# Rethinking Propagation for Unsupervised Graph Domain Adaptation

**Meihan Liu**[1], **Zeyu Fang**[1], **Zhen Zhang**[3], **Ming Gu**[1], **Sheng Zhou**[1,2*], **Xin Wang**[4], **Jiajun Bu**[1]

[1]Zhejiang Provincial Key Laboratory of Service Robot,
College of Computer Science, Zhejiang University, Hangzhou, China
[2]School of Software Technology, Zhejiang University, Ningbo, China
[3]Department of Computer Science, National University of Singapore, Singapore
[4]Department of Computer Science, Tsinghua University, Beijing, China
{lmh_zju, fangzeyu, guming444, zhousheng_zju, bjj}@zju.edu.cn,
zhen@nus.edu.sg, xin_wang@tsinghua.edu.cn

## Abstract

Unsupervised Graph Domain Adaptation (UGDA) aims to transfer knowledge from a labelled source graph to an unlabelled target graph in order to address the distribution shifts between graph domains. Previous works have primarily focused on aligning data from the source and target graph in the representation space learned by graph neural networks (GNNs). However, the inherent generalization capability of GNNs has been largely overlooked. Motivated by our empirical analysis, we reevaluate the role of GNNs in graph domain adaptation and uncover the pivotal role of the propagation process in GNNs for adapting to different graph domains. We provide a comprehensive theoretical analysis of UGDA and derive a generalization bound for multi-layer GNNs. By formulating GNN Lipschitz for k-layer GNNs, we show that the target risk bound can be tighter by removing propagation layers in source graph and stacking multiple propagation layers in target graph. Based on the empirical and theoretical analysis mentioned above, we propose a simple yet effective approach called A2GNN for graph domain adaptation. Through extensive experiments on real-world datasets, we demonstrate the effectiveness of our proposed A2GNN framework.

## Introduction

The past decade has witnessed remarkable achievements of Graph Neural Networks (GNNs) (Kipf and Welling 2017; Zhang et al. 2022; Yang et al. 2022) across a wide range of applications, from social network analysis (Xia et al. 2021) and protein prediction (Li et al. 2021), to traffic flow forecasting (Lu et al. 2022), etc. As the distribution shifts commonly exist among real-world graphs, it makes the majority of existing GNNs fail to generalize to new domains. Recently, Unsupervised Graph Domain Adaptation (UGDA) has emerged as a key focus in both industrial and academic communities. The goal of UGDA is to facilitate the knowledge transfer from a labelled source graph to an unlabelled target graph, which addresses the issue of distribution shifts, thus unlocking the full potential of graph neural networks for diverse applications.
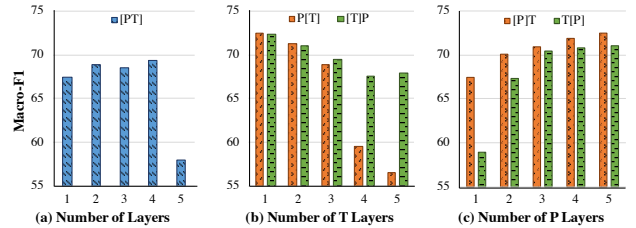
Figure 1: The influence of different operations in GNNs on task $D \to A$. $[\cdot]$ indicates the module stacking operation.

Inspired by the success of domain adaptation in computer vision and natural language processing (Ganin and Lempitsky 2015), most existing UGDA methods have primarily concentrated on aligning source and target graph distributions in the representation space generated by GNNs (Zhu et al. 2023; Liu et al. 2023). This includes learning domain-invariant representations with measurements like maximum mean discrepancy (Shen et al. 2021), graph subtree discrepancy (Wu, He, and Ainsworth 2023), or performing domain discrimination (Shen et al. 2020; Zhang et al. 2021; Dai et al. 2022; Wu et al. 2020; You et al. 2023). However, the underlying generalization capability of GNNs has been largely overlooked. Given that the representations learned by GNNs play a pivotal role in UGDA, we propose to revisit the step prior to alignment and pose a fundamental research question: ***Does GNNs have an inherent capability to adapt to new domains?***

To answer this question, we first decouple GNN's mechanism into two independent operations: *Propagation* (**P**) and *Transformation* (**T**), where the **P** operation performs message passing between neighboring nodes and the **T** operation conducts linear transformations to the node representations. Following the disentanglement, the GNN's architecture can be constructed with the combinations of **P** and **T**. Figure 1 presents the results under different architectures, from which we have the following key observations: *(i) Simultaneously adding P and T does not substantially enhance its generalization capability*. As we can see from Figure 1(a), an incremental trend can be observed when the number of layers is

fewer than 4, and more layers will result in decreased performance. *(ii) Solely adding T may impair its adaptation capability*. The UGDA performance in Figure 1(b) shows a significant decrease with the addition of more transformation layers while keeping the number of propagation layers fixed; *(iii) Operation P plays a crucial role in enhancing its adaptation capability*. Interestingly, results in Figure 1(c) demonstrates that increasing propagation layers can achieve significant performance improvements compared with increasing both transformation and propagation layers.

Motivated by the above findings, we revisit the GNN's architecture in unsupervised graph domain adaptation and propose a simple yet effective model named **A**symmetric **A**daptive **GNN** (abbreviated as A2GNN [1]). While most existing UGDA approaches use a shared architecture between source and target graph, we argue that it's not necessary as we thought. By introducing a single transformation layer on source graph and stacking multiple propagation layers on target graph, we are able to achieve superior performance. Furthermore, we also provide a comprehensive theoretical analysis for our proposed A2GNN via the graph filter theory and prove that the generalization gap of multi-layer GNNs depends on its propagation layers. With the designed asymmetric architecture, our proposed A2GNN exhibits a tighter target risk bound. Empirical results on multiple real-world datasets demonstrate that A2GNN outperforms recent state-of-the-art baselines on node classification tasks with different gains. We highlight our contributions as follows:

- **New Perspective**. Our work is the first to investigate the underlying generalization capability of GNNs in UGDA task. Remarkably, our method outperforms state-of-the-art graph domain adaptation approaches.

- **Simple but Effective Method**. Based on the aforementioned observations, we propose an embarrassingly simple yet effective method A2GNN with asymmetric network architecture. Comprehensive experimental results show the effectiveness of our proposed A2GNN.

- **Theoretical Analysis**. In addition to investigating the underlying generalization capability of GNNs, we also derive a domain adaptation bound for multi-layer GNNs. Furthermore, we prove that our proposed A2GNN yields a tighter error bound.

## Related Work

Unsupervised domain adaptation (UDA) involves the transfer of knowledge from a labelled source domain to an unlabelled target domain (Pan and Yang 2010; Wang and Deng 2018). A commonly used approach in domain adaptation is to minimize the domain discrepancy and learn domain-invariant representations (Ganin and Lempitsky 2015; Long et al. 2015, 2017; Pei et al. 2018; Tzeng et al. 2017), where great success has been achieved in computer vision and natural language processing communities (Blitzer, McDonald, and Pereira 2006; Venkateswara et al. 2017). However, these methods assume that their inputs are independently and identically distributed data, thus they are not appropriate

---

[1] https://github.com/Meihan-Liu/24AAAI-A2GNN

for tasks involving non-IID data, such as node classification in graph-structured data.

Recently, several approaches have been proposed to transfer knowledge across graph-structured non-IID data. These methods can be broadly categorized into two classes: minimizing pre-defined domain discrepancy metrics and adversarial learning techniques. For the pre-defined domain discrepancy metrics minimization models (Shen et al. 2021; Wu, He, and Ainsworth 2023), node representations are first obtained from the encoder, and then domain-invariant representations are learned by minimizing pre-defined domain discrepancy metrics, such as MMD (Gretton et al. 2012), graph subtree discrepancy (Wu, He, and Ainsworth 2023), etc. Instead of explicitly minimizing domain discrepancies, some methods incorporate the encoder with a domain classifier that predicts the domain from which the representation originates. Among them, UDAGCN (Wu et al. 2020) and AdaGCN (Dai et al. 2022) integrates graph convolution with adversarial training for graph transfer learning. ACDNE (Shen et al. 2020) utilizes Gradient Reversal Layer (GRL) (Ganin et al. 2016) to make node representation domain-invariant. ASN (Zhang et al. 2021) further improves node representations with attention mechanisms and disentanglement, where domain-private and domain-shared information are preserved. SpecReg (You et al. 2023) derives the generalization bound for one-layer GNNs and proposes spectral regularization to restrict the bound. Instead of focusing on minimizing domain discrepancy, in this paper, we investigate the GNN's underlying generalization capability behind its architecture and propose a simple yet effective method for unsupervised graph domain adaptation.

## Preliminaries

**Notations.** Consider an attributed graph $G = (\mathcal{V}, \mathcal{E})$ with $n$ nodes and $m$ edges. We denote $\mathbf{X} = \{x_v | v \in \mathcal{V}\} \in \mathbb{R}^{n \times d}$ as node attribute matrix, where $d$ is the dimension of node attributes. The adjacency matrix is denoted as $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $\mathbf{A}_{i,j} = 1$ means there exists an edge $e_{i,j} \in \mathcal{E}$ connecting node $v_i$ and $v_j$, and $\mathbf{A}_{i,j} = 0$ otherwise. For a node classification task, we denote the prediction targets by $\mathcal{Y} \in \mathbb{R}^{n \times C}$, where $C$ is the number of classes.

