, Q. Song, Y. Li, and X. Hu, "Graph recurrent networks with attributed random walks," in *KDD*, pp. 732–740, ACM, 2019.

[66] L. Chen, S. Gong, J. Bruna, and M. Bronstein, "Attributed random walk as matrix factorization," in *neural information processing systems, graph representation learning workshop*, 2019.

[67] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. W. Tsang, "Learning on attribute-missing graphs," *CoRR*, vol. abs/2011.01623, 2020.

[68] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[69] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *CoRR*, vol. abs/1406.2661, 2014.

[70] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[71] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016.

[72] C. Zhang, D. Florêncio, and P. A. Chou, "Graph signal processing-a probabilistic framework," *Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2015-31*, 2015.

[73] I. Han, D. Malioutov, and J. Shin, "Large-scale log-determinant computation through stochastic chebyshev expansions," in *ICML*, vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 908–917, JMLR.org, 2015.

[74] M. Tomczak, S. Swaroop, and R. E. Turner, "Efficient low rank gaussian variational inference for neural networks," in *NeurIPS*, 2020.

[75] D. A. Harville, "Matrix algebra from a statistician's perspective," 1998.

[76] R. K. Ando and T. Zhang, "Learning on graph with laplacian regularization," in *NIPS*, pp. 25–32, MIT Press, 2006.

[77] J. Pang and G. Cheung, "Graph laplacian regularization for image denoising: Analysis in the continuous domain," *IEEE Trans. Image Process.*, vol. 26, no. 4, pp. 1770–1785, 2017.

[78] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, pp. 974–983, ACM, 2018.

[79] Ö. Simsek and D. D. Jensen, "Navigating networks by using homophily and degree," *Proc. Natl. Acad. Sci. USA*, vol. 105, no. 35, pp. 12758–12762, 2008.

[80] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, pp. 8024–8035, 2019.

[81] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, OpenReview.net, 2019.

[82] M. Qu, Y. Bengio, and J. Tang, "GMNN: graph markov neural networks," in *ICML*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 5241–5250, PMLR, 2019.

[83] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[84] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *ICML*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 7134–7143, PMLR, 2019.

[85] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *AAAI*, pp. 10737–10745, AAAI Press, 2021.

[86] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Curgraph: Curriculum learning for graph classification," in *WWW*, pp. 1238–1248, ACM / IW3C2, 2021.

[87] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie, "Graph structure estimation neural networks," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pp. 342–353, ACM / IW3C2, 2021.

[88] J. You, X. Ma, D. Y. Ding, M. J. Kochenderfer, and J. Leskovec, "Handling missing data with graph representation learning," in *NeurIPS*, 2020.

[89] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional networks," in *ICDM*, pp. 929–934, IEEE Computer Society, 2018.

[90] H. Cui, Z. Lu, P. Li, and C. Yang, "On positional and structural node features for graph neural networks on non-attributed graphs," *CoRR*, vol. abs/2107.01495, 2021.

[91] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *ICLR*, OpenReview.net, 2020.

[92] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[93] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *WWW*, pp. 1067–1077, 2015.

[94] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, pp. 1225–1234, ACM, 2016.

[95] S. Jo, J. Yoo, and U. Kang, "Fast and scalable distributed loopy belief propagation on real-world graphs," in *WSDM*, pp. 297–305, 2018.

[96] A. T. Ihler, J. W. F. III, and A. S. Willsky, "Loopy belief propagation: Convergence and effects of message errors," *Journal of Machine Learning Research*, vol. 6, pp. 905–936, 2005.

[97] J. M. Mooij and H. J. Kappen, "Sufficient conditions for convergence of loopy belief propagation," *CoRR*, vol. abs/1207.1405, 2012.

[98] W. Gatterbauer, S. Günnemann, D. Koutra, and C. Faloutsos, "Linearized and single-pass belief propagation," *PVLDB*, vol. 8, no. 5, pp. 581–592, 2015.

[99] W. Gatterbauer, "The linearization of belief propagation on pairwise markov random fields," in *AAAI*, pp. 3747–3753, 2017.

[100] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *KDD*, 2006.

[101] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 2, pp. 277–284, 1987.

