Zero-shot Domain Adaptation of Heterogeneous Graphs via Knowledge Transfer Networks

Minji Yoon 1 John Palowitch 2 Dustin Zelle 2 Ziniu Hu 3 Ruslan Salakhutdinov 1 Bryan Perozzi 2

Abstract

How can we make predictions for nodes in a heterogeneous graph when an entire type of node (e.g. user) has no labels (perhaps due to privacy issues) at all? Although heterogeneous graph neural networks (HGNNs) have shown superior performance as powerful representation learning techniques, there is no direct way to learn using labels rooted at different node types. Domain adaptation (DA) targets this setting, however, existing DA cannot be applied directly to HGNNs. In heterogeneous graphs, the source and target domains have different modalities, thus HGNNs provide different feature extractors to them, while most of DA assumes source and target domains share a common feature extractor. In this work, we address the issue of zero-shot domain adaptation in HGNNs. We first theoretically induce a relationship between source and target domain features extracted from HGNNs, then propose a novel domain adaptation method, Knowledge Transfer Networks for HGNNs (HGNN-KTN). HGNN-KTN learns the relationship between source and target features, then maps the target distributions into the source domain. HGNN-KTN outperforms state-of-the-art baselines, showing up to 73.3% higher in MRR on 18 different domain adaptation tasks running on real-world benchmark graphs.

1. Introduction

Over the past decade, a significant line of research has been concerned with mining heterogeneous graphs (Sun & Han 2012) – graphs which connect a set of data with different modalities (e.g., videos, text, images) and express complex relationships between them. One active question in this area has been how to best apply graph neural networks (GNNs) (Chami et al., 2020) to learn representations of heterogeneous graph data (Schlichtkrull et al., 2018; Wang et al., 2019; Zhang et al., 2019a; Hu et al., 2020b). These models, called heterogeneous graph neural networks (HGNNs) extend how GNNs work on homogeneous graphs by providing different convolution modules for each node/edge types. By aggregating different node types of different modalities through convolution filters that are customized to each edge type, HGNNs can deal with multimodal datasets. These HGNNs have shown state-of-the-art performance in various graph mining tasks such as link prediction, recommendation, node classification, and clustering.

Despite their many successes, there are a number of critical challenges facing HGNNs. One of the most common problems in real-world applications of graph representation learning involves the scarcity of labels available for many classification tasks (Zhu et al., 2021; Halcrow et al., 2020). With their diverse nodes types, HGNN models are even more likely to face challenges due to label scarcity. For instance, there might be too much content in a social graph to serve as labels, while user nodes may have limitations due to privacy issues.

The field of domain adaptation (DA) targets this setting, seeking to transfer knowledge from a source domain with abundant labels to a target domain which lacks them (Long et al., 2017b; Ganin et al., 2016; Shen et al., 2018). Most DA methods focus on learning between different domains
of the same data modality (e.g., transfer from webcam images to DSLR images), with the assumption that the source and target domains share the same feature extractors. Unfortunately, such assumption is not satisfied for heterogeneous graphs, and thus existing DA methods do not directly apply to HGNNS. HGNNs need to deal with various node types, each of which may have its own modality and thus requires its own feature extractors for each domain. Additionally, in HGNNs, each feature extractor is composed of not only each domain's distinct encoder, but also may consist of shared modules (latent representations). In this case, the target domain feature extractor may not receive gradients as previous domain adaptation methodologies expect — indeed, some modules may not receive any gradients at all, while other modules might receive only small (or wrong) gradients.

In this work, we propose a novel domain adaption method for HGNNs to transfer knowledge between different node types on a heterogeneous graph. We first analyze how feature extractors are designed independently for each node type in HGNNs and then relate how distributions of the source and target domains could be represented in terms of each other. Next, we model this theoretical relationship between the two domains as a Knowledge Transfer Network (HGNN-KTN) which can be optimized to transform data according to this relationship. HGNN-KTN transforms target embeddings to fit into source domain distributions.

We perform an extensive evaluation of our method on real datasets where we compare against many popular domain adaptation baselines, such as DAN (Long et al., 2015), DANN (Ganin et al., 2016), CDAN (Long et al., 2017a), WDGRL (Shen et al., 2018) and many more. These results show that HGNN-KTN can achieve superior performance on a large variety of DA tasks. Finally, in order to understand which environments are ideal for transferring knowledge between different node types for heterogeneous graphs, we formulate a synthetic graph generator that allows us to study the sensitivity of these methods.

Our main contributions are:

• **Domain Adaptation for Heterogeneous Graphs**: To the best of our knowledge, HGNN-KTN is the first cross-type domain adaptation method designed for heterogeneous graphs.

• **Theoretical Foundations**: HGNN-KTN is a principled approach analytically induced from the architecture of HGNNs.

• **Experimental Results**: On real-world datasets, we show that HGNN-KTN outperforms state-of-the-art domain adaptation methods, being up to 73.3% higher in MRR on 18 different domain adaptation tasks.

• **Sensitivity Analysis**: We provide a heterogeneous graph generator model to analyze how the feature and edge distributions of heterogeneous graphs affect the performance of HGNN-KTN and other methods on the task.

2. Related Work

Zero-shot domain adaptation: zero-shot domain adaptation can be categorized into three groups — MMD-based methods, adversarial methods, and optimal-transport-based methods. MMD-based methods (Long et al., 2015; Sun et al., 2016; Long et al., 2017b) minimize the maximum mean discrepancy (MMD) (Gretton et al., 2012) between the mean embeddings of two distributions in reproducing kernel Hilbert space. Adversarial methods (Ganin et al., 2016; Long et al., 2017a) are motivated by theory in (Ben-David et al., 2007; 2010) suggesting that a good cross-domain representation contains no discriminative information about the origin of the input. They learn domain invariant features by a min-max game between the domain classifier and the feature extractor. Optimal transport-based methods (Shen et al., 2018) estimate the empirical Wasserstein distance (Redko et al., 2017) between two domains and minimizes the distance in an adversarial manner. All three categories rely on two networks — a feature extractor network and a task classifier network. Adversarial and OT-based methods use an additional network, the domain classifier. Assuming source and target domains have the same modality, previous methods use the same feature extractor for both domains such as convolutional neural networks for image domains.

Label propagation (LP): LP (Zhu, 2005) is a traditional graph mining algorithm which can solve the zero-shot domain adaptation problem on a heterogeneous graph. In LP, nodes propagate their labels to their neighbors according to normalized edge weights. LP relies on only a graph’s edges, and is therefore easily applied to a heterogeneous graph – labels are simply propagated across edges, regardless of type. We propose another simple baseline, embedding propagation (EP). Similar to LP, EP recursively averages embeddings from the source domain until they reach the target domain. EP exploits both feature information and graph structure, but only uses the source features.

3. Preliminaries

In this section, we briefly review the notation for heterogeneous graphs, and heterogeneous graph neural networks (HGNNS).

3.1. Heterogeneous graph

Heterogeneous graphs are an important abstraction for modeling the relational data of multi-modal systems. Formally, a heterogeneous graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$ where the node set $\mathcal{V} := \{1, \ldots, |V|\}$; the
edge set \( E \) consisting of ordered tuples \( e_{ij} \) := \((i, j)\) with \( i, j \in V \), where \( e_{ij} \in E \) iff an edge exists from \( i \) to \( j \); the set of node types \( T \) with associated map \( \tau : V \rightarrow T \); the set of relation types \( R \) with associated map \( \phi : E \rightarrow R \). This flexible formulation allows directed, multi-type edges. We additionally assume the existence of a node feature vector \( x_i \in X_{(i)} \) for each \( i \in V \), where \( X_{(i)} \) is a feature space specific to nodes of type \( t \). This allows \( G \) to represent nodes with different feature modalities such as images, text, locations, or booleans. Note that these modalities are not necessarily exclusive (e.g., two node types \( s, t \) might share the same feature space, \( X_s = X_t \)).

