Zero-shot Transfer Learning within a Heterogeneous Graph via Knowledge Transfer Networks

Minji Yoon*
Carnegie Mellon University

John Palowitch
Google Research

Dustin Zelle
Google Research

Ziniu Hu*
University of California Los Angeles

Ruslan Salakhutdinov
Carnegie Mellon University

Bryan Perozzi
Google Research

Abstract

Data continuously emitted from industrial ecosystems such as social or e-commerce platforms are commonly represented as heterogeneous graphs (HG) composed of multiple node/edge types. State-of-the-art graph learning methods for HGs known as heterogeneous graph neural networks (HGNNs) are applied to learn deep context-informed node representations. However, many HG datasets from industrial applications suffer from label imbalance between node types. As there is no direct way to learn using labels rooted at different node types, HGNNs have been applied on only a few node types with abundant labels. We propose a zero-shot transfer learning module for HGNNs called a Knowledge Transfer Network (KTN) that transfers knowledge from *label-abundant* node types to *zero-labeled* node types through rich relational information given in the HG. KTN is derived from the theoretical relationship, which we introduce in this work, between distinct *feature extractors* for each node types given in a HGNN model. KTN improves performance of 6 different types of HGNN models by up to 960% for inference on zero-labeled node types and outperforms state-of-the-art transfer learning baselines by up to 73% across 18 different transfer learning tasks on HGs.

1 Introduction

Large technology companies commonly maintain large relational datasets, derived from their internal logs, that can be represented as or joined into a massive heterogeneous graph (HG) composed of nodes and edges with multiple types [30]. For instance, in e-commerce networks, there are product, user, and review nodes, all interconnected by many edge types that represent forms of interactions such as spending (user-product), reviewing (user-review), and reviews-of (product-review). To learn powerful features representing the complex multimodal structure of HGs, various heterogeneous graph neural networks (HGNN) have been proposed [15, 26, 35, 43].

A common issue in these industrial applications of HGNNs is the label imbalance among different node types. For instance, publicly available *content* nodes – such as those representing video, text, and image content – are abundantly labelled, whereas labels for other types (such as *user* or *account* nodes) may be much more expensive to collect (or even not available, e.g. due to privacy restrictions). This means that in most standard training settings, HGNN models can only learn to make good inferences for a few label-abundant node types, and can usually not make any inferences for the remaining node types, given the absence of any labels for them.

If there is a pair of *label-abundant* and *zero-labeled* node types which share an inference task, could we transfer knowledge between them? One body of work has focused on transferring knowledge

*Work done while interning at Google

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

between nodes of the *same* type from two *different* HGs (i.e., graph-to-graph transfer learning) [16, 40]. However, these approaches are not applicable in many real-world scenarios for three reasons. First, any external large-scale HG that could be used in a graph-to-graph transfer learning setting would almost surely be proprietary. Second, even if practitioners could obtain access to an external industrial HG, it is unlikely the distribution of that (source) graph would match their target graph well enough to apply transfer learning. Finally, node types suffering label scarcity are likely to suffer the same issue on other HGs (e.g. user nodes).

In this paper, we introduce a zero-shot transfer learning approach for a *single* HG (assumed to be fully-owned by the practitioners), transferring knowledge from labelled to unlabelled node types. This setting is distinct from any graph-to-graph transfer learning scenarios, since the source and target domains exist in the same HG dataset, and are assumed to have different node types. Our model utilizes the shared context between source and target node types; for instance, in the e-commerce network, the latent (unknown) labels of user nodes can be strongly correlated with spending/reviewing patterns that are encoded in the cross-edges between user nodes and product/review nodes. We propose a novel zero-shot transfer learning problem for this HG learning setting as follows:

Informal Problem Definition 1. Zero-shot cross-type transfer learning running on a HG: *Given a heterogeneous graph $\mathcal{G}$ with node types $\{\mathbf{s}, \mathbf{t}, \cdots\}$ with abundant labels for source type $\mathbf{s}$ but no labels for target type $\mathbf{t}$, can we train HGNNs to infer the labels of target-type nodes?*

A naïve solution to this problem would be to re-use an HGNN pre-trained on the source nodes for target node inference, given that both domains exist in the same HG. However, as we show in our paper, HGNNs have distinct parameter sets for each node type [15], edge type [26], and meta-path type [8, 35]. These facts cause HGNNs to learn entirely different *feature extractors* for nodes and edges of different types – in other words, the final embeddings for source and target nodes are computed by different sets of parameters in HGNNs. Thus, a classifier pre-trained on source nodes will fail to perform well on inference tasks for target nodes. 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 [9, 19, 20, 27]. However, distinct feature extractors across node types in HGNNs break a standard assumption of DA setting, namely that source and target domains share the same feature extractors (e.g., CNNs for both source and target image domains). As we demonstrate in this paper, in our problem setting, DA approaches fail to achieve the outstanding performance they are known for in computer vision and NLP.