**Problem Definition.** Unsupervised Graph Domain Adaptation (UGDA) aims to transfer the knowledge from a labelled source graph to an unlabelled target graph. Formally, given graph $G_S = (\mathcal{V}_S, \mathcal{E}_S, \mathcal{Y}_S)$ and graph $G_T = (\mathcal{V}_T, \mathcal{E}_T)$ with the covariate shift assumption (Ben-David et al. 2006, 2009) that $\mathbb{P}_S(G) \neq \mathbb{P}_T(G)$ and $\mathbb{P}_S(Y|G) = \mathbb{P}_T(Y|G)$, where $\mathbb{P}_S$ and $\mathbb{P}_T$ are the probability distributions of the source and target domains, respectively. The target graph label $\mathcal{Y}^t$ is unknown. Our goal is to train a graph neural network (GNN) $h : \mathcal{G} \rightarrow \mathcal{Y}$ using the labeled source graph $G^s$ and the unlabeled target graph $G^t$ to accurately predict labels $\hat{y}^t$ on target graph $G^t$.

## Rethinking Propagation for UGDA

In this section, we start with an empirical analysis of GNNs in the context of UGDA, focusing particularly on the propagation and transformation process. Guided by the intriguing

Table 1: Performance on different architectures.

| Architecture | [PT] | [P]T | [T]P |
|---|---|---|---|
| Macro-F1 | 68.86 | 72.41 | 70.61 |

Table 2: Macro-F1 with/without propagation in graphs.

| target \ source | w/o propagation | w/ propagation |
|---|---|---|
| w/o propagation | 57.43 | 55.68 |
| w/ propagation | **75.43** | 72.41 |



Figure 2: The framework of existing models and our proposed A2GNN model.

insights obtained from our empirical analysis, we introduce a straightforward yet effective framework called A2GNN for unsupervised graph domain adaptation. Lastly, we offer a theoretical analysis of the proposed approach.

## Empirical Analysis

Although the Introduction presented several observations, this section provides more detailed elaboration and additional information. In general, most existing GNNs (Kipf and Welling 2017; Wu et al. 2019; Klicpera, Bojchevski, and Günnemann 2018) follow the neural message passing mechanism, which learns node representations by recursively aggregating information from neighborhood nodes. Specifically, the message passing procedure contains two major steps, namely *Transformation* (**T**) and *Propagation* (**P**) (Hamilton, Ying, and Leskovec 2017a; Zhang et al. 2022). Let $\mathbf{H}^{(l-1)}$ denote the node representations generated by the previous layer and we set $\mathbf{H}^0 = \mathbf{X}$ at the beginning of the GNN. The **P** operation can be donated as $\mathbf{P}(\mathbf{H}^{(l-1)}) = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}$, where $\hat{\mathbf{A}}$ is the normalized adjacency matrix with self-connections and $\mathbf{W}$ is the learnable weight matrix. The **T** operation applies non-linear transformations to the node representations, which can be donated as $\mathbf{T}(\mathbf{H}^{(l)}) = \sigma(\mathbf{H}^{(l)}\mathbf{W}^{(l)})$, where $\sigma$ is the non-linear activation function.

To investigate the role of transformation and propagation operations in UGDA task, we use the classical metric MMD (Gretton et al. 2012) to minimize the domain discrepancy on task $D \rightarrow A$. The implementation details is provided in Appendix. According to the results, we have the following interesting and inspiring findings:

**Observation 1: Propagation operation reflects the generalization capability of GNNs.** Since we have decoupled message passing procedure into operations **T** and **P**, existing GNNs can be broadly categorized into the following three types: [PT] which means stacking multiple **P** and **T** operations simultaneously; [P]T which indicates stacking multiple propagation operations without changing **T**; [T]P which represents stacking multiple transformation operations without changing **P**; [·] is the stacking operation. We first search the optimal architecture for each category and report their mean performance in Table 1. As we can see, both [P]T and [T]P architectures outperform the [PT] architecture, indicat-
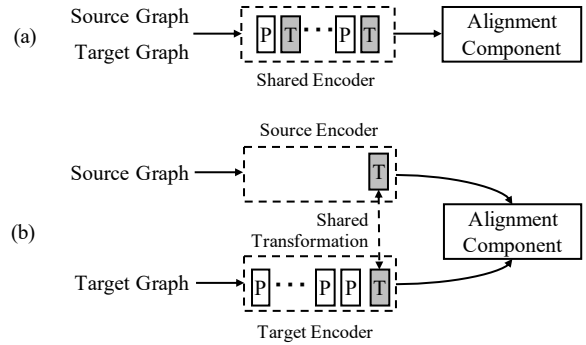
ing that the network architecture in UGDA indeed affects its performance. To further analyze the advantage of the [P]T and [T]P architectures, we evaluate its effects by increasing the number of **P** or **T** operations. The results, already depicted in Figure 1, demonstrate that stacking **P** operations improves its performance in both architectures, while stacking **T** operations has the opposite effect. Based on these findings, we conclude that (*i*) **P** *operation plays a pivotal role in determining the generalization ability of the model*, and (*ii*) *The* **T** *operations are only needed to be executed once*.

**Observation 2: Conducting propagation solely on target graph is of primary importance.** Although we have observed that propagation contributes to the improvement of generalization capability in GNNs, the impacts of propagation on both graphs remain unclear. To investigate this issue, we use the [P]T architecture and examine the effect of propagation on different domains while keeping the number of **T** operations unchanged. When comparing the second and first rows of Table 2, we find propagation on the target graph is the primary factor that affects the model's generalization capability, while conducting propagation solely on the source graph produces the worst performance. Interestingly, performing propagation solely on the target graph leads to better performance compared to performing it on both graphs (yielding a 1.85% increase in Macro-F1). These observations support the idea that propagation has different effects on different domains. Specifically, (*iii*) *stacking* **P** *operations on the target graph is particularly important*, and (*iv*) **P** *operations are not necessary on the source graph*.

## The Proposed Framework

The aforementioned two observations unveil the inherent generalization capability of GNNs. Building upon these insights, we propose an asymmetric graph domain adaptation model called A2GNN, which is embarrassingly simple yet extremely effective. The overall framework is illustrated in Figure 2(b), while Figure 2(a) demonstrates the symmetric architecture commonly employed by existing methods (Shen et al. 2020; Dai et al. 2022).

According to the analyses in the previous section, we find that it's not necessary to utilize a shared GNN architecture

between source and target graph, meanwhile stacking more propagation layers and using fewer transformation layers on target graph could lead to better performance. Therefore, we initialize our A2GNN with the following asymmetric architecture. For the source graph branch, we remove all the propagation layers, then it will degenerate to a non-linear MLP, i.e., $\mathbf{H}^s = \sigma(\mathbf{XW})$, where $\mathbf{X}$ is the node feature matrix and $\mathbf{W}$ denotes the learnable weight matrix. While for the target graph branch, we stack $k$ propagation layers with one transformation layer, i.e., $\mathbf{H}^t = \sigma(\mathbf{A}^k \mathbf{XW})$. Note that the learnable weight matrix $\mathbf{W}$ is shared across the source and target graph to prevent significant divergence between their representations.

The objective function of the unsupervised graph domain adaptation framework can be formulated as follows:

$$\mathcal{L}_{GDA} = \mathcal{L}_{cls} + \alpha \cdot \mathcal{L}_{align}, \tag{1}$$

where the first term $\mathcal{L}_{cls}$ indicates the cross-entropy loss function for the node classification task using source graph labels, while the second term $\mathcal{L}_{align}$ focuses on minimizing domain discrepancy to obtain domain-invariant representations. $\alpha$ serves as a trade-off parameter that balances the contributions of the two loss terms.

It is worth highlighting that our domain adaptation framework is flexible to incorporate various alignment loss functions. In our experiments, we utilize two classical alignment components: maximum mean discrepancy (MMD) (Gretton et al. 2012) for explicit alignment, represented as $\mathcal{L}_{GDA} = \mathcal{L}_{cls} + \alpha \cdot \mathcal{L}_{mmd}$, and adversarial training mechanism (Goodfellow et al. 2020) with gradient reversal layer (GRL) (Ganin et al. 2016) for implicit alignment, represented as $\mathcal{L}_{GDA} = \mathcal{L}_{cls} + \alpha \cdot \mathcal{L}_{adv}$. Furthermore, our domain adaptation framework offers the versatility to integrate various propagation mechanisms, and we will delve into the effects of different propagation mechanisms in the ablation study section.

## Theoretical Analysis

In this subsection, we aim to provide a theoretical explanation for the efficacy of our proposed A2GNN. Specifically, we first revisit the graph domain adaptation error bound formulated by (You et al. 2023), then we prove that the operation of transformation is independent of tightening the error bound. Lastly, we show how our proposed A2GNN narrows down the error bound.