### 3.2. Heterogeneous GNNs

A graph neural network (GNN) can be regarded as a graph encoder which uses the input graph data as the basis for the neural network’s computation graph (Chami et al., 2020).

At a high-level, for any node \( j \) neural network’s computation graph:

\[
3.2. \text{Heterogeneous GNNs}
\]

same feature space, \( X \) with different feature modalities such as images, text, locations, or booleans. Note that these modalities are not necessarily exclusive (e.g. two node types \( s, t \) might share the same feature space, \( X_s = X_t \)).

**3.2. Heterogeneous GNNs**

A graph neural network (GNN) can be regarded as a graph encoder which uses the input graph data as the basis for the neural network’s computation graph (Chami et al., 2020).

At a high-level, for any node \( j \), the embedding of node \( j \) at the \( l \)-th GNN layer is obtained with the following generic formulation:

\[
\begin{align*}
\hat{h}_j^{(l)} &= \text{Transform}^{(l)} \left( \text{Aggregate}^{(l)}(E(j)) \right) \\
\text{Aggregate}(E(j)) &= \left\{ (i, k) \in E : i, k \in V, k = j \right\}
\end{align*}
\]

where \( E(j) = \{ (i, k) \in E : i, k \in V, k = j \} \) denotes all the edges which connect (directionally) to \( j \). HGGNs are a recently-introduced class of GNNs for modeling heterogeneous graphs. For HGNNs, the above operations typically involve type-specific parameters to exploit the inherent multiplicity of modalities in heterogeneous graphs.

We now define the commonly-used versions of Aggregate and Transform for HGNNs, which we use throughout this paper. First, we define a linear Message function

\[
\text{Message}^{(l)}(i, j) = M^{(l)}_{\phi(i, j)} \cdot \left( h^{(l-1)}_i \| h^{(l-1)}_j \right)
\]

where \( M^{(l)}_{\phi(i, j)} \) are the specific message passing parameters for each \( r \) \( R \) and each of \( L \) layers. Then defining \( E_r(j) \) as the set of edges of type \( r \) pointing to node \( j \), our HGNN Aggregate function mean-pools messages by edge type, and concatenates:

\[
\text{Aggregate}^{(l)}(E(j)) = \left\| \frac{1}{|E_r(j)|} \sum_{e \in E_r(j)} \text{Message}^{(l)}(e) \right\|_2
\]

Finally, Transform maps the message into a type-specific latent space:

\[
\text{Transform}^{(l)}(j) = \alpha(W_{\tau(j)} \cdot \text{Aggregate}^{(l)}(E(j)))
\]

The above formulation of HGNNs allows for full handling of the complexities of a real-world heterogeneous graph. By stacking HGNN blocks for \( L \) layers, each node aggregates a larger proportion of nodes — with different types and relations — in the full graph, which generates highly contextualized node representations. The final node representations can be fed into another model to perform downstream heterogeneous network tasks, such as node classification or link prediction.

### 4. Cross-Type Transformations in HGNNs

Borrowing paralnc from the domain adaptation setting, we define \( f_t : X_t \rightarrow \mathbb{R}^d \) to be the “feature extractor” of an HGNN, which represents the heterogeneous graph convolutions that map node features of type \( t \) into a shared latent space \( \mathbb{R}^d \). In this section, for any two nodes \( i, j \) with types \( \tau(i) = s \) and \( \tau(j) = t \), we analyze the relationship between \( f_s(x_i) \) and \( f_t(x_j) \), and derive a strict transformation between the feature output spaces. For simplicity, throughout this section we ignore the activation \( \alpha(\cdot) \) within the Transform function in Equation (4). In Section [5] we use this transformation to motivate HGNN-KTN, a novel approach which learns a mapping function between \( f_s(x_i) \) and \( f_t(x_j) \).

#### 4.1. Feature extractors for a toy heterogeneous graph

We first reason intuitively about the differences between \( f_s(x_i) \) and \( f_t(x_j) \) when \( s \neq t \), using a toy heterogeneous graph shown in Figure 2(a). In that graph, consider nodes \( v_1 \) and \( v_2 \), noticing that \( \tau(1) \neq \tau(2) \). Using Equations (2)-(4) from Section 3.2, for any \( l \in \{0, \ldots, L - 1\} \) we have

\[
\begin{align*}
\hat{h}_1^{(l)} &= W_s^{(l)} \left[ M_{s}^{(l)} \left( h_1^{(l-1)} \| h_2^{(l-1)} \right) \| M_{s}^{(l)} \left( h_2^{(l-1)} \| h_1^{(l-1)} \right) \right] \\
\hat{h}_2^{(l)} &= W_t^{(l)} \left[ M_{s}^{(l)} \left( h_1^{(l-1)} \| h_2^{(l-1)} \right) \| M_{t}^{(l)} \left( h_2^{(l-1)} \| h_2^{(l-1)} \right) \right]
\end{align*}
\]

where \( h_j^{(0)} = x_j \). From these equations, we see that the features of nodes \( v_1 \) and \( v_2 \), which are of different types, are extracted using disjoint sets of model parameters at each layer. In a 2-layer HGNN, this creates unique gradient backpropagation paths between the two node types, as illustrated in Figures 2(b)-(c). In other words, even though the same HGNN is applied to node types \( s \) and \( t \), the feature extractors \( f_s \) and \( f_t \) have different computational paths. Therefore they project node features into different latent spaces, and have different update equations during training. We study the consequences of this next.

#### 4.2. Empirical gap between \( f_s \) and \( f_t \)

Here we study the experimental consequences of the above observation via simulation. We first construct a synthetic graph extending the 2-type graph in Figure 2(a) to have multiple nodes per-type, and multiple classes. Next, we include well-separated classes in both the graph and feature space, where edges and Euclidean node features are well-clustered within-type and within-class (more details available in Appendix [A.4.1]).

On such a well-separated graph, without considering the observation in Section 4.1 there may seem to be no need for domain adaptation from \( f_t \) to \( f_s \). However, when we train the HGNN model solely on \( s \)-type nodes, as shown
Figure 2: Illustration of a toy heterogeneous graph and the gradient paths for feature extractors \( f_s \) and \( f_t \). We see that the same HGNN nonetheless produces different feature extractors for each feature domain \( \mathcal{X}_s \) and \( \mathcal{X}_t \). Colored arrows in figures (b) and (c) show the gradient paths for feature domains \( \mathcal{X}_s \) and \( \mathcal{X}_t \), respectively. Note the over-emphasis of the respective gradients in the (b) source and (c) target feature extractors, which can lead to poor generalization.

Figure 3: HGNNs trained on a source domain underfit a target domain and perform poorly on a “nice” heterogeneous graph. Our theoretically-induced version of HGNN-KTN adapts the model to the target domain successfully. In (a) we see performance on the simulated heterogeneous graph, for 4 kinds of feature extractors; (source: source extractor \( f_s \) on source domain, target-src-path: \( f_s \) on target domain, target-org-path: \( f_t \) on target domain, and theoretical-KTN: \( f_t \) on target domain using HGNN-KTN.) In (b-c), here L2 norms of gradients of parameters \( W^{(l)} \) and \( M^{(l)} \) in HGNN models.

in Figure 3(a) we find the test accuracy for \( s \)-type nodes to be high (90%) and the test accuracy for \( t \)-type nodes to be quite low (25%). Now if instead we make the \( t \)-type nodes use the source feature extractor \( f_s \), much more transfer learning is possible (∼65%, orange line). This shows the performance drop mainly comes from the different feature extractors present in the HGNN model, and so domain adaptation on it can not be solved by simply matching data distributions.