[102] S. Nie, D. D. Mauá, C. P. de Campos, and Q. Ji, "Advances in learning bayesian networks of bounded treewidth," in *NIPS*, 2014.

[103] S. Nie, C. P. de Campos, and Q. Ji, "Learning bayesian networks with bounded tree-width via guided search," in *AAAI*, pp. 3294–3300, 2016.

[104] M. Scanagatta, G. Corani, C. P. de Campos, and M. Zaffalon, "Learning treewidth-bounded bayesian networks with thousands of variables," in *NIPS*, pp. 1462–1470, 2016.

[105] R. Li, J. X. Yu, L. Qin, R. Mao, and T. Jin, "On random walk based graph sampling," in *ICDE*, pp. 927–938, 2015.

[106] N. K. Ahmed, J. Neville, and R. R. Kompella, "Network sampling: From static to streaming graphs," *TKDD*, vol. 8, no. 2, pp. 7:1–7:56, 2013.

[107] H. L. Bodlaender and A. M. C. A. Koster, "Treewidth computations i. upper bounds," *Inf. Comput.*, vol. 208, no. 3, pp. 259–275, 2010.

[108] H. Patil, "On the structure of k-trees," *Journal of Combinatorics, Information and System Sciences*, vol. 11, no. 2-4, pp. 57–64, 1986.

[109] D. Karger, D. Karger, and N. Srebro, "Learning markov networks: Maximum bounded tree-width graphs," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pp. 392–401, Society for Industrial and Applied Mathematics, 2001.

[110] N. Srebro, "Maximum likelihood bounded tree-width markov networks," *Artificial intelligence*, vol. 143, no. 1, pp. 123–138, 2003.

[111] M. Scanagatta, G. Corani, M. Zaffalon, J. Yoo, and U. Kang, "Efficient learning of bounded-treewidth bayesian networks from complete and incomplete data sets," *Int. J. Approx. Reasoning*, vol. 95, pp. 152–166, 2018.

[112] A. Cano and S. Moral, "Heuristic algorithms for the triangulation of graphs," in *IPMU*, pp. 98–107, 1994.

[113] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *KDD*, 2005.

[114] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding graph sampling algorithms for social network analysis," in *ICDCS*, pp. 123–128, 2011.

[115] E. Voudigari, N. Salamanos, T. Papageorgiou, and E. J. Yannakoudakis, "Rank degree: An efficient algorithm for graph sampling," in *ASONAM*, pp. 120–129, 2016.

[116] Y. Lim and U. Kang, "MASCOT: memory-efficient and accurate sampling for counting local triangles in graph streams," in *KDD*, pp. 685–694, 2015.

[117] Y. Lim, M. Jung, and U. Kang, "Memory-efficient and accurate sampling for counting local triangles in graph streams: From simple to multigraphs," *ACM Trans. Knowl. Discov. Data*, vol. 12, pp. 4:1–4:28, Jan. 2018.

[118] M. Jung, Y. Lim, S. Lee, and U. Kang, "FURL: fixed-memory and uncertainty reducing local triangle counting for multigraph streams," *Data Min. Knowl. Discov.*, vol. 33, no. 5, pp. 1225–1253, 2019.

[119] S. Nandanwar and M. N. Murty, "Structural neighborhood based classification of nodes in a network," in *KDD*, pp. 1085–1094, 2016.

[120] L. A. Adamic and N. S. Glance, "The political blogosphere and the 2004 U.S. election: divided they blog," in *LinkKDD*, pp. 36–43, 2005.

[121] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J. Cui, and A. G. Percus, "Reducing large internet topologies for faster simulations," in *NETWORKING*, pp. 328–341, 2005.

[122] B. F. Ribeiro and D. F. Towsley, "Estimating and sampling graphs with multidimensional random walks," in *IMC*, pp. 390–403, 2010.

[123] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, vol. 6, p. 60, 2019.

[124] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," in *IJCAI*, pp. 4653–4660, ijcai.org, 2021.

[125] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. H. Hovy, "A survey of data augmentation approaches for NLP," in *Findings of ACL*, 2021.

[126] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Graphcrop: Subgraph cropping for graph classification," *CoRR*, vol. abs/2009.10564, 2020.