**Theorem 1.** Suppose the feature extractor $f$ and the classifier $c$ is $K_f$-Lipschitz and $K_c$-Lipschitz continuous, where the Lipschitz norm $\|f\|_{\text{Lip}} = \max_{G_1, G_2} \frac{\|f(G_1) - f(G_2)\|_2}{\eta(G_1, G_2)} = K_f$ holds for some graph distance measure $\eta$. Let $\mathcal{H} := \{h : \mathcal{G} \to \mathcal{Y}\}$ be the set of bounded real-valued functions, i.e. $h = c \circ f \in \mathcal{H}$, with probability at least $1 - \delta$ the target risk $\epsilon_T(g, \hat{g})$ is bounded as in the following inequality (You et al. 2023):

$$\epsilon_T(h, \hat{h}) \leq \hat{\epsilon}_S(h, \hat{h}) + 2K_f K_c D(\mathbb{P}_S(G), \mathbb{P}_T(G)) + \sqrt{\frac{4d}{n^s} \log\left(\frac{en^s}{d}\right) + \frac{1}{n^s} \log\left(\frac{1}{\delta}\right)} + \omega, \tag{2}$$

where $\hat{\epsilon}_S(h, \hat{h})$ and $\hat{\epsilon}_T(h, \hat{h})$ represent the empirical source and target risks respectively. $\hat{h}$ denotes the labelling function. $\omega = \min_{\|c\|_{\text{Lip}} \leq K_c, \|f\|_{\text{Lip}} \leq K_f} \{\epsilon_S(h, h^*) + \epsilon_T(h, h^*)\}$ is the combined error of the ideal hypothesis $h^*$ that we expect to be small. $D(\cdot)$ indicates the distance metric, which formulates the source and target error functions in a Reproducing Kernel Hilbert Space.

Theorem 1 indicates that the generalization gap depends on the domain divergence $2K_f K_c D(\mathbb{P}_S(G), \mathbb{P}_T(G))$, where the Lipschitz constant $K_f$ is a conceptual property related to the model that needs to be instantiated (You et al. 2023). Therefore, one way to tighten the bound is to regularize the Lipschitz constant $K_f$ of the feature extractor $f$.

Next, we construct a GNN by composing a graph filter and nonlinear mapping that $f(G) = r(\sigma(\mathcal{S}(A) XW))$, where $r$ is the mean/sum/max readout function to pool node representations. $\mathcal{S}$ is the polynomial function that $\mathcal{S}(A) = \sum_{k=0}^{\infty} s_k A^k$. $W \in \mathcal{R}^{D \times D'}$ denotes the learnable weight matrix. The pointwise nonlinearity holds as $|\sigma(b) - \sigma(a)| \leq |b - a|, \forall a, b \in \mathcal{R}$. Based on these notations, a $l$-layer GNN can be constructed as $f(G) = r \circ f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}(G)$. The $l$-layer GNN Lipschitz constant $K_f$ is instantiated in the following lemma.

**Lemma 1.** Suppose that the edge perturbation is bounded that $\forall G_1, G_2 \in \mathcal{G}, EP = \|A_1 - P^* A_2 P^{* \text{T}}\|_{\text{F}} \leq \varepsilon$ with the optimal permutation $P^*$, and there exists an eigenvalue $\lambda^* \in \mathcal{R}$ to achieve the maximum $|\mathcal{S}(\lambda^*)| < \infty$. Assuming $\|X\|_{\text{op}} \leq 1$ and $\|W\|_{\text{op}} \leq 1$ ($\|\cdot\|_{\text{op}}$ stands for operator norm), we can then calculate the Lipschitz constant $l$-layer GNN as follows:

$$K_f = \max\left\{ T_1^{(l)} + \sum_{i=1}^{l-1} (\prod_{j=i+1}^{l} T_2^{(j)}) T_1^{(i)}, \prod_{i=0}^{l} T_2^{(i)} \right\}, \tag{3}$$

where $T_1^{(l)} = K_\lambda (1 + \tau\sqrt{n}) EP + \varepsilon \cdot \mathcal{O}(EP^2)$ and $T_2^{(l)} = |\mathcal{S}^{(l)}(\lambda_T)|$. $\tau$ stands for the eigenvector misalignment that can be bounded. $n = n_S + n_T$ means the total number of nodes in source and target graph. $\mathcal{O}(\cdot)$ is the remainder term defined in (Gama, Bruna, and Ribeiro 2019a), and $K_\lambda$ is the spectral Lipschitz constant that $\forall \lambda_i, \lambda_j, |\mathcal{S}(\lambda_i) - \mathcal{S}(\lambda_j)| \leq K_\lambda |\lambda_i - \lambda_j|$.

By integrating Theorem 1 and Lemma 1, we obtain the graph domain adaptation bound for a $l$-layer GNN. Lemma 1 indicates that (*i*) *decreasing the Lipschitz constant $K_f$ can tighten the target error bound in Theorem 1*, and (*ii*) *the transformation operation is independent of the GNN Lipschitz constant $K_f$*. Therefore, it is reasonable to employ a single transformation layer in conjunction with multiple propagation layers in our proposed A2GNN, which aligns with the empirical experiments in **Observation 1**. Based on the above theory, we next prove A2GNN has a tighter target error bound.

**Lemma 2.** Following the setting in Lemma 1, we use a single linear transformation in conjunction with $k$ propagation layers in A2GNN as $f_M^{(l)}(G) = r(\sigma(\mathcal{S}^{(l)}(A^k) XW))$. We rephrase $T_2$ as $T_2(x)$, $x$ is the number of propagation layers.

Table 3: Dataset Statistics.

| Types | Datasets | #Node | #Edge | #Feat | #Label |
|-------|----------|-------|-------|-------|--------|
| Citation | ACMv9 | 9,360 | 15,556 | 6,775 | 5 |
| | Citationv1 | 8,935 | 15,098 | | |
| | DBLPv7 | 5,484 | 8,117 | | |
| Social | Germany | 9,498 | 153,138 | 3,170 | 2 |
| | England | 7,126 | 35,324 | | |

For $\forall k > 1$, there exists:

$$\prod_{i=0}^{l} T_2^{(i)}(k) < \prod_{i=0}^{l} T_2^{(i)}(1), \qquad (4)$$

Based on Lemma 2, we state that the second term of $K_f$ can be smaller when we have $k$ ($k > 1$) propagation layers. Next, we show that the first term of $K_f$ can be smaller too.

**Lemma 3.** Following the setting in Lemma 2, if the source graph structure does not participate in the training procedure, then the corresponding GNN degenerates into $f_L(G) = r(\sigma(XW))$. We rephrase $T_1^{(l)}$ as $T_1^{(l)}(x)$, and $x$ indicates the number of propagation layers on source graph. For $\forall k > 1$, there exists:

$$T_1^{(l)}(0) + \sum_{i=1}^{l-1} (\prod_{j=i+1}^{l} T_2^{(j)}(k)) T_1^{(i)}(0) < T_1^{(l)}(1) + \sum_{i=1}^{l-1} (\prod_{j=i+1}^{l} T_2^{(j)}(1)) T_1^{(i)}(1). \qquad (5)$$

According to Lemma 3, we state that the first term of $K_f$ can be smaller when we conduct $k$ ($k > 1$) propagation layers on the target graph and only one transformation layer on the source graph. We denote the Lipschitz constant $K_f'$ of A2GNN as follows:

$$K_f' = \max \left\{ T_1^{(l)}(0) + \sum_{i=1}^{l-1} (\prod_{j=i+1}^{l} T_2^{(j)}(k)) T_1^{(i)}(0), \prod_{i=0}^{l} T_2^{(i)}(k) \right\}. \qquad (6)$$

With Lemma 2 and Lemma 3, we derive that $K_f' < K_f$. In other words, the Lipschitz constant $K_f'$ of A2GNN is smaller than that of the previous frameworks. This conclusion is in line with **Observation 2**. Combining this conclusion with Theorem 1, we prove that the graph domain error bound of A2GNN is tighter. Note that there is currently no theoretical evidence to support that more layers lead to tighter bound in Formula (6). Investigation on $k$ is shown in Fig 3(a). A more detailed proof process is provided in the Appendix.

## Experiments and Analyses

### Datasets

We conduct comprehensive experiments on three public citation networks and two social networks across a range of settings. These datasets are acquired from diverse sources and time periods with explicit covariate shifts. The citation networks[2] consist of three datasets: ACMv9, Citationv1, and DBLPv7. Each node represents a paper, and the edges indicate the citation relationships among them. **ACMv9 (A)** contains papers extracted from ACM between 2000 and 2010,

**Citationv1 (C)** comprises papers obtained from Microsoft Academic Graph prior to 2008, and **DBLPv7 (D)** encompasses papers collected from DBLP during the period from 2004 to 2008. Our objective is to classify all the papers into five distinct research topics, i.e., Databases, Artificial Intelligence, Computer Vision, Information Security, and Networking. As for social networks, we choose Twitch gamer networks[3], which are collected from different regions. Each node within these networks represents a user, and the connections between them indicate their friendships. We extract node features encompassing information about users' gaming preferences, geographical locations, streaming habits, etc. Specifically, we focus on the two largest graphs: **Germany (DE)** and **England (EN)**. In this scenario, users are classified into two groups based on whether they employ explicit language. For a comprehensive overview of these datasets, please refer to Table 3.