To analyze this phenomenon at the level of backpropagation, in Figures 3(b), 3(c) we show the magnitude of gradients passed to source and target node types. As we intuited in Section 4.2 and as illustrated in Figures 2(b), 2(c) we find that the final-layer Transform parameter \( W^{(l)} \) for \( t \)-type nodes have zero gradients (Figure 3(b)), and similarly for the final-layer Message parameters (Figure 3(c)). Additionally, those same parameters in the first-layer for \( t \)-type nodes have much smaller gradients than their \( s \)-type counterparts: \( W^{(1)} \) (blue line in Figure 3(b)), \( M^{(1)} \) (blue and orange lines in Figure 3(c)) appear below than other lines. This is because they contribute to \( f_s \) less than \( f_t \).

4.3. Relationship between feature extractors in HGNNs

The case study introduced in Section 4.2 shows that even when an HGNN is trained on a relatively simple, balanced, and class-separated heterogeneous graph, a model trained only on the source domain node type cannot transfer to the target domain node type. Here, to rigorously describe this phenomenon and the intuition behind it, we derive a strict transformation between \( f_s \) and \( f_t \), which will motivate the core domain adaptation component of HGNN-KTN. The following theorem assumes an HGNN as in Equations (2)-(4) without skip-connections, a simplification which we define and explain along with the proof in the Appendix:

**Theorem 1.** Given a heterogeneous graph \( \mathcal{G} = \{ \mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R} \} \). For any layer \( l > 0 \), define the set of \((l-1)\)-th layer HGNN parameters as

\[
Q^{(l-1)} = \{ M_r^{(l-1)} : r \in \mathcal{R} \} \cup \{ W_t^{(l-1)} : t \in \mathcal{T} \}. \tag{7}
\]

Let \( A \) be the total \( n \times n \) adjacency matrix. Then for any \( s, t \in \mathcal{T} \) there exist matrices \( A_s^{(l)} = a_{ts}(A) \) and \( Q_r^{(l)} = q_{ts}(Q_{ls}^{(l-1)}) \) such that

\[
H_s^{(l)} = A_s^{(l)} H_t^{(l)} Q_r^{(l)} \tag{8}
\]

where \( a_{ts}(\cdot) \) and \( q_{ts}(\cdot) \) are matrix functions that depend only on \( s, t \).
Algorithm 1 Training step for one minibatch

Require: heterogeneous graph \( G = (V, E, A, R) \), node feature matrices \( X \), source node type \( s \), target node type \( t \), adjacency matrix \( A_{ts} \), source node label matrix \( Y_s \).

Ensure: HGNN model \( f \), classifier \( g \), HGNN-KTN model \( t_{KTN} \).

1: \( H^{(0)}_t, H^{(0)}_s = f(H^{(0)} = X, G) \)
2: \( H^*_t = t_{KTN}(H^{(L)}_t) = A_{ts}H^{(L)}_tT_{ts} \)
3: \( L_{KTN} = \lVert H^{(L)}_s - H^*_t \rVert_2 \)
4: \( L = L_{CL}(g(H^{(L)}_s), Y_s) + \lambda L_{KTN} \)
5: Update \( f \), \( g \), \( t \) using \( \nabla L \)

The full proof of Theorem 1 can be found in Appendix A.1. Notice how in Equation 8, \( Q^*_t \) acts as a macro-
Message/Transform operator that maps \( H^{(L)}_t \) into the source domain, then \( A_{ts} \) aggregates the mapped embeddings into \( s \)-type nodes. To examine the implications of Theorem 1, we run the same experiment as described in Section 4.2 while this time mapping the target features \( H^{(L)}_t \) into the source domain by multiplying with \( Q^*_t \) in Equation 8 before passing over to a task classifier. We see via the red line in Figure 3(a) that, with this mapping, the accuracy in the target domain becomes much closer to the accuracy in the source domain (~70\%). Thus, we use this theoretical transformation as a foundation for our trainable HGNN domain adaptation module, introduced in the following section.

5. Method: HGNN-KTN

In the previous section, we show a HGNN model is transferred successfully from a source domain to a target domain using our theoretically-induced mapping function \( Q^*_t \). Here we discuss generalizing this idea to learn the mapping function directly.

Equation 8 in Theorem 1 looks similar to a single-layer graph convolutional network with a deterministic transformation matrix \( Q^*_t \) and a combination of adjacency matrices directing from target node type \( t \) to source node type \( s \) \( (A_{ts}) \). Our goal is to learn the mapping function \( Q^*_t \) that maps target embeddings into the source domain. By modelling Equation 8 as a trainable graph convolutional network, we can instead learn \( Q^*_t \). In order to do this, we introduce a knowledge transfer network \( t_{KTN} \) that replaces \( Q^*_t \) and \( A_{ts} \) in Equation 8 as follows:

\( t_{KTN}(H^{(L)}_t) = A_{ts}H^{(L)}_tT_{ts} \)

\( L_{KTN} = \lVert H^{(L)}_s - t_{KTN}(H^{(L)}_t) \rVert_2 \)

where \( T_{ts} \) is a trainable transformation matrix. By minimizing \( L_{KTN} \), \( T_{ts} \) is optimized to a mapping function of the target domain into the source domain.

Algorithm 2 Test step for a target domain

Require: pretrained HGNN model \( f \), classifier \( g \), HGNN-KTN model \( t_{KTN} \), target node label matrix \( Y_t \)

Ensure: target node label matrix \( Y_t \).

1: \( H^{(L)}_t = f(H^{(0)} = X, G) \)
2: \( H^*_t = H^{(L)}_tT_{ts} \)
3: return \( g(H^*_t) \)

5.1. Algorithm

We minimize a classification loss \( L_{CL} \) and a transfer loss \( L_{KTN} \) jointly with regard to a HGNN model \( f \), a classifier \( g \), and a knowledge transfer network \( t_{KTN} \) as follows:

\[
\min_{f, g, t_{KTN}} L_{CL}(g(f(X_s)), Y_s) + \lambda \| f(X_s) - t_{KTN}(f(X_t)) \|_2
\]

where \( \lambda \) is a hyperparameter regulating the effect of \( L_{KTN} \).

Algorithm 1 describes a training step with a minibatch. After computing the node embeddings \( H^{(L)}_s, H^{(L)}_t \) using a HGNN model \( f \), we map \( H^{(L)}_t \) to the source domain using \( t_{KTN} \) and compute \( L_{KTN} \). Finally, we update the models using gradients of \( L_{CL} \) and \( L_{KTN} \). Algorithm 2 describes the test phase on the target domain. After we get node embeddings \( H^{(L)}_t \) from the trained HGNN model \( f \), we map \( H^{(L)}_t \) into the source domain using the trained transformation matrix \( T_{ts} \). Finally we pass the transformed target embeddings \( H^*_t \) into the classifier \( g \) which was trained on the source domain.

Indirect Connections We note that in practice, the source and target node types can be indirectly connected in heterogeneous graphs via other node types. Appendix A.2 describes how we can easily extend HGNN-KTN to cover domain adaption scenarios in this case.

6. Experiments

6.1. Datasets

Open Academic Graph (OAG). A dataset introduced in Zhang et al. (2019b) composed of five types of nodes: papers, authors, institutions, venues, fields and their corresponding relationships. Paper and author nodes have text-based attributes, while institution, venue, and field nodes have text- and graph structure-based attributes. Paper, author, and venue nodes are labeled with research fields in two hierarchical levels, L1 and L2. To test the generalization of the proposed model, we construct three field-specific subgraphs from OAG: computer science, computer networks, and machine learning academic graphs.