[127] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018.

[128] T. Zhao, Y. Liu, L. Neves, O. J. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," *CoRR*, vol. abs/2006.06830, 2020.

[129] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang, "Graphmix: Regularized training of graph neural networks for semi-supervised learning," *CoRR*, vol. abs/1909.11715, 2019.

[130] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, "Nodeaug: Semi-supervised node classification with data augmentation," in *KDD*, pp. 207–217, ACM, 2020.

[131] J. Zhou, J. Shen, and Q. Xuan, "Data augmentation for graph classification," in *CIKM*, pp. 2341–2344, ACM, 2020.

[132] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, OpenReview.net, 2020.

[133] R. V. Book, "Comparing complexity classes," *J. Comput. Syst. Sci.*, vol. 9, no. 2, pp. 213–229, 1974.

[134] B. Du and H. Tong, "Mrmine: Multi-resolution multi-network embedding," in *CIKM*, 2019.

[135] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *ICLR*, OpenReview.net, 2018.

[136] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *ICCV*, pp. 6022–6031, IEEE, 2019.

[137] J. Kim, W. Choo, and H. O. Song, "Puzzle mix: Exploiting saliency and local statistics for optimal mixup," in *ICML*, vol. 119, pp. 5275–5285, PMLR, 2020.

[138] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *NeurIPS*, pp. 13333–13345, 2019.

[139] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *CoRR*, vol. abs/2007.08663, 2020.

[140] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[141] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *ICLR*, OpenReview.net, 2020.

[142] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[143] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[144] R. Song, F. Giunchiglia, K. Zhao, and H. Xu, "Topological regularization for graph neural networks augmentation," *CoRR*, vol. abs/2104.02478, 2021.

[145] K. Fu, T. Mao, Y. Wang, D. Lin, Y. Zhang, J. Zhan, X. Sun, and F. Li, "Ts-extractor: large graph exploration via subgraph extraction based on topological and semantic information," *J. Vis.*, vol. 24, no. 1, pp. 173–190, 2021.

[146] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," in *ICML*, vol. 139 of *Proceedings of Machine Learning Research*, pp. 12121–12132, PMLR, 2021.

[147] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *NeurIPS*, 2020.

[148] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pp. 3663–3674, ACM / IW3C2, 2021.

[149] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI*, 2020.

[150] F. Feng, X. He, J. Tang, and T. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *CoRR*, vol. abs/1902.08226, 2019.

[151] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein, "FLAG: adversarial data augmentation for graph neural networks," *CoRR*, vol. abs/2010.09891, 2020.

[152] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*, 2018.

[153] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, OpenReview.net, 2019.

[154] K. Xu, H. Chen, S. Liu, P. Chen, T. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," in *IJCAI*, pp. 3961–3967, ijcai.org, 2019.

[155] J. Ma, S. Ding, and Q. Mei, "Towards more practical adversarial attacks on graph neural networks," in *NeurIPS*, 2020.

[156] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *KDD*, pp. 1365–1374, ACM, 2015.

[157] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *CoRR*, vol. abs/1606.08928, 2016.

[158] B. Rieck, C. Bock, and K. M. Borgwardt, "A persistent weisfeiler-lehman procedure for graph classification," in *ICML*, vol. 97, pp. 5448–5458, PMLR, 2019.

[159] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, pp. 4805–4815, 2018.

[160] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *KDD*, pp. 723–731, ACM, 2019.

[161] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *ICML*, 2020.

[162] A. Galdran, A. Alvarez-Gila, M. I. Meyer, C. L. Saratxaga, T. Araujo, E. Garrote, G. Aresta, P. Costa, A. M. Mendonça, and A. J. C. Campilho, "Data-driven color augmentation techniques for deep skin image analysis," *CoRR*, vol. abs/1703.03702, 2017.

[163] I. Rocco, R. Arandjelovic, and J. Sivic, "Convolutional neural network architecture for geometric matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 11, pp. 2553–2567, 2019.

[164] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.

[165] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *AAAI*, pp. 13001–13008, AAAI Press, 2020.