### Baselines

We compare our proposed A2GNN with three categories of baselines: (*i*) *Hypothesis transfer with unsupervised graph representation learning*: DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), node2vec (Grover and Leskovec 2016), and ANRL (Zhang et al. 2018). These methods first learn node representations in an unsupervised manner, and then the target graph node representations are evaluated using a classifier that has been trained on the source graph. (*ii*) *Source-only graph neural networks*: GCN (Kipf and Welling 2017), GAT (Velickovic et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017b), and GIN (Xu et al. 2018). This group of methods trains graph neural networks on the source graph in an end-to-end manner, thus they can be directly applied to target graph for evaluation. (*iii*) *Graph domain adaptation methods*: CDNE (Shen et al. 2021), AdaGCN (Dai et al. 2022), ACDNE (Shen et al. 2020), UDAGCN (Wu et al. 2020), ASN (Zhang et al. 2021), GRADE (Wu, He, and Ainsworth 2023), and SpecReg (You et al. 2023). Compared with the previous two groups, approaches in this group are specifically designed to address the graph domain adaptation problem, which are competitive baselines.

### Parameter Settings

Following previous works (Wu et al. 2020; Shen et al. 2020), we use all the source nodes and target nodes for model training. Among them, 80% of the labelled source nodes are utilized to provide supervision signals. The remaining labelled source nodes compose the validation set, and the evaluation is carried out on the target nodes. To ensure a fair comparison, we utilize the source codes released by the authors for each baseline, and then fine-tune their hyper-parameters to their optimal values. The node representation dimension is uniformly set as 128 for all the methods. Our proposed A2GNN is implemented with PyTorch (Paszke et al. 2019), where the learning rate and weight decay are searched in the range of $\{1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 5e^{-4}\}$. We consider two variants of our proposed model: $\text{A2GNN}_{mmd}$ which uses Gaussian kernel as the selected kernel for MMD (Shen et al.

Table 4: Unsupervised node classification on citation network. The best result is bold and the second best is underlined.

| Models | $D \to A$ | | $C \to A$ | | $A \to D$ | | $C \to D$ | | $A \to C$ | | $D \to C$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| DeepWalk | 19.83 | 26.23 | 19.33 | 21.94 | 19.87 | 25.94 | 17.51 | 22.57 | 17.72 | 21.05 | 22.76 | 29.46 |
| node2vec | 22.05 | 28.61 | 17.99 | 21.76 | 19.50 | 24.54 | 24.98 | 28.95 | 25.84 | 29.89 | 16.22 | 21.16 |
| ANRL | 19.12 | 29.56 | 22.04 | 31.84 | 23.33 | 29.54 | 22.71 | 25.90 | 20.93 | 30.31 | 18.25 | 25.99 |
| GAT | 43.95 | 52.93 | 42.14 | 50.37 | 41.36 | 53.80 | 45.25 | 55.85 | 43.64 | 57.13 | 50.04 | 55.52 |
| GraphSAGE | 57.31 | 59.22 | 64.69 | 65.22 | 61.80 | 64.82 | 66.86 | 69.96 | 69.14 | 71.40 | 64.90 | 67.85 |
| GIN | 56.50 | 58.98 | 59.48 | 60.46 | 50.49 | 59.10 | 63.48 | 66.27 | 62.49 | 68.61 | 63.21 | 69.25 |
| GCN | 59.42 | 63.35 | 70.39 | 70.58 | 65.29 | 69.05 | 71.37 | 74.53 | 74.78 | 77.38 | 69.79 | 74.17 |
| CDNE | 70.45 | 69.62 | 75.06 | 74.22 | 69.24 | 71.58 | 71.34 | 74.36 | 76.83 | 78.76 | 77.36 | 78.88 |
| UDAGCN | 55.89 | 58.16 | 67.22 | 66.80 | 64.83 | 66.95 | 69.46 | 71.77 | 60.33 | 72.15 | 61.12 | 73.28 |
| ACDNE | 72.64 | 71.29 | 74.79 | 73.59 | 73.59 | 76.24 | 75.74 | 77.21 | 80.09 | 81.75 | 78.83 | 80.14 |
| ASN | 71.49 | 70.15 | 73.17 | 72.74 | 71.40 | 73.80 | 73.98 | 76.36 | 77.81 | 80.64 | 75.17 | 78.23 |
| AdaGCN | 69.47 | 69.67 | 70.77 | 71.67 | 71.39 | 75.04 | 72.34 | 75.59 | 76.51 | 79.32 | 74.22 | 78.20 |
| GRADE | 59.35 | 63.72 | 69.34 | 69.55 | 63.03 | 68.22 | 70.02 | 73.95 | 72.52 | 76.04 | 69.32 | 74.32 |
| SpecReg | 72.34 | 71.01 | 73.15 | 72.04 | 73.98 | 75.93 | 73.64 | 75.74 | 78.83 | 80.55 | 77.78 | 79.04 |
| **A2GNN**$_{adv}$ | **73.81** | **72.17** | **76.64** | **75.22** | **74.03** | 75.42 | **75.82** | **77.32** | 78.60 | 80.27 | 78.18 | 80.08 |
| Improv.(%) | +1.17 | +0.88 | +1.58 | +1.00 | +0.05 | -0.82 | +0.08 | +0.11 | -1.49 | -1.48 | -0.65 | -0.06 |
| **A2GNN**$_{mmd}$ | **75.69** | **74.12** | **77.57** | **76.15** | **75.78** | **77.43** | **77.04** | **78.13** | **81.30** | **82.64** | **79.74** | **81.54** |
| Improv.(%) | +3.05 | +2.83 | +2.52 | +1.93 | +1.80 | +1.19 | +1.30 | +0.92 | +1.21 | +0.89 | +0.91 | +1.40 |

2021) and A2GNN$_{adv}$ which employs adversarial training for invariant representation learning (Shen et al. 2020). The experiments are repeated five times, and we report the mean performance in terms of Micro-F1 and Macro-F1 scores.

## Main Results

Table 4 shows the node classification performance on citation network, and we have the following observations: (*i*) Our proposed A2GNN outperforms recent state-of-the-art baselines with different gains. On average, A2GNN$_{mmd}$ achieves 1.80% improvement in Macro-F1 and 1.53% improvement in Micro-F1 scores under various settings, while A2GNN$_{adv}$ also yields comparable results. (*ii*) The graph domain adaptation methods outperform the vanilla graph neural networks and the hypothesis transfer approaches, indicating that considering the domain discrepancy across networks is crucial. Furthermore, it should be noted that A2GNN$_{mmd}$ surpasses A2GNN$_{adv}$ by varying margins. This divergence can be attributed to the fact that MMD offers explicit signals, whereas adversarial training solely provides implicit signals.

Similarly, we can draw similar conclusions when it comes to social networks. The results are shown in Table 5. In general, our proposed A2GNN outperforms or shows comparable performance among all the settings. More specifically, A2GNN$_{adv}$ outperforms all the baselines on all tasks. A2GNN$_{mmd}$ obtains the highest scores on both Macro-F1 and Micro-F1 for the $EN \to DE$ task, and it ranks second in terms of Micro-F1 for the $DE \to EN$ task. On average, A2GNN$_{mmd}$ achieves 1.88% improvement in Macro-F1 and 1.13% improvement in Micro-F1, while A2GNN$_{adv}$ achieves a 2.54% increase in Macro-F1 and 1.93% improvement in Micro-F1. Moreover, we also observe that some domain adaptation methods are outperformed by source-only

graph neural networks, which is known as negative transfer. This can be attributed to the heterophilic nature of these social graphs, and it introduces significant challenges for domain adaptation tasks. In conclusion, our proposed simple yet effective A2GNN consistently exhibits impressive performance on different types of graphs, thus verifying its capability in graph domain adaptation task.

## Ablation Study

**The impact of the number of propagation layers $k$.** As we discussed in previous sections, Lemma 2 and Lemma 3 have proved that propagation on the target graph for $k$ ($k > 1$) times can tighten the target error bound. Therefore, we conducted some experiments to analyze the sensitivity of $k$, which is shown in Figure 3(a). On the one hand, utilizing a small number of $k$ ($k < 5$) can significantly improve its performance. On the other hand, varying values of $k$ yield distinct effects across various tasks. For the $D \to A$ task, as the value of $k$ increases, its performance improves correspondingly. In contrast, for the $D \to C$ task, optimal performance is observed within the range of 5 to 15, followed by a gradual decline when $k$ surpasses 15. This decline is particularly pronounced in terms of Macro-F1 scores. In conclusion, the empirical results align with our theoretical analyses.