PubMed. A network of genes, diseases, chemicals, and species (Yang et al., 2020), which has 11 types of edges. Gene and chemical nodes have graph structure-based attributes, while disease and species nodes have text-based attributes. Each gene or disease is labeled with a set of diseases they belong to or cause.

Synthetic heterogeneous graphs. We generate stochas-
We run 18 different zero-shot domain adaptation tasks across three OAG and PubMed graphs. Each heterogeneous graph has node classification tasks for both source and target node types. Only source node types have labels, while target node types have none during training. The performance is evaluated by NDCG and MRR — widely adopted ranking metrics (Hu et al., 2020b;a).

In Tables 1 and 2, our proposed method HGNN-KTN consistently outperforms all baselines on all tasks and graphs by up to 73.3% higher in MRR (P-A(L1) task in OAG-CS, Table 1). When we compare with the original accuracy possible using the model pretrained on the source domain without any domain adaptation (3rd column, Source), the results are even more impressive. Here we see our method HGNN-KTN provides relative gains of up to 340% higher MRR without using any labels from the target domain. These results show the clear effectiveness of HGNN-KTN on zero-shot domain adaptation tasks on a heterogeneous graph.

We note that in OAG graphs, the paper and author node types have different modalities (text and graph embeddings), and in the PubMed graph, disease and gene node types have different modalities (text and graph embeddings). In all cases, HGNN-KTN still transfers knowledge successfully while all baselines show poor performance even between domains of the same modalities (as they do not consider different feature extractors in HGNN models).

Finally, we mention that venue and author node types are not directly connected in the OAG graphs (Figure 6(a)), but HGNN-KTN consistently outperforms all baselines on all tasks and graphs by up to 340% higher MRR (P-A(L1) task in OAG-CS, Table 1). When we compare with the original accuracy possible using the model pretrained on the source domain without any domain adaptation (3rd column, Source), the results are even more impressive. Here we see our method HGNN-KTN provides relative gains of up to 340% higher MRR without using any labels from the target domain. These results show the clear effectiveness of HGNN-KTN on zero-shot domain adaptation tasks on a heterogeneous graph.

We note that in OAG graphs, the paper and author node types have different modalities (text and graph embeddings), and in the PubMed graph, disease and gene node types have different modalities (text and graph embeddings). In all cases, HGNN-KTN still transfers knowledge successfully while all baselines show poor performance even between domains of the same modalities (as they do not consider different feature extractors in HGNN models).

Finally, we mention that venue and author node types are not directly connected in the OAG graphs (Figure 6(a)), but HGNN-KTN successfully transfer knowledge by passing the intermediate nodes.

### Baseline Performance

Among baselines, MMD-based models (DAN and JAN) outperform adversarial based methods (DANN, CDAN, and CDAN-E) and optimal transport-based method (WDGRL), unlike results reported in (Long et al., 2017a; Shen et al., 2018). These re-

---

**Table 1: Open Academic Graph on Computer Science field.** The gain column shows the relative gain of our method over using no domain adaptation (source column).

| Task   | Metric | Source | DAN | JAN | DANN | CDAN | CDAN-E | WDGRL | LP | EP | KTN (gain) |
|--------|--------|--------|-----|-----|------|------|--------|-------|----|----|------------|
| P-A (L1) | NDCG  | 0.399  | 0.452 | 0.405 | 0.292 | 0.262 | 0.261 | 0.260 | 0.178 | 0.425 | 0.623 (56%) |
| A-P (L1) | NDCG  | 0.401  | 0.566 | 0.598 | 0.294 | 0.364 | 0.246 | 0.195 | 0.153 | 0.557 | 0.733 (83%) |
| A-V (L1) | NDCG  | 0.459  | 0.457 | 0.470 | 0.382 | 0.346 | 0.359 | 0.403 | 0.207 | 0.461 | 0.671 (46%) |
| V-A (L1) | NDCG  | 0.283  | 0.443 | 0.435 | 0.242 | 0.372 | 0.418 | 0.272 | 0.153 | 0.154 | 0.584 (107%) |
| P-A (L2) | NDCG  | 0.229  | 0.230 | o.o.m | 0.239 | o.o.m | o.o.m | 0.168 | o.o.m | 0.215 | 0.282 (23%) |
| A-P (L2) | NDCG  | 0.197  | 0.162 | o.o.m | 0.204 | 0.158 | 0.161 | 0.132 | o.o.m | 0.208 | 0.287 (46%) |
| A-V (L2) | NDCG  | 0.347  | 0.329 | 0.295 | 0.325 | 0.288 | 0.273 | 0.289 | o.o.m | 0.297 | 0.402 (116%) |
| V-A (L2) | NDCG  | 0.301  | 0.249 | 0.251 | 0.214 | 0.197 | 0.205 | 0.217 | o.o.m | 0.119 | 0.252 (7%) |

---

**Table 2: PubMed**

| Task | Metric | Source | DAN | JAN | DANN | CDAN | CDAN-E | WDGRL | LP | EP | KTN (gain) |
|------|--------|--------|-----|-----|------|------|--------|-------|----|----|------------|
| D-G  | NDCG  | 0.587  | 0.629 | 0.615 | 0.614 | 0.624 | 0.646 | 0.604 | 0.601 | 0.571 | 0.700 (19%) |
| G-D  | NDCG  | 0.596  | 0.599 | 0.577 | 0.599 | 0.581 | 0.606 | 0.578 | 0.576 | 0.580 | 0.662 (11%) |

---

**6.2. Baselines**

We compare HGNN-KTN with two MMD-based DA methods (DAN (Long et al., 2015), JAN (Long et al., 2017b)), three adversarial DA methods (DANN (Ganin et al., 2016), CDAN (Long et al., 2017b), CDAN-E (Long et al., 2017a)), one optimal transport-based method (WDGRL (Shen et al., 2018)), and two traditional graph mining methods (LP and EP (Zhu, 2005)). For DA methods, we use a HGNN model as their feature extractors. More information of each method is described in Appendix A.4.

---

**6.3. Zero-shot domain adaptation**

We compare HGNN-KTN with two MMD-based DA methods (DAN (Long et al., 2015), JAN (Long et al., 2017b)), three adversarial DA methods (DANN (Ganin et al., 2016), CDAN (Long et al., 2017b), CDAN-E (Long et al., 2017a)), one optimal transport-based method (WDGRL (Shen et al., 2018)), and two traditional graph mining methods (LP and EP (Zhu, 2005)). For DA methods, we use a HGNN model as their feature extractors. More information of each method is described in Appendix A.4.

---

**6.3. Zero-shot domain adaptation**

We compare HGNN-KTN with two MMD-based DA methods (DAN (Long et al., 2015), JAN (Long et al., 2017b)), three adversarial DA methods (DANN (Ganin et al., 2016), CDAN (Long et al., 2017b), CDAN-E (Long et al., 2017a)), one optimal transport-based method (WDGRL (Shen et al., 2018)), and two traditional graph mining methods (LP and EP (Zhu, 2005)). For DA methods, we use a HGNN model as their feature extractors. More information of each method is described in Appendix A.4.
versed results are a consequence of HGNN’s unique feature extractors for source and target domains. DANN and CDAN define their adversarial losses as a cross entropy loss $\langle \log f_s(x_s) \rangle - \langle \log f_t(x_t) \rangle$ where gradients of the subloss $\langle \log f_s(x_s) \rangle$ computed from the source feature extractor $f_s(x_s)$ are passed only back to $f_s(x_s)$, while gradients of the subloss $\langle \log f_t(x_t) \rangle$ computed from the target feature extractor $f_t(x_t)$ are passed only back to $f_t(x_t)$. Importantly, source and target feature extractors do not share any gradient information, resulting in divergence. This did not occur in their original test environments where source and target domains share a single feature extractor. Similarly, WDGL measures the first-order Wasserstein distance as an adversarial loss, which also brings the same effect as the cross-entropy loss we described above, leading to divergent gradients between source and target feature extractors. On the other hand, DAN and JAN define a loss in terms of higher-order MMD between source and target feature extractors. The on the other hand, DAN and JAN define a loss in terms of higher-order MMD between source and target features. Then the gradients of the loss passed to each feature extractor contain both source and target feature information, resulting in a more stable gradient estimation. This shows again the importance of considering different feature extractors in HGNNs. More analysis can be found in Appendix A.3.