[166] K. K. Singh and Y. J. Lee, "Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization," in *ICCV*, pp. 3544–3553, IEEE Computer Society, 2017.

[167] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 2574–2582, IEEE Computer Society, 2016.

[168] J. Lee, M. Z. Zaheer, M. Astrid, and S. Lee, "Smoothmix: a simple yet effective data augmentation to train robust classifiers," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pp. 3264–3274, IEEE, 2020.

[169] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, 2018.

[170] X. Zhu, Y. Liu, J. Li, T. Wan, and Z. Qin, "Emotion classification with data augmentation using generative adversarial networks," in *PAKDD*, vol. 10939 of *Lecture Notes in Computer Science*, pp. 349–360, Springer, 2018.

[171] R. Takahashi, T. Matsubara, and K. Uehara, "Data augmentation using random image cropping and patching for deep cnns," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 9, pp. 2917–2931, 2020.

[172] J. Yoo and U. Kang, "Attention-based autoregression for accurate and efficient multivariate time series forecasting," in *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*, pp. 531–539, SIAM, 2021.

[173] J. Yoo, Y. Soun, Y. Park, and U. Kang, "Accurate multivariate stock movement prediction via data-axis transformer with multi-level contexts," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pp. 2037–2045, ACM, 2021.

[174] J. Yoo, J. Kim, H. Yoon, G. Kim, C. Jang, and U. Kang, "Accurate graph-based pu learning without class prior," in *ICDM*, 2021.

[175] J. Yoo, M. Cho, T. Kim, and U. Kang, "Knowledge extraction with no observable data," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 2701–2710, 2019.

[176] J. Yoo and L. Sael, "Edit: Interpreting ensemble models via compact soft decision trees," in *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pp. 1438–1443, IEEE, 2019.

[177] J. Yoo and L. Sael, "Gaussian soft decision trees for interpretable feature-based classification," in *Advances in Knowledge Discovery and Data Mining - 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11-14, 2021, Proceedings, Part II*, vol. 12713 of *Lecture Notes in Computer Science*, pp. 143–155, Springer, 2021.

[178] J. Yoo and L. Sael, "Transition matrix representation of trees with transposed convolutions," in *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, SIAM, 2022.

요 약

그래프 데이터는 소셜 네트워크, 웹 그래프, 화합물 그래프 등 다양한 형태로 실세계에 나타난다. 이러한 실세계 그래프는 정점 피처나 레이블과 같이 다양한 형태의 속성 정보를 갖는데, 그래프 내의 각 간선이 의미하는 구체적인 관계는 이러한 속성 정보에 의해 결정된다. 즉, 연결된 정점들이 비슷한 특징을 갖는다는 일반적인 동종친화성 특성이 이러한 속성 그래프에는 적용되지 않고, 그래프의 내재적 특성이 동적으로 결정되는 것이다. 따라서, 실세계 그래프가 갖는 복잡한 특성을 고려하는 효율적인 알고리즘을 설계하는 것은 어려운 문제로 여겨진다.

본 학위 논문의 목표는 확률 기반의 방법을 통해 그래프 내에 존재하는 피처, 라벨, 그리고 그래프 구조 사이의 동적인 관계를 이해하는 것이다. 구체적으로 본 학위 논문에서는 정점 분류, 귀납적 학습, 정점 피처 추론, 서브그래프 샘플링, 그래프 증강 등의 문제를 다룬다. 제안하는 방법들은 기존 방법에 비해 실세계 그래프에서 우수한 성능을 보이는데, 특히 주어진 데이터의 양이 적어 그래프 신경망과 같은 복잡한 모델을 학습하기 어려운 상황에서 그 장점이 두드러진다. 이는 확률 모델링이 적은 파라미터를 가지고 그래프 전체에 적용되는 일관적인 확률적 가정을 만들 수 있기 때문에 갖는 강건성 및 확장성 때문이다.