**The influence of different propagation schemes.** To further investigate whether our proposed framework can be employed to various types of propagation schemes, we replace the widely used symmetric normalized transition matrix $\mathbf{P}_{sym} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ in GCN with three commonly used transition matrices (Hassani and Khasahmadi 2020): (1) $\mathbf{P}_{no\text{-}loop} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, i.e., removing the self-loops; (2) $\mathbf{P}_{rw} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$, i.e., the random walk matrix; (3) $\mathbf{P}_{diff} =$

Table 5: Unsupervised node classification on social network. The best result is bold and the second best is underlined.

| Models | DE→EN | | EN→DE | |
|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| Node2vec | 46.96 | 52.64 | 50.10 | 54.61 |
| DeepWalk | 46.54 | 52.18 | 49.97 | 55.08 |
| GAT | 49.50 | 54.84 | 40.08 | 43.65 |
| GCN | 54.55 | 54.77 | 51.04 | 62.03 |
| GIN | 49.91 | 52.39 | 44.26 | 55.26 |
| UDAGCN | <u>58.19</u> | <u>59.74</u> | 56.35 | 58.69 |
| ACDNE | 56.31 | 58.08 | <u>57.92</u> | 58.79 |
| ASN | 51.21 | 55.45 | 45.90 | 60.45 |
| AdaGCN | 35.30 | 54.56 | 31.18 | 40.22 |
| GRADE | 56.38 | 56.40 | 56.83 | 61.18 |
| SpecReg | 50.30 | 56.43 | 46.13 | <u>61.45</u> |
| **A2GNN**$_{mmd}$ | **58.44** | 59.54 | **61.42** | **63.90** |
| Improv.(%) | +0.25 | -0.20 | +3.50 | +2.45 |
| **A2GNN**$_{adv}$ | **59.16** | **59.94** | **62.03** | **65.11** |
| Improv.(%) | +0.97 | +0.20 | +4.11 | +3.66 |
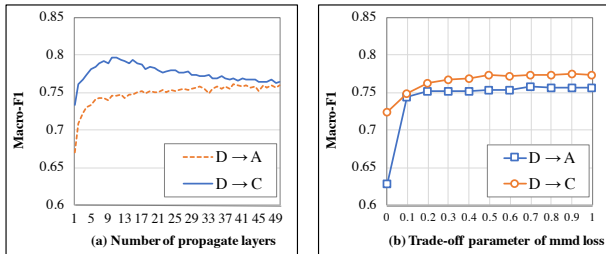


Figure 3: (a) shows the sensitivity of the num of propagation layers $k$. (b) shows the sensitivity of trade-off parameter $\alpha$.

$\sum_{p=0}^{\infty} \frac{1}{e \cdot p!} \left( \tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \right)^p$, i.e., the heat kernel diffusion matrix, where $p$ is set as 10 by default. The results are presented in Table 6. We observe that the relative performance stays at almost the same level when removing self-loop, which verifies our observation that deep neighbor information is more important than local information. The performance of random walk matrix drops on $D \to C$ task while staying at almost the same level on $D \to A$ task. The possible reason is that random walk matrix have incomplete neighbor information, which affects the model's ability to capture the structural information and thus affects its generalization capability. We also note a relatively deteriorated performance achieved by the diffusion transition matrix. This can be attributed to its emphasis on global information, neglecting the crucial local neighborhood information for node classification task.

**The effectiveness of different modules.** Table 7 demonstrates that the addition of each module contributes to the final results without any performance degradation. The first line, $\mathcal{L}_{cls}$, only considers the source domain information and yields unsatisfactory performance, indicating that the model cannot be effectively applied to the target graph due to the existence of domain shifts. The second line incorporates the

Table 6: Performance with different transition matrices in the propagation layer of A2GNN.

| Transition matrix | $D \to A$ | | $D \to C$ | |
|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| $\mathbf{P}_{sym}$ | 75.69 | 74.12 | 79.74 | 81.54 |
| $\mathbf{P}_{no\text{-}loop}$ | 75.96 | 74.51 | 78.69 | 80.39 |
| $\mathbf{P}_{rw}$ | 75.77 | 74.04 | 78.10 | 79.72 |
| $\mathbf{P}_{diff}$ | 74.63 | 73.18 | 77.48 | 79.17 |

Table 7: Performance contribution of each part in A2GNN

| Modules | $D \to A$ | | $D \to C$ | |
|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| $\mathcal{L}_{cls}$ | 58.50 | 61.11 | 66.78 | 70.43 |
| + $\mathcal{L}_{mmd}$ | 67.09 | 65.85 | 72.96 | 74.83 |
| + $k$ propagation layers | 72.41 | 72.10 | 78.00 | 80.47 |
| + asymmetric architecture | **75.69** | **74.12** | **79.74** | **81.54** |

reduction of domain discrepancy by adding commonly used MMD constraint $\mathcal{L}_{mmd}$, highlighting the importance of reducing domain discrepancy. Next, we show how our model utilizes the inherent generalization capability of GNNs without adding additional trainable parameters. The third line illustrates that performing propagation for $k$ times enhances the generalization capability of GNN, which is consistent with Lemma 1. The fourth line demonstrates that the asymmetric architecture of our proposed A2GNN can further boost its performance, which is in line with Lemma 2 and Lemma 3. In conclusion, our proposed A2GNN achieves the best performance, which validates the effectiveness of combining all the aforementioned modules.

**The sensitivity of the trade-off parameter** $\alpha$. To assess the sensitivity of the trade-off parameter $\alpha$, we conduct experiments on $D \to A$ and $D \to C$ tasks, as depicted in Figure 3(b). The results indicate that A2GNN consistently performs well across the range of $[0.0, 1.0]$, suggesting its robustness in the optimization process. When $\alpha = 0$, we solely use the source graph, and the overall loss function is reduced to $\mathcal{L} = \mathcal{L}_{cls}$. The drop in performance highlights the importance of minimizing domain discrepancy. Thus, it's necessary to minimize the distribution discrepancy. We also have provided more detailed ablation studies in Appendix.

## Conclusion

In this paper, we study the problem of unsupervised graph domain adaptation on node classification task. We start by conducting a series of experiments to reveal the inherent generalization capability of GNNs. These observations provide important insights into designing a simple but effective model A2GNN for UGDA. To take a step further, we derive the theoretical analysis of the error bound of multi-layer GNNs and show how A2GNN tightens the error bound. Extensive experiments on real-world datasets demonstrate the effectiveness of our A2GNN framework. In the future, we

will extend our model to more complicated scenarios, such as source-free unsupervised graph domain adaptation and open-set graph domain adaptation.

## Acknowledgments

## References

Arghal, R.; Lei, E.; and Bidokhti, S. S. 2022. Robust graph neural networks via probabilistic lipschitz constraints. In *Learning for Dynamics and Control Conference*, 1073–1085. PMLR.

Ben-David, S.; Blitzer, J.; Crammer, K.; Kulesza, A.; Pereira, F. C.; and Vaughan, J. W. 2009. A theory of learning from different domains. *Machine Learning*, 79: 151–175.

Ben-David, S.; Blitzer, J.; Crammer, K.; and Pereira, F. C. 2006. Analysis of Representations for Domain Adaptation. In *NIPS*.

Blitzer, J.; McDonald, R.; and Pereira, F. 2006. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, 120–128.

Dai, Q.; Wu, X.-M.; Xiao, J.; Shen, X.; and Wang, D. 2022. Graph Transfer Learning via Adversarial Domain Adaptation with Graph Convolution. *TKDE*.

Gama, F.; Bruna, J.; and Ribeiro, A. 2019a. Stability Properties of Graph Neural Networks. *IEEE Transactions on Signal Processing*, 68: 5680–5695.

Gama, F.; Bruna, J.; and Ribeiro, A. 2019b. Stability Properties of Graph Neural Networks. *IEEE Transactions on Signal Processing*, 68: 5680–5695.

Ganin, Y.; and Lempitsky, V. S. 2015. Unsupervised Domain Adaptation by Backpropagation. *ArXiv*, abs/1409.7495.

Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. S. 2016. Domain-Adversarial Training of Neural Networks. In *JMLR*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Gretton, A.; Borgwardt, K. M.; Rasch, M. J.; Schölkopf, B.; and Smola, A. 2012. A Kernel Two-Sample Test. *JMLR*, 13: 723–773.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. *KDD*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017a. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017b. Inductive Representation Learning on Large Graphs. In *NIPS*.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, 4116–4126. PMLR.

Kipf, T.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*.

Li, S.; Zhou, J.; Xu, T.; Huang, L.; Wang, F.; Xiong, H.; Huang, W.; Dou, D.; and Xiong, H. 2021. Structure-aware interactive graph neural networks for the prediction of protein-ligand binding affinity. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 975–985.

Liu, S.; Li, T.; Feng, Y.; Tran, N.; Zhao, H.; Qiang, Q.; and Li, P. 2023. Structural Re-weighting Improves Graph Domain Adaptation. In *International Conference on Machine Learning*.

Long, M.; Cao, Y.; Wang, J.; and Jordan, M. I. 2015. Learning Transferable Features with Deep Adaptation Networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, 97–105. JMLR.org.

Long, M.; Cao, Z.; Wang, J.; and Jordan, M. I. 2017. Conditional Adversarial Domain Adaptation. In *Neural Information Processing Systems*.

Lu, B.; Gan, X.; Jin, H.; Fu, L.; Wang, X.; and Zhang, H. 2022. Make more connections: Urban traffic flow forecasting with spatiotemporal adaptive gated graph convolution network. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1–25.

Pan, S. J.; and Yang, Q. 2010. A Survey on Transfer Learning. *TKDE*, 22: 1345–1359.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

Pei, Z.; Cao, Z.; Long, M.; and Wang, J. 2018. Multi-Adversarial Domain Adaptation. In *AAAI Conference on Artificial Intelligence*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: online learning of social representations. *KDD*.