### 6.4. Sensitivity analysis

Using our synthetic heterogeneous graph described in Section 6.1, we generate non-trivial 2-type heterogeneous graphs to examine how the feature and edge distributions of heterogeneous graphs affect the performance of HGNN-KTN and other baselines. We generate a range of test-case scenarios by manipulating (1) signal-to-noise ratio $\sigma_f$ of within-class feature distribution and (2) signal-to-noise ratio $\sigma_e$ of within-class edge probability.

![Graphs](image)

**Figure 4:** Effects of edge probabilities and feature distributions across classes and types in 2-node type heterogeneous graphs.

| Task   | Metric | Source | DAN | JAN | DANN | CDAN | CDAN-E | WDGLR | LP | EP | KTN (gain) |
|--------|--------|--------|-----|-----|------|------|--------|-------|----|----|------------|
| **P-A (L2)** | NDCG | 0.331 | 0.344 | 0.335 | 0.355 | 0.287 | 0.221 | 0.270 | 0.382 | (16%) |
|        | MRR   | 0.250 | 0.277 | 0.280 | 0.250 | 0.199 | 0.130 | 0.270 | 0.360 | (44%) |
| **A-P (L2)** | NDCG | 0.313 | 0.290 | 0.250 | 0.234 | 0.168 | 0.266 | 0.114 | 0.319 | 0.364 | (17%) |
|        | MRR   | 0.250 | 0.233 | 0.250 | 0.130 | 0.116 | 0.051 | 0.212 | 0.038 | 0.296 | 0.368 | (47%) |
| **A-V (L2)** | NDCG | 0.539 | 0.521 | 0.510 | 0.467 | 0.362 | 0.471 | 0.232 | 0.443 | 0.567 | (5%)  |
|        | MRR   | 0.584 | 0.528 | 0.510 | 0.293 | 0.294 | 0.365 | 0.000 | 0.406 | 0.628 | (8%)  |
| **V-A (L2)** | NDCG | 0.256 | 0.343 | 0.345 | 0.265 | 0.328 | 0.316 | 0.263 | 0.133 | 0.119 | 0.348 | (33%) |
|        | MRR   | 0.117 | 0.296 | 0.286 | 0.151 | 0.285 | 0.275 | 0.147 | 0.000 | 0.000 | 0.296 | (141%) |

**Table 3:** Open Academic Graph on Computer Network field

| Task   | Metric | Source | DAN | JAN | DANN | CDAN | CDAN-E | WDGLR | LP | EP | KTN (gain) |
|--------|--------|--------|-----|-----|------|------|--------|-------|----|----|------------|
| **P-A (L2)** | NDCG | 0.268 | 0.290 | 0.291 | 0.249 | 0.232 | 0.272 | 0.215 | 0.318 | (19%) |
|        | MRR   | 0.134 | 0.220 | 0.222 | 0.095 | 0.098 | 0.195 | 0.143 | 0.269 | (102%) |
| **A-P (L2)** | NDCG | 0.261 | 0.225 | 0.234 | 0.228 | 0.241 | 0.241 | 0.119 | 0.267 | 0.319 | (22%) |
|        | MRR   | 0.207 | 0.127 | 0.155 | 0.152 | 0.095 | 0.182 | 0.035 | 0.214 | 0.287 | (39%) |
| **A-V (L2)** | NDCG | 0.465 | 0.493 | 0.477 | 0.408 | 0.422 | 0.393 | 0.224 | 0.242 | 0.538 | (16%) |
|        | MRR   | 0.469 | 0.542 | 0.519 | 0.412 | 0.240 | 0.213 | 0.001 | 0.391 | 0.632 | (35%) |
| **V-A (L2)** | NDCG | 0.252 | 0.293 | 0.292 | 0.237 | 0.242 | 0.255 | 0.250 | 0.137 | 0.119 | 0.302 | (20%) |
|        | MRR   | 0.131 | 0.212 | 0.199 | 0.086 | 0.085 | 0.129 | 0.118 | 0.000 | 0.000 | 0.227 | (73%) |

**Table 4:** Open Academic Graph on Machine Learning field
of classes and same feature dimensions, (II) “hard” source and target node types have different number of classes and feature dimensions. At each unique value of $\sigma$ across the six $(\sigma, r)$ pairs, we generate 5 heterogeneous graphs, train HGNN-KTN and other DA baselines using source class labels, and test using target class labels.

The findings from our synthetic data study are shown in Figure 4. Figures 4(a) and 4(c) show results from changing $\sigma_e$ across the three relation types. We see that HGNN-KTN is affected only by $\sigma_e$ across the $s \leftrightarrow t$ relationship, which accords with our theory, since HGNN-KTN exploits the between-type computation (adjacency) matrix. Surprisingly, as seen in Figures 4(b) and 4(d) we do not find a similar dependence of HGNN-KTN on $\sigma_f$, which shows that HGNN-KTN is robust by learning purely from edge homophily in the absence of feature homophily. This robustness is a result of our theoretically-motivated formulation of KTN, allowing the full expressivity of HGNNs within the transfer-learning task.

Regarding the performance of other baselines, EP shows similar tendencies as HGNN-KTN— only affected by cross-type $\sigma_e$ — because EP also relies on cross-type propagation along edges. However, its accuracy is bounded above due to the fact that it does not model or propagate the (unlabelled) target features. DAN and DANN, which do not exploit cross-type edges, are not affected by cross-type $\sigma_e$. However, they show either low or unstable performance across different scenarios. DAN shows especially poor performance in the “hard” scenarios (Figure 4(c) and 4(d)), failing to deal with different feature spaces for source and target domains.

### 6.5. Different types of HGNNs

Using different parameters for each node and edge types in HGNNs result in different feature extractors for source and target node types. By sharing more parameters among node/edge types, could we see domain adaptation effect? Here, we design two variants of HGNNs. HGNN-v1 provides node-wise input layer that maps different modalities into the shared dimension then shares all the remaining parameters across nodes and layers. HGNN-v2 provides node-wise transformation matrices and edge-wise message matrices, but sharing them across layers. In Table 5 HGNN-v1 shows lower accuracy for both source and target node types. More parameters specialized to each node/edge types, HGNN models show higher accuracy on source domain, thus higher performance could be transferred to target domain. Regardless of HGNN model types, HGNN-KTN transfers knowledge between source and target node types consistently.

### 6.6. Effect of trade-off coefficient $\lambda$

We examine the effect of $\lambda$ on the domain adaptation performance. In Table 6 as $\lambda$ decreases, target accuracy decreases as expected. Source accuracy also sees small drops since $L_{KTN}$ functions as a regularizer; by removing the regularization effect, source accuracy decreases. When $\lambda$ becomes large, both source and target accuracy drop significantly. Source accuracy drops since the effect of $L_{KTN}$ becomes bigger than the classification loss $L_{CL}$. Even the effect of transfer learning become bigger by having bigger $\lambda$, since the source accuracy which will be transferred to the target domain is low, the target accuracy is also low. Thus we set $\lambda$ to 1 throughout the experiments.