본 논문에서 제안하는 방법들은 크게 구조 활용과 구조 개선의 두 가지 분류로 나뉜다. 구조 활용 알고리즘의 목적은 주어진 그래프 구조를 최대한 이용하여 정점의 내재적 특성 및 여러 정점 사이의 복잡한 관계를 이해하는 것이다. 주어진 그래프 구조는 상수로 취급되어 변하지 않는다고 가정한다. 구조 개선 알고리즘의 목적은 주어진 그래프의 구조를 수정하여 목표하는 속성을 더 잘 드러나게 하거나 원하는 모델의 학습을 돕는 것이다. 그래프 구조 자체를 변화시키기 때문에 그래프 기반 모델의 성능을 큰 폭으로 향상시킬 수 있고, 주어진 데이터셋의 신뢰도가

낮아 훈련에 충분한 근거를 제공하지 않을 때 그 효과가 두드러진다.

실세계 데이터에 대한 대규모의 실험 결과 본 논문에서 제시하는 방법들은 기존에 존재하던 확률 및 신경망 기반 방법들에 비해 뛰어난 성능을 보였다. 본 논문에서는 특히 그래프 분야의 대표적인 문제인 정점 분류 및 그래프 분류 문제에서 다양한 실험을 수행하였다. 정점 분류 문제에서는 일반적 상황에서 최대 15.6% 더 높은 정확도, 귀납 학습에서 최대 5.2% 더 높은 정확도, 서브그래프 샘플링을 통해 최대 13.7% 더 높은 F1 점수, 그리고 피처 추론을 통해 최대 7.0% 더 높은 정확도를 얻었다. 그래프 분류 문제에서는 기존 분류기의 성능을 일관성 있게 개선하였고 기존의 그래프 증강 기법에 비해 최대 2.1배의 성능 개선을 이루었다.

주요어 : 그래프 추론, 그래프 신경망, 신뢰 전파 알고리즘, 마코프 랜덤 필드, 귀납 학습, 정점 분류, 피처 추론, 서브그래프 샘플링, 그래프 분류, 그래프 증강
학번 : 2016-21218

감사의 글

우선 박사과정 동안 저를 지도하고 이끌어 주신 강유 교수님께 진심어린 감사를 드립니다. 단순히 연구를 하고 논문을 쓰는 것을 넘어 공학적 언어로 세상을 인식하고, 이해하고, 문제를 해결할 수 있는 시야를 갖게 해주심에 감사합니다. 포기하지 않고 긍정적인 마음으로 도전하는 교수님의 태도가 박사과정 동안 제게 큰 영감을 주었고, 그로 인해 멈추지 않고 앞으로 나아갈 수 있었습니다.

졸업논문 심사에 참여하여 귀중한 의견을 주신 김형주 교수님, 김선 교수님, 박근수 교수님, 유환조 교수님께도 깊은 감사의 말씀을 드립니다.

박사과정 동안 함께 연구했던 뛰어난 동료들께도 감사의 마음을 전합니다. 전북대학교 정진홍 교수님, 데이터 마이닝 연구실의 조새한, 전현식, 조민용, 김태범, 손예준, 박용찬, 김정헌, 윤호영, 심수연, 스위스 IDSIA 연구소의 Mauro Scanagatta, Giorgio Corani, Marco Zaffalon, 엔씨소프트의 장창원, 김건수, KAIST의 신기정 교수님, 최민영, 이건, 백운성 님께 감사드립니다. 새로운 주제에 도전할 용기와 영감을 주신 아주대학교 이슬 교수님께 특히 감사드립니다.

모두 언급하기는 어렵지만, 친구이자 동료로서 많은 시간을 함께했던 데이터 마이닝 연구실의 모든 구성원들에게 진심어린 감사를 전합니다.

무엇보다 제 곁에서 한결같이 저를 응원해준 가족들에게 가장 큰 감사를 전합니다. 저의 가장 친한 친구였던 일진, 상화, 효진, 효이에게, 그리고 엄마, 아빠, 장모님, 장인어른, 할머니, 할아버지께 갚을 수 없는 사랑을 받았습니다.

마지막으로, 제가 박사과정 동안 이룬 모든 것들을 제가 살아가는 이유이자 제 모든 기쁨과 열정의 원천인 세진에게 바칩니다. 세진의 무한한 인내와 사랑 없이는 지금 제가 가진 그 무엇도 없었을 것입니다. 또한, 처음부터 끝까지 인도하시고 저의 삶을 축복해주시는 하나님께 감사드립니다.