Shen, X.; Dai, Q.; Chung, K. F.-L.; Lu, W.; and Choi, K.-S. T. 2020. Adversarial Deep Network Embedding for Cross-network Node Classification. *AAAI*.

Shen, X.; Dai, Q.; Mao, S.; lai Chung, F.; and Choi, K.-S. T. 2021. Network Together: Node Classification via Cross-Network Deep Network Embedding. *IEEE Transactions on Neural Networks and Learning Systems*, 32: 1935–1948.

Tzeng, E.; Hoffman, J.; Saenko, K.; and Darrell, T. 2017. Adversarial Discriminative Domain Adaptation. *CVPR*, 2962–2971.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio', P.; and Bengio, Y. 2018. Graph Attention Networks.

Venkateswara, H.; Eusebio, J.; Chakraborty, S.; and Panchanathan, S. 2017. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5018–5027.

Wang, M.; and Deng, W. 2018. Deep Visual Domain Adaptation: A Survey. *Neurocomputing*, 312: 135–153.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, 6861–6871. PMLR.

Wu, J.; He, J.; and Ainsworth, E. 2023. Non-iid transfer learning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 9, 10342–10350.

Wu, M.; Pan, S.; Zhou, C.; Chang, X.; and Zhu, X. 2020. Unsupervised Domain Adaptive Graph Convolutional Networks. *Proceedings of The Web Conference 2020*.

Xia, W.; Li, Y.; Wu, J.; and Li, S. 2021. DeepIS: Susceptibility estimation on social networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 761–769.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks?

Yang, C.; Wu, Q.; Wang, J.; and Yan, J. 2022. Graph neural networks are inherently good generalizers: Insights by bridging gnns and mlps.

You, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2023. Graph Domain Adaptation via Theory-Grounded Spectral Regularization. In *The Eleventh International Conference on Learning Representations*.

Zhang, W.; Sheng, Z.; Yin, Z.; Jiang, Y.; Xia, Y.; Gao, J.; Yang, Z.; and Cui, B. 2022. Model Degradation Hinders Deep Graph Neural Networks. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Zhang, X.; Du, Y.; Xie, R.; and Wang, C. 2021. Adversarial Separation Network for Cross-Network Node Classification. *CIKM*.

Zhang, Z.; Yang, H.; Bu, J.; Zhou, S.; Yu, P.; Zhang, J.; Ester, M.; and Wang, C. 2018. ANRL: Attributed Network Representation Learning via Deep Neural Networks. In *IJCAI*.

Zhu, Q.; Jiao, Y.; Ponomareva, N.; Han, J.; and Perozzi, B. 2023. Explaining and Adapting Graph Conditional Shift. arXiv:2306.03256.

# Appendix

## Proof for Lemma 1

We construct a GNN by composing a graph filter and nonlinear mapping that $f(G) = r(\sigma(\mathcal{S}(A)XW))$, where $r$ is the mean/sum/max readout function to pool node representations. $\mathcal{S}$ is the polynomial function that $\mathcal{S}(A) = \sum_{k=0}^{\infty} s_k A^k$. $W \in \mathcal{R}^{D \times D'}$ denotes the learnable weight matrix. The pointwise nonlinearity holds as $|\sigma(b) - \sigma(a)| \leq |b - a|, \forall a, b \in \mathcal{R}$. Based on these notations, a $l$-layer GNN can be constructed as $f(G) = r \circ f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}(G)$. The $l$-layer GNN Lipschitz constant $K_f$ is instantiated in the following lemma.

**Lemma 1.** Suppose that the edge perturbation is bounded that $\forall G_1, G_2 \in \mathcal{G}$, $EP = \left\| A_1 - P^* A_2 P^{* \, \mathrm{T}} \right\|_{\mathrm{F}} \leq \varepsilon$ with the optimal permutation $P^*$, and there exists an eigenvalue $\lambda^* \in \mathcal{R}$ to achieve the maximum $|\mathcal{S}(\lambda^*)| < \infty$. Assuming $\|X\|_{\mathrm{op}} \leq 1$ and $\|W\|_{\mathrm{op}} \leq 1$ ( $\| \cdot \|_{\mathrm{op}}$ stands for operator norm), we can then calculate the Lipschitz constant $l$-layer GNN as follows:

$$K_f = \max \left\{ T_1^{(l)} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) T_1^{(i)}, \prod_{i=0}^{l} T_2^{(i)} \right\},$$

where $T_1^{(l)} = K_\lambda^{(l)} (1 + \tau \sqrt{n}) EP + \varepsilon \cdot \mathcal{O}(EP^2)$ and $T_2^{(l)} = \left| \mathcal{S}^{(l)}(\lambda_T) \right|$. $\tau$ stands for the eigenvector misalignment that can be bounded. $n = n_S + n_T$ means the total number of nodes in source and target graph. $\mathcal{O}(\cdot)$ is the remainder term defined in (Gama, Bruna, and Ribeiro 2019a), and $K_\lambda$ is the spectral Lipschitz constant that $\forall \lambda_i, \lambda_j, |\mathcal{S}(\lambda_i) - \mathcal{S}(\lambda_j)| \leq K_\lambda |\lambda_i - \lambda_j|$.

*Proof.* Denote the optimal permutation matrix for $G_1, G_2$ as $P^*$, $T_{1,1} = (1 + \tau \sqrt{n}) EP$, $T_{1,2} = \mathcal{O}(EP^2)$, $T_{1,0} = K_\lambda T_{1,1} + T_{1,2}$, and $T_{2,0}^{(l)} = \max(|\mathcal{S}^{(l)}(\Lambda_2)|)$. We first prove the following inequality for a $l$-layer GNN, which compute the difference of the encoder outputs:

$$\|f(G_1) - f(G_2)\|_2 \leq T_{1,0}^{(l)} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) T_{1,0}^{(i)} + \prod_{i=0}^{l} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}}. \tag{7}$$

We use mathematical induction to prove this inequality. (*i*) *Base Case*: Formula 7 holds true for $l = 1$ (You et al. 2023). (*ii*) *Inductive Hypothesis*: Assume that the theorem holds true for $l = k$, where $k$ is an integer greater than or equal to 1. (*iii*) *Inductive Step*: We aim to prove that the theorem also holds true for $l = k + 1$:

$$
\begin{aligned}
&\|f(G_1) - f(G_2)\|_2 \\
&= \left\| \left( r \circ f^{(k+1)} \right) \circ f^{(k)}(G_1) - \left( r \circ f^{(k+1)} \right) \circ f^{(k)}(G_2) \right\|_2 \\
&\overset{(a)}{\leq} K_\lambda^{(k+1)} T_{1,1} + T_{1,2} + T_{2,0}^{(k+1)} \left\| X_1^{(k)} - P^* X_2^{(k)} \right\|_{\mathrm{F}} \\
&\overset{(b)}{\leq} K_\lambda^{(k+1)} T_{1,1} + T_{1,2} + T_{2,0}^{(k+1)} \left( T_{1,0}^{(k)} + \sum_{i=1}^{k-1} ( \prod_{j=i+1}^{k} T_{2,0}^{(j)}) T_{1,0}^{(i)} + \prod_{i=0}^{l} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}} \right) \\
&= T_{1,0}^{(k+1)} + \sum_{i=1}^{k} ( \prod_{j=i+1}^{k+1} T_{2,0}^{(j)}) T_{1,0}^{(i)} + \prod_{i=0}^{k+1} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}},
\end{aligned}
\tag{8}
$$

where (a) has been proven in (You et al. 2023) and (b) is the inductive hypothesis. By proving the base case and the inductive step, we can conclude that according to mathematical induction, Formula 8 holds true for all integers greater than or equal to the initial value.

Next, following the same spirit in (You et al. 2023), to calculate the Lipschitz constant $K_f$, we assure the inequality:

$$
\begin{aligned}
&\|f(G_1) - f(G_2)\|_2 \\
&\leq T_{1,0}^{(l)} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) T_{1,0}^{(i)} + \prod_{i=0}^{l} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}} \\
&\overset{(a)}{=} K_\lambda^{(l)} T_{1,1} + T_{1,2} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) K_\lambda^{(i)} T_{1,1} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) T_{1,2} + \prod_{i=0}^{l} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}} \\
&= \left( K_\lambda^{(l)} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) K_\lambda^{(i)} \right) T_{1,1} + \left( 1 + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) \right) T_{1,2} + \prod_{i=0}^{l} T_{2,0}^{(i)} \|X_1 - P^* X_2\|_{\mathrm{F}} \\
&\leq K_f \eta(G_1, G_2),
\end{aligned}
\tag{9}
$$

where (a) is due to $T_{1,0}^{(l)} = K_\lambda^{(l)} T_{1,1} + T_{1,2}$. Following (You et al. 2023), we define the commonly-used matching distance as $\eta(G_1, G_2) = \min_{P \in \Pi} \left\{ \|X_1 - PX_2\|_F + \|A_1 - PA_2P^\top\|_F \right\}$ (Gama, Bruna, and Ribeiro 2019b; Arghal, Lei, and Bidokhti 2022). Then, above inequality is:

$$
\left( (1 + \tau\sqrt{n}) K_\lambda^{(l)} + (1 + \tau\sqrt{n}) \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) K_\lambda^{(i)} - K_f \right) EP + \left( 1 + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) \right) T_{1,2}
$$
$$
+ \left( \prod_{i=0}^{l} T_2^{(i)} - K_f \right) \|X_1 - P^* X_2\|_F \le 0, \tag{10}
$$

which is necessary for:

$$
\left( (1 + \tau\sqrt{n}) K_\lambda^{(l)} + (1 + \tau\sqrt{n}) \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) K_\lambda^{(i)} - K_f \right) EP + \left( 1 + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) \right) T_{1,2} \le 0,
$$
$$
\left( \prod_{i=0}^{l} T_2^{(i)} - K_f \right) \|X_1 - P^* X_2\|_F \le 0, \tag{11}
$$

which is equivalent to:

$$
K_f \ge K_\lambda^{(l)} T_{1,1} + \varepsilon \cdot T_{1,2} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_{2,0}^{(j)}) \left( K_\lambda^{(i)} T_{1,1} + \varepsilon \cdot T_{1,2} \right),
$$
$$
= T_1^{(l)} + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}) T_1^{(i)} \tag{12}
$$
$$
K_f \ge \prod_{i=0}^{l} T_2^{(i)}
$$

Let $K_f$ takes the larger value between them, we complete the proof.