### 7. Conclusion

In this work, we proposed the first zero-shot domain adaptation method for heterogeneous graphs. Our method, Knowledge Transfer Networks for Heterogeneous Graph Neural Networks (HGNN-KTN), transfers knowledge from a node type which has label information to node types without any. We illustrate the strength of our method on 18 domain adaptation tasks using large-scale dataset like the Open Academic Graph. In these experiments we see HGNN-KTN handily outperforms many challenging baselines, by up to 73.3% higher in MRR. Future work in the area includes filtering noisy edges between source and target domain and improving adjacency matrices used in...
HGNN-KTN. This direction would make HGNN-KTN more robust and less dependent on structure of given noisy heterogeneous graphs.

Acknowledgement

MY gratefully acknowledges support from Amazon Graduate Research Fellowship. GPUs are partially supported by AWS Cloud Credit for Research program.

References

Abbe, E. Community detection and stochastic block models: recent developments. The Journal of Machine Learning Research, 18(1):6446–6531, 2017.

Ben-David, S., Blitzer, J., Crammer, K., Pereira, F., et al. Analysis of representations for domain adaptation. Advances in neural information processing systems, 19:137, 2007.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. Machine learning, 79(1):151–175, 2010.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26, 2013.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675, 2020.

Dong, Y., Chawla, N. V., and Swami, A. metapath2vec: Scalable representation learning for heterogeneous networks. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 135–144, 2017.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks. The journal of machine learning research, 17(1):2096–2129, 2016.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. The Journal of Machine Learning Research, 13(1):723–773, 2012.

Halscrow, J., Mosoi, A., Ruth, S., and Perozzi, B. Grale: designing networks for graph learning. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2523–2532, 2020.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035, 2017.

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., and Sun, Y. Gpt-gnn: Generative pre-training of graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1857–1867, 2020a.

Hu, Z., Dong, Y., Wang, K., and Sun, Y. Heterogeneous graph transformer. In Proceedings of The Web Conference 2020, pp. 2704–2710, 2020b.

Long, M., Cao, Y., Wang, J., and Jordan, M. Learning transferable features with deep adaptation networks. In International conference on machine learning, pp. 97–105. PMLR, 2015.

Long, M., Cao, Z., Wang, J., and Jordan, M. I. Conditional adversarial domain adaptation. arXiv preprint arXiv:1705.10667, 2017a.

Long, M., Zhu, H., Wang, J., and Jordan, M. I. Deep transfer learning with joint adaptation networks. In International conference on machine learning, pp. 2208–2217. PMLR, 2017b.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pp. 3111–3119, 2013.

Redko, I., Habrard, A., and Sebban, M. Theoretical analysis of domain adaptation with optimal transport. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 737–753. Springer, 2017.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In European semantic web conference, pp. 593–607. Springer, 2018.

Shen, J., Qu, Y., Zhang, W., and Yu, Y. Wasserstein distance guided representation learning for domain adaptation. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-J., and Wang, K. An overview of microsoft academic service (mas) and applications. In Proceedings of the 24th international conference on world wide web, pp. 243–246, 2015.
Sun, B., Feng, J., and Saenko, K. Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Sun, Y. and Han, J. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159, 2012.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 990–998, 2008.

Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

Tsitsulin, A., Rozemberczki, B., Palowitch, J., and Perozzi, B. Synthetic graph generation to benchmark graph learning. *WWW’21. Workshop on Graph Learning Benchmarks*, 2021.

Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., and Yu, P. S. Heterogeneous graph attention network. In *The World Wide Web Conference*, pp. 2022–2032, 2019.

Wolf, T., Chaumond, J., Debut, L., Sanh, V., Delangue, C., Moi, A., Cistac, P., Funkowicz, M., Davison, J., Shleifer, S., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Yang, C., Xiao, Y., Zhang, Y., Sun, Y., and Han, J. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Zhang, C., Song, D., Huang, C., Swami, A., and Chawla, N. V. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 793–803, 2019a.

Zhang, F., Liu, X., Tang, J., Dong, Y., Yao, P., Zhang, J., Gu, X., Wang, Y., Shao, B., Li, R., et al. Oag: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2585–2595, 2019b.

Zhu, Q., Ponomareva, N., Han, J., and Perozzi, B. Shift-robust gns: Overcoming the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhu, X. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.
Algorithm 3 Training step (indirect version)

Require: heterogeneous graph \( G = (V, E, A, R) \), node feature matrices \( X \), adjacency matrices \( A_{xy}(x, y) \in \mathbb{R} \), source node type \( s \), target node type \( t \), source node label matrix \( Y_s \).

Ensure: HGNN \( f \), classifier \( g \), HGNN-KTN \( t_{\text{KTN}} \)

1: \( H^{(1)}_s, H^{(1)}_t = f(H^{(0)} = X, G), H^{(1)}_t = 0 \)
2: for each meta-path \( p = t \to s \) do
3: \( x = t \), \( Z = H^{(1)}_t \)
4: for each node type \( y \in p \) do
5: \( Z = A_{xy} Z T_{xy} \)
6: \( x = y \)
7: end for
8: \( H^{(1)}_i = H^{(1)}_i + Z \)
9: end for
10: \( L_{\text{KTN}} = \| H^{(1)}_i - H^{(1)}_s \|_2 \)
11: \( L = L_{\text{CL}}(g(H^{(1)}_s), Y_s) + \lambda L_{\text{KTN}} \)
12: Update \( f, g, t_{\text{KTN}} \) using \( \nabla L \)

A. Appendix

A.1. Proof of Theorem 1

The proof of Theorem 1 is below. As stated in the assumptions of the theorem, we adopt a simplified version of our message-passing function that ignores the skip-connection:

\[
\text{Message}^{(l)}(i, j) = M^{(l)}_{st}(i, j) h^{(l)}_i.
\]

This lets the Theorem match the experimental results shown in Figure 3 as the HGNN trained in that experiment does not use skip-connections and hence represents an “idealized” HGNN without skip-connections, and with a theoretically-exact KTN component. In the real experiments, we use (1) skip-connections, exploiting their usual benefits [Hamilton et al. 2017], and (2) the trainable version of KTN.

Proof. Without loss of generality, we prove the result for the case where \( \mathcal{R} = \{(s, t) : s, t \in \mathcal{T}\} \), meaning the type of an edge is identified with the (ordered) types of the neighbor nodes. In other words, there is only one edge modality possible, such as a social network with multiple node types (e.g. “users”, “groups”) but only one edge modality (“friendship”). In the case of multiple edge modalities (e.g. “friendship” and “message”), the result is extended trivially (though with more algebraically-dense forms of \( a_{st} \) and \( q_{ts} \)).

Throughout this proof, we use the following notation for the set of all \( j \)-adjacent edges of relation type \( r \):

\[
\mathcal{E}_r(j) := \{(i, j) : i \in \mathcal{V}, (i, j) = r\}.
\]

We write \( A_{rj} \) to denote the sub-matrix of the total \( n \times n \) adjacency matrix \( A \) corresponding to node types \( x_1, x_2 \in \mathcal{T} \), and \( A_{rj} \) to denote the same matrix divided by its sum. \( H^{(l)}_i \) is the (row-wise) \( n_x \times d_l \) embedding matrix of \( x \)-type nodes at layer \( l \).