## Proof for Lemma 2

**Lemma 2.** Following the setting in Lemma 1, we use a single linear transformation in conjunction with $k$ propagation layers in A2GNN as $f_M^{(l)}(G) = r \left( \sigma \left( \mathcal{S}^{(l)} \left( A^k \right) XW \right) \right)$. We rephrase $T_2$ as $T_2(x)$, $x$ is the number of propagation layers. For $\forall k > 1$, there exists:

$$
\prod_{i=0}^{l} T_2^{(i)}(k) < \prod_{i=0}^{l} T_2^{(i)}(1),
$$

*Proof.* $T_2^{(l)} = \left| \mathcal{S}^{(l)}(\lambda_T) \right|$ equals the spectral radius (You et al. 2023), which is the maximum absolute eigenvalue of the adjacency matrix $A$ and provides a measure of the smoothness of the graph. According to the consistency of matrix norm, the following inequality holds:

$$
0 < T_2^{(l)}(k) \le \|A^p\|^{\frac{1}{p}} \le \|A\| = T_2^{(l)}(1) \tag{13}
$$

When $k = 1$, the equal sign holds. Thus, for any $l > 1$, $T_2^{(l)}(k) < T_2^{(l)}(1)$ holds, then we complete the proof. Intuitively, as the number of propagations increases, the signals on the similar nodes become more consistent, and the graph becomes smoother. This leads to a decrease in the spectral radius and the eigenvalues of the adjacency matrix.

## Proof for Lemma 3

**Lemma 3.** Following the setting in Lemma 2, if the source graph structure does not participate in the training procedure, then the corresponding GNN degenerates into $f_L(G) = r(\sigma(XW))$. We rephrase $T_1^{(l)}$ as $T_1^{(l)}(x)$, and $x$ indicates the number of propagation layers on source graph. For $\forall k > 1$, there exists:

$$
T_1^{(l)}(0) + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}(k)) T_1^{(i)}(0) < T_1^{(l)}(1) + \sum_{i=1}^{l-1} ( \prod_{j=i+1}^{l} T_2^{(j)}(1)) T_1^{(i)}(1).
$$

*Proof.* As the Lipschitz constant $K_f = \max_{G_1, G_2} \frac{\|f(G_1) - f(G_2)\|_2}{\eta(G_1, G_2)}$ and there exist data distribution shift between the source graph and target graph. Thus, we suppose the distance between nodes belonging to the same graph is smaller than the distance between different graphs. The following two inequality holds:

$$
\begin{aligned}
\|f(G_{S,1}) - f(G_{T,2})\|_2 &> \|f(G_{S,1}) - f(G_{S,2})\|_2, \\
\|f(G_{S,1}) - f(G_{T,2})\|_2 &> \|f(G_{T,1}) - f(G_{T,2})\|_2,
\end{aligned}
\tag{14}
$$

where $G_{S,i}$ and $G_{T,i}$ is the ego-graph from source and target graph respectively. Thus, $EP$ can be rephrased as $EP = \left\| A_{S,1} - P^* A_{T,2} P^{* \mathrm{T}} \right\|_{\mathrm{F}}$. If the source graph structure does not participate in the training procedure, then the corresponding GNN degenerates into $f_L(G) = r(\sigma(XW)) = r(\sigma(IXW))$. In this case, we represent $EP$ as $EP' = \left\| I - P^* A_T P^{* \mathrm{T}} \right\|_{\mathrm{F}}$. Then, the inequality holds:

$$
\begin{aligned}
EP'^2 &= \left\| I - P^* A_T P^{* \mathrm{T}} \right\|_{\mathrm{F}}^2 \\
&< \left\| I - P^* A_T P^{* \mathrm{T}} \right\|_{\mathrm{F}}^2 + \left\| A_S - I \right\|_{\mathrm{F}}^2 + Re(< A_S - I, I - P^* A_T P^{* \mathrm{T}} >) \\
&= \left\| A_S - I + I - P^* A_T P^{* \mathrm{T}} \right\|_{\mathrm{F}}^2 \\
&= \left\| A_S - P^* A_T P^{* \mathrm{T}} \right\|_{\mathrm{F}}^2 = EP^2.
\end{aligned}
\tag{15}
$$

For $EP'$ and $EP$ both greater than 0, we can derive $EP' < EP$. Next, for $T_1^{(l)}(1) = K_\lambda^{(l)} (1 + \tau \sqrt{n}) EP + \varepsilon \cdot \mathcal{O}(EP^2)$ and $T_1^{(l)}(0) = K_\lambda^{(l)} (1 + \tau \sqrt{n}) EP' + \varepsilon \cdot \mathcal{O}(EP'^2)$, we can derive $T_1^{(l)}(1) > T_1^{(l)}(0)$. Finally, substitute $T_1^{(l)}(0)$ and $T_2^{(l)}(k)$ into Formula 3, the proof complete.

## Detialed of Minimize Domain Discrepancy

There are two common approaches to reducing domain shift in domain adaptation. One approach involves explicitly measuring the discrepancy between the source and target graphs. This can be achieved using techniques such as maximum mean discrepancy (MMD) (Gretton et al. 2012). The other approach involves employing an adversarial objective with a domain discriminator in an implicit manner. By minimizing the domain classification loss while maximizing the classification accuracy, the main task network learns to produce domain-invariant features that can be used for classification in both the source and target domains.

**Maximum Mean Discrepancy Eestimation.** MMD quantifies the discrepancy between the two domains by comparing their mean embeddings in the reproducing kernel Hilbert space (RKHS). To calculate MMD, we first map the samples from each domain to their corresponding feature space using a kernel function. Then, we calculate the mean embeddings of the two distributions. The MMD is then defined as the distance between these mean embeddings, representing the discrepancy between the domains. The squared value of MMD is estimated as follows:

$$
\begin{aligned}
\mathcal{L}_{mmd} = {}& \frac{1}{n_S^2} \sum_{i=1}^{n_S} \sum_{j=1}^{n_S} k_l \left( \phi_l \left( \boldsymbol{x}_i^s \right), \phi_l \left( \boldsymbol{x}_j^s \right) \right) \\
& + \frac{1}{n_T^2} \sum_{i=1}^{n_T} \sum_{j=1}^{n_T} k_l \left( \phi_l \left( \boldsymbol{x}_i^t \right), \phi_l \left( \boldsymbol{x}_j^t \right) \right) \\
& - \frac{2}{n_S n_T} \sum_{i=1}^{n_S} \sum_{j=1}^{n_T} k_l \left( \phi_l \left( \boldsymbol{x}_i^s \right), \phi_l \left( \boldsymbol{x}_j^t \right) \right),
\end{aligned}
\tag{16}
$$

where $n_S$ and $n_T$ is the number of nodes of source and target graph. And $k$ denotes the selected kernel. $\boldsymbol{x}_i^s$ and $\boldsymbol{x}_i^t$ denote the feature of source and target node respectively. By minimizing the MMD value, we aim to align the distributions of the source and target graphs to reduce the domain shift.