To begin, we first compute the \( l \)-th output \( g^{(l)}_j \) of the

Algorithm 4 Test step for a target domain (indirect version)

Require: pretrained HGNN \( f \), classifier \( g \), HGNN-KTN \( t_{\text{KTN}} \)

Ensure: target node label matrix \( Y_t \)

1: \( H^{(1)}_t = f(H^{(0)} = X, G), H^{(1)}_t = 0 \)
2: for each meta-path \( p = t \to s \) do
3: \( x = t, Z = H^{(1)}_t \)
4: for each node type \( y \in p \) do
5: \( X = Z T_{xy} \)
6: \( x = y \)
7: end for
8: \( H^{(1)}_i = H^{(1)}_i + Z \)
9: end for
10: return \( g(H^{(1)}_i) \)

Aggregate step defined for HGNNs in Equation [3] for any node \( j \in \mathcal{V} \) such that \( \tau(j) = s \). The output of Aggregate is in fact a concatenation of edge-type-specific aggregations (see Equation [3]). Note that at most \( T = |\mathcal{T}| \) elements of this concatenation are non-zero, since the node \( j \) only participates in \( T \) out of \( T^2 \) relation types in \( R \). Thus we can write \( g^{(1)}_j \) as

\[
g^{(1)}_j = \frac{1}{|\mathcal{E}_r(j)|} \sum_{e \in \mathcal{E}_r(j)} \text{Message}^{(1)}(e) = \frac{1}{|\mathcal{E}_x(j)|} \sum_{e \in \mathcal{E}_x(j)} \text{Message}^{(1)}(e) \]

\[
= \frac{1}{|\mathcal{E}_x(j)|} \sum_{e \in \mathcal{E}_x(j)} M^{(1)} h^{(1)}_i
\]

\[
= \frac{1}{|\mathcal{E}_x(j)|} M^{(1)} h^{(1)}_i
\]

\[
= \sum_{i \in \mathcal{E}_x(j)} M^{(1)} h^{(1)}_i
\]

\[
= M^{(1)} (H^{(l-1)}_x)' \tilde{A}_x(j),
\]

where \( \tilde{A}_x(j) \) denotes the \( j \)-th column of \( \tilde{A}_x \). Notice that

\[
h^{(1)}_j = \text{Transform}^{(1)}(j) = W^{(1)}_s g^{(1)}_j,
\]

and (again) at most \( T \) elements of the concatenation \( g^{(1)}_j \) are non-zero. Therefore let \( W^{(1)}_s \) be the columns of \( W^{(1)}_s \) that select the concatenated element of \( g^{(1)}_j \) corresponding to node type \( x \). Then we can write

\[
h^{(1)}_j = \sum_{x \in \mathcal{T}} W^{(1)}_s M^{(1)}_x (H^{(l-1)}_x)' \tilde{A}_x(j).\]

Defining the operator \( Q^{(l)}_x := (W^{(1)}_s M^{(1)}_x)' \), this implies that

\[
H^{(l)}_x = \sum_{x \in \mathcal{T}} \tilde{A}_x H^{(l-1)}_x Q^{(l)}_x
\]

\[
= [\tilde{A}_{x_1}, \ldots, \tilde{A}_{x_T}] \begin{bmatrix}
H^{(l-1)}_x & 0 & 0 \\
0 & \ldots & 0 \\
0 & H^{(l-1)}_x & \ldots
\end{bmatrix} \begin{bmatrix}
Q^{(l-1)}_{x_1} \\
Q^{(l-1)}_{x_2} \\
\vdots
\end{bmatrix}
\]

\[
= \tilde{A}_x H^{(l-1)} Q^{(l-1)}_x
\]

Similarly we have \( H^{(l)}_t = \tilde{A}_t H^{(l-1)} Q^{(l-1)}_x \). Since \( H^{(l)}_x \)
and $H_t^{(i)}$ share the term $H_t^{(i-1)}$, we can write

$$H_t^{(i)} = A_s A_t^{-1} H_t^{(i-1)} (Q_t^{(i-1)})^{-1} Q_s^{(i-1)}, \quad (15)$$

where $X^{-1}$ denotes the pseudo-inverse. This proves the result.

### A.2. Indirectly Connected Source and Target Node Types

When source and target node types are indirectly connected by another node type $x$, we can simply extend $f_{KTN}(H_t^{(L)})$ to $(A_{xs}(A_{tx} H_t^{(L)} T_{xs}) T_{xs})$ where $T_{tx} T_{xs}$ becomes a mapping function from target to source domains. Algorithm 3 and 4 show how HGNN-KTN is extended. For every step ($x \rightarrow y$) in a meta-path ($t \rightarrow \cdots \rightarrow s$) connecting from target node type $t$ to source node type $s$, we define a transformation matrix $T_{xy}$, run a convolution operation with an adjacency matrix $A_{xy}$, and map the transformed embedding to the source domain. We run the same process for all meta-paths connecting from target node type $t$ to source node type $s$, and sum up them to match with the source embeddings. In the test phase, we run the same process to get the transformed target embeddings, but this time, without adjacency matrices. We run Algorithm 3 and 4 for domain adaptation tasks between author and venue nodes which are indirectly connected by paper nodes in OAG graphs (Figure 6(a)). As shown in Tables 3 and 4 we successfully transfer HGNN models between author and venue nodes (A-V and V-A) for both L1 and L2 tasks.

Which meta-path between source and target node types should we choose? Will lengths of meta-paths affect the performance? We examine the performance of HGNN-KTN varying the length of meta-paths. In Table 7 accuracy decreases with longer meta-paths. When we add additional meta-paths than the minimum path, it also brings noise in every edge types. Note that author and venue nodes are indirectly connected by paper nodes; thus the minimum length of meta-paths in the A-V (L1) task is 2. The accuracy in the A-V (L1) task with a meta-path of length 1 is low because HGNN-KTN fails to transfer anything with a meta-path shorter than the minimum. Using the minimum length of meta-paths is enough for HGNN-KTN.

### A.3. Analysis for Baselines in Section 6.3

JAN, CDAN, and CDAN-E often show out of memory issues in Tables 1, 3, and 4. These baselines consider the classifier prediction whose dimension is equal to the number of classes in a given task. That is why JAN, CDAN, and CDAN-E fail at the L2 field prediction tasks in OAG graphs where the number of classes is 17,729.

LP performs worst among the baselines, showing the limitation of relying only on graph structures. LP maintains a label vector with the length equal to the number of classes for each node, thus shows out-of-memory issues on tasks with large number of classes on large-size graphs (L2 tasks with 17,729 labels on the OAG-CS graph). EP performs moderately well similar to other DA methods, but lower than HGNN-KTN up to 60% absolute points of MRR, showing the limitation of not using target node attributes.

### A.4. Synthetic Heterogeneous Graph Generator

Our synthetic heterogeneous graph generator is based on attributed Stochastic Block Models (SBM) (Tsitsulin et al. 2020, 2021), using clusters (blocks) as the node classes. In the attributed SBM, graphs exhibit within-type cluster homophily at the edge-level (nodes most-frequently connect to other nodes in their cluster), and at the feature-level (nodes are closest in feature space to other nodes in their cluster). To produce heterogeneous graphs, we additionally introduce between-type cluster homophily, which allows us to model real-world heterogeneous graphs in which knowledge can be shared across node types.

The first step in generating a heterogeneous SBM is to decide how many clusters will partition each node type. Assume within-type cluster counts $k_1, \ldots, k_T$. We allow for cross-type homophily with a $K_T := \min \{k_i\}$-partition of clusters such that each cluster group has at least one cluster from each node type.

Secondly, edge-level homophily is controlled by signal-to-noise ratios $\sigma_e = p/q$ where nodes within-cluster are connected with probability $p$ and nodes between-cluster are connected with probability $q$. Additionally, nodes within the same cluster group across-types (see previous paragraph) can generate between-edges with some $\sigma_e > 1.0$. In Section 6.4 we describe the manipulation of multiple $\sigma_e$ parameters within-and-across types.

Finally, node attributes are generated by a multivariate Normal mixture model, using the cluster partition as the mixture groups. Thus feature-level homophily is controlled by increasing the variance of the cluster centers $\sigma_f$, while keeping the within-cluster variance fixed. Note that features of different types are allowed to have different dimensions, as we generate different mixture-model cluster centers for each cluster within each type. Cross-type fea-

### Table 7: Meta-path length in HGNN-KTN: increasing the meta-path longer than the minimum does not bring significant improvement to HGNN-KTN. Note that the minimum length of meta-paths in the A-V (L1) task is 2.

| Meta-path length | P-A (L1) | A-V (L1) |
|------------------|---------|---------|
|                  | NDCG   | MRR     | NDCG   | MRR     |
| 1                | 0.623  | 0.621   | 0.208  | 0.010   |
| 2                | 0.627  | 0.628   | 0.673  | 0.696   |
| 3                | 0.608  | 0.611   | 0.627  | 0.648   |
| 4                | 0.61   | 0.623   | 0.653  | 0.671   |
ture homophily is not necessary, since HGNN-KTN learns a transformation function between the type feature spaces.

A.4.1. TOY HETEROGENEOUS GRAPH IN SECTION 4.2

Using the synthetic graph procedure described above, we used the following hyperparameters to simulate the toy heterogeneous graph shown in Figure 3. We generate the graph with two node types and four edge types as described in Figure 2(a), then we divide each node type into 4 classes of 400 nodes. To generate an easy-to-transfer scenario, signal-to-noise ratio $\sigma_f$ between means of feature distributions are all set to 10. The ratio $\sigma_e$ of the number of intra-class edges to the number of inter-class edges is set to 10 among the same node types and across different node types. The dimension of features is set to 24 for both node types.

A.4.2. SENSITIVITY TEST IN SECTION 6.4

Figure 5 shows the structures of graphs we used in Section 6.4. The dimension of features are set to 24 for both node types for the “easy” scenario and, and 32, 48 for types $s$ and $t$ (respectively) for the “hard” scenario. Additionally, for the “hard” scenario, we divide the $t$ nodes into 8 clusters instead of 4. The other hyperparameters $\sigma_e$ and $\sigma_f$ are described in Section 6.4.

A.5. Real-world Dataset

Open Academic Graph (OAG) (Sinha et al., 2015; Tang et al., 2008; Zhang et al., 2019b) is the largest publicly available heterogeneous graph. It is composed of five types of nodes: papers, authors, institutions, venues, and fields and their corresponding relationships. Papers and authors have text-based attributes, while institutions, venues, and fields have text- and graph structure-based attributes. To test the generalization of the proposed model, we construct three field-specific subgraphs from OAG: the Computer Science (OAG-CS), Computer Networks (OAG-CN), and Machine Learning (OAG-ML) academic graphs.

Papers, authors, and venues are labeled with research fields in two hierarchical levels, L1 and L2. OAG-CS has both L1 and L2 labels, while OAG-CN and OAG-ML have only L2 labels (their L1 labels are all “computer science”). Domain adaptation is performed on the L1 and L2 field prediction tasks between papers, authors, and venues for each of the aforementioned subgraphs. Note that paper-author (P-A) and paper-venue (P-V) are directly connected, while author-venue (A-V) are indirectly connected via papers.

The number of classes in the L1 task is 275, while the number of classes in the L2 task is 17,729. The graph statistics are listed in Table 8, in which P–A, P–F, P–V, A–I, P–P, and F–F denote the edges between paper and author, paper and field, paper and venue, author and institute, the citation links between two papers, the hierarchical links between two fields. The graph structure is described in Figure 6(a).

For paper nodes, features are generated from each paper’s title using a pre-trained XLNet (Wolf et al., 2020). For author nodes, features are averaged over features of papers they published. Feature dimension of paper and author nodes is 769. For venue, institution, and field node types, features of dimension 400 are generated from their heterogeneous graph structures using metapath2vec (Dong et al., 2017).

PubMed (Yang et al., 2020) is a novel biomedical network constructed through text mining and manual processing on biomedical literature. PubMed is composed of genes, diseases, chemicals, and species. Each gene or disease is labeled with a set of diseases (e.g., cardiovascular disease) they belong to or cause. Domain adaptation is performed on a disease prediction task between genes and disease node types.

The number of classes in the disease prediction task is 8. The graph statistics are listed in Table 9, in which G, D, C, and S denote genes, diseases, chemicals, and species node types. The graph structure is described in Figure 6(b).

For gene and chemical nodes, features of dimension 200 are generated from related PubMed papers using word2vec (Mikolov et al., 2013). For diseases and species nodes, features of dimension 50 are generated based on their graph structures using TransE (Bordes et al., 2013).

A.6. Baselines

Zero-shot domain adaptation can be categorized into three groups — MMD-based methods, adversarial methods, and optimal-transport-based methods. MMD-based meth-
### Table 8: Statistics of Open Academic Graph

| Domain          | #papers | #authors | #fields | #venues | #institutes |
|-----------------|---------|----------|---------|---------|-------------|
| Computer Science| 544,244 | 510,189  | 45,717  | 6,934   | 9,097       |
| Computer Network| 75,015  | 82,724   | 12,014  | 2,115   | 4,193       |
| Machine Learning| 90,012  | 109,423  | 19,028  | 3,226   | 5,455       |

| Domain          | #P-A    | #P-F    | #P-V    | #A-I    | #P-P    | #F-F    |
|-----------------|---------|---------|---------|---------|---------|---------|
| Computer Science| 1,091,560 | 3,709,711 | 544,245  | 612,873  | 11,592,709 | 525,053  |
| Computer Network| 155,147 | 562,144 | 75,016  | 111,180  | 1,154,347  | 110,869  |
| Machine Learning| 166,119 | 585,339 | 90,013  | 156,440  | 1,209,443  | 163,837  |

### Table 9: Statistics of PubMed Graph

| #gene | #disease | #chemicals | #species |
|-------|----------|------------|----------|
| 13,561| 20,163   | 26,522     | 2,863    |
| #G-G  | #G-D     | #D-D       | #C-G     | #C-D     |
| 32,211| 25,963   | 68,219     | 31,278   | 51,324   |
| #C-C  | #C-S     | #S-G       | #S-D     | #S-S     |
| 124,375| 6,298    | 3,156      | 5,246    | 1,597    |

### A.7. Experimental Settings

All experiments were conducted on the same p2.xlarge Amazon EC2 instance. Here, we describe the structure of HGNNs used in each heterogeneous graph.

**Open Academic Graph:** We use a 4-layered HGNN with transformation and message parameters of dimension 128 for HGNN-KTN and other baselines. Learning rate is set to $10^{-4}$.

**PubMed:** We use a single-layered HGNN with transformation and message parameters of dimension 10 for HGNN-KTN and other baselines. Learning rate is set to $5 \times 10^{-5}$.

**Synthetic Heterogeneous Graphs:** We use a 2-layered HGNN with transformation and message parameters of dimension 128 for HGNN-KTN and other baselines. Learning rate is set to $10^{-4}$.

We implement LP, EP and HGNN-KTN using Pytorch. For the domain adaptation baselines (DAN, JAN, DANN, CDAN, CDAN-E, and WDGRL), we use a public domain adaptation library ADA[^1][^2]

[^1]: https://github.com/criteo-research/pytorch-ada