**Adversarial Training.** The goal of adversarial training, similar to MMD, is to learn domain-invariant representations without bias towards specific domains. By incorporating the domain discriminator into the training process, the encoder is encouraged to learn features that are indistinguishable between the source and target domains, aligning their distributions in the learned representation space. In the most routine procedure (Shen et al. 2020; Zhang et al. 2021; You et al. 2023) for implementing adversarial training, the domain classification loss is defined as follows:

$$
\mathcal{L}_{adv} = -\frac{1}{n} \sum_{v_i \in V} (1 - d_i) \log \left( 1 - \hat{d}_i \right) + d_i \log \left( \hat{d}_i \right)
\tag{17}
$$

where $d_i$ is the ground-truth domain label of $v_i$. And $\hat{d}_i$ represents the predicted domain probability of $v_i$. $n = n_S + n_T$ means the total number of nodes for the source graph and the target graph. $V = \{V^s \cup V^t\}$ is the collection of source and target nodes.
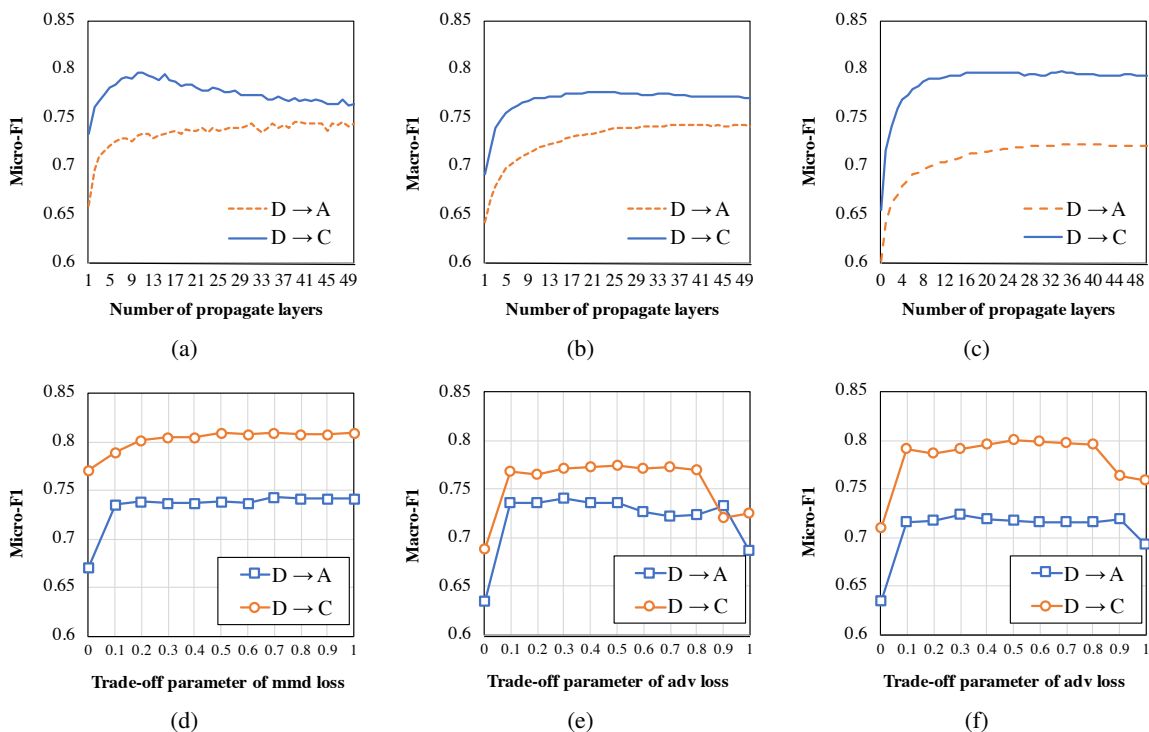
Figure 4: (a-c) shows the sensitivity of the number of propagation layers $k$. (a) is the Micro-F1 of A2GNN$_{mmd}$. (b-c) is the Macro-F1 and Micro-F1 of A2GNN$_{adv}$ respectively. (d-f) shows the sensitivity of trade-off parameter $\alpha$. (d) is the Micro-F1 of A2GNN$_{mmd}$. (e-f) is the Macro-F1 and Micro-F1 of A2GNN$_{adv}$ respectively.

Table 8: Performance contribution of each part in A2GNN$_{adv}$

| Modules | $D \to A$ | | $D \to C$ | |
|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| $\mathcal{L}_{cls}$ | 63.51 | 63.41 | 68.89 | 70.93 |
| + $\mathcal{L}_{adv}$ | 64.19 | 64.13 | 69.59 | 72.01 |
| + $k$ propagation layers | 66.95 | 68.09 | 75.56 | 79.20 |
| + asymmetric architecture | **73.81** | **72.17** | **78.18** | **80.08** |

In the adversarial training framework, a Gradient Reversal Layer (GRL) (Ganin et al. 2016) is often inserted between the encoder and the domain discriminator to facilitate simultaneous updates during backpropagation. The GRL operates by reversing the sign of the gradients during backpropagation. Overall, the adversarial approach leverages the domain discriminator to implicitly align the node representations across domains, allowing the model to adapt and perform well in both the source and target domains.

## More Experimental Results

More experimental results are shown in Figure 4. Among them, Figure 4(a) and 4(d) is a supplement of A2GNN$_{mmd}$, which in line with the conclusion in Ablation Study. The remaining images validates the effectiveness of A2GNN$_{adv}$. Table 8 demonstrates that the addition of each module in A2GNN$_{adv}$ contributes to the final results without any performance degradation. Figure 4(b) and Figure 4(c) demonstrate that utilizing a small number of $k$ ($k < 5$) can significantly improve its performance, which is in line with the previous conclusion. For the $D \to A$ task, as the value of $k$ increases, its performance improves correspondingly. In contrast, for the $D \to C$ task, optimal performance is observed within the range of 5 to 10. Figure 4(e) and Figure 4(f) demonstrate that A2GNN$_{adv}$ consistently performs well across the range of $[0.1, 0.8]$, suggesting its robustness in the optimization process. The optimal performance has a gradual decline as the value of $\alpha$ is greater than 0.8. The reason behind this is that, with an increase in the proportion of the adversarial loss function, the training of the model becomes gradually unstable, and the utilization of supervised signals weakens. The standard deviations of the results are shown in Table 9&10.

Table 9: Standard deviations of A2GNN on social network.

| Models | DE→EN | | EN→DE | |
|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| **A2GNN**$_{mmd}$ | 58.44 ± 0.32 | 59.54 ± 0.37 | 61.42 ± 0.84 | 63.90 ± 0.20 |
| **A2GNN**$_{adv}$ | 59.16 ± 0.21 | 59.94 ± 0.20 | 62.03 ± 0.20 | 65.11 ± 0.24 |

Table 10: Standard deviations of A2GNN on citation network.

| Models | D → A | | C → A | | A → D | | C → D | | A → C | | D → C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 | Ma-F1 | Mi-F1 |
| **A2GNN**$_{mmd}$ | 75.69 ± 0.13 | 74.12 ± 0.11 | 77.57 ± 0.19 | 76.15 ± 0.23 | 75.78 ± 0.45 | 77.43 ± 0.23 | 77.04 ± 0.65 | 78.13 ± 0.42 | 81.30 ± 0.14 | 82.64 ± 0.03 | 79.74 ± 0.11 | 81.5 ± 0.06 |
| **A2GNN**$_{adv}$ | 73.81 ± 0.03 | 72.17 ± 0.08 | 76.64 ± 0.20 | 75.22 ± 0.24 | 74.03 ± 0.70 | 75.42 ± 0.28 | 75.82 ± 0.76 | 77.32 ± 0.42 | 78.60 ± 0.06 | 80.27 ± 0.11 | 78.18 ± 0.44 | 80.08 ± 0.34 |

## Explanation of different operations in Fig 1

Staking vanilla GCN layer denoted as [PT]. When stacking $n$ transformation operations, P[T] implies propagating on the graph before applying $n$ transformations with activation functions, while [T]P implies applying transformations before propagating once on the graph. For the implementation details, we use the vanilla GCNConv, and iterate linear transformation and propagation to achieve P[T] and [T]P respectively. Activation function is ReLU. For optimization, we utilize the Adam optimizer with a learning rate of 0.005 and a weight decay of 0.001.

## Performance on source graph

When transferring knowledge to the target graph, it is important to note that the decrease in the effectiveness of the source domain should not be considered as a cost. Therefore, we present the performance on the source graph with a baseline method called GCN, which is trained solely using the source graph. The experiments are repeated five times, and we report the mean performance in terms of Mi-F1. As can be seen in Table 11, the performance on source graphs remains consistently high throughout the experiments.

Table 11: Performance on source graphs.

| Models | D → A | C → A | A → D | C → D | A → C | D → C | DE→EN | EN→DE |
|---|---|---|---|---|---|---|---|---|
| GCN | 85.25 | 85.84 | 84.08 | 85.04 | 84.24 | 85.84 | 67.29 | 67.38 |
| **A2GNN**$_{mmd}$ | -0.04 | +0.02 | -0.62 | +0.40 | -0.79 | -0.15 | -0.67 | -1.20 |
| **A2GNN**$_{adv}$ | -0.06 | +0.02 | +0.08 | +0.94 | +1.38 | -0.02 | -3.45 | +0.68 |

## Implementation details

**Implementation details of Tab 1&2**  The experimental settings of Tab 1&2 are consistent with the experimental section, which conducts node classification on citation dataset (D→A) as described in Section: Empirical Analysis. Activation function is ReLU. For optimization, we utilize the Adam optimizer with a learning rate of 0.005 and a weight decay of 0.001. We set the trade-off parameter to 1.

**Hardware & Software Environment**  The experiments are performed on one Linux servers (CPU: Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz, Operation system: Ubuntu 18.04.5 LTS). For GPU resources, one NVIDIA Tesla V100 card with 32G are utilized The python libraries we use to implement our experiments are Python 3.8 and Pytorch 1.11.0. The maximum time for training one epoch is no more than five minute.