In our work, we first dissect the gradient path of HGNN models to see how feature extractors are designed independently for each node type, and some empirical consequences. Then we theoretically analyze how feature extractors across node types relate to each other and how their output distributions could be represented in terms of each other. We model this theoretical relationship between two feature extractors as a Knowledge Transfer Network (KTN) which can be optimized to transform target embeddings to fit the source domain distribution. We perform an extensive evaluation of our method on 18 different transfer learning tasks on HGs where we compare against state-of-the-art domain adaptation baselines. Additionally, in order to understand which environments are ideal for transferring knowledge between different node types for HGs, we formulate a synthetic heterogeneous graph generator that allows us to study the sensitivity of these methods.

Our main contributions are:

- **Novel and practical problem definition:** To the best of our knowledge, KTN is the first zero-shot cross-type transfer learning method running on a heterogeneous graph — transfer knowledge across different node types within a heterogeneous graph.
- **Generality:** KTN is a principled approach analytically induced from the architecture of HGNNs, thus applicable to any HGNN models, showing up to 960% performance improvement for zero-labeled node inference across 6 different HGNN models.
- **Effectiveness:** We show that KTN outperforms state-of-the-art domain adaptation methods, being up to 73.3% higher in MRR on 18 different transfer learning tasks on HGs.
- **Sensitivity Analysis:** We provide a HG generator model to analyze how the node attribute and edge distributions of HGs affect the performance of KTN and other methods on the task.

2 Related Work

Various transfer learning problems have been defined on the graph domain. [21, 22, 38, 42] construct synthetic graphs from unstructured data and transfer knowledge over the graphs using GNNs. On

the other hand, [13, 14, 24, 37] focus on extracting knowledge from the existing graph structures. They pretrain a GNN model on a source graph and re-use the model on a target graph. While these methods focus on homogeneous graphs, [16, 40] transfer HGNNs across different HGs. However, none of them can be directly applied to our cross-type transfer learning problem running on a single HG. Here we cover two classes of learning approaches that are related to our problem. As HGNNs are the models to which our method can be applied, we cover them extensively in Section 3.

Zero-shot domain adaptation (DA) transfers knowledge from a source domain with abundant labels to a target domain which lacks them. Zero-shot DA can be categorized into three groups — MMD-based methods, adversarial methods, and optimal-transport-based methods. MMD-based methods [18, 20, 29] minimize the maximum mean discrepancy (MMD) [11] between the mean embeddings of two distributions in reproducing kernel Hilbert space. Adversarial methods [9, 19] are motivated by theory in [2, 3] 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 [27] estimate the empirical Wasserstein distance [25] 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 domain classifier network. Due to the assumption that source and target domains have the same modality [2], the standard DA setting assumes identical feature extractors across domains. More descriptions can be found in Appendix A.9.

Label propagation (LP) approaches (e.g., [45]) use message-passing to pass each node's label to its 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. In this paper we also evaluate a similarly-simple baseline, embedding propagation (EP). Similar to LP, EP recursively propagates source embeddings (computed using source labels) until they reach the target domain. EP exploits both node attribute information and the node adjacencies, but only uses the source node embeddings.

3 Preliminaries

In this section we review heterogeneous graphs and heterogeneous graph neural networks (HGNNs).

3.1 Heterogeneous graph

Heterogeneous graphs (HGs) 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 $\mathcal{V}$; the edge set $\mathcal{E}$ consisting of ordered tuples $e_{ij} := (i, j)$ with $i, j \in \mathcal{V}$, where $e_{ij} \in \mathcal{E}$ iff an edge exists from i to j; the set of node types $\mathcal{T}$ with associated map $\tau : \mathcal{V} \mapsto \mathcal{T}$; the set of relation types $\mathcal{R}$ with associated map $\phi : \mathcal{E} \mapsto \mathcal{R}$. This flexible formulation allows directed, multi-type edges. We additionally assume the existence of a node attribute vector $x_i \in \mathcal{X}_{\tau(i)}$ for each $i \in \mathcal{V}$, where $\mathcal{X}_t$ is an attribute matrix specific to nodes of type t .

3.2 Heterogeneous Graph Neural Networks (HGNN)

Various HGNN models have been proposed [15, 26, 35, 41, 43]. Fully-specified HGNN models have specialized parameters for each node type [15], edge type [26], and meta-path type [8] to most effectively utilize the complex relationships encoded in the HG data structure. In this paper, we use a flavor of HGNN known as a Heterogeneous Message-Passing Neural Network (HMPNN) as our base model on which to demonstrate KTN (though KTN can be implemented in almost any HGNN, as we show in experiments in Section 6). The HMPNN merely extends the standard MPNN [10] by specializing all transformation and message matrices in each node/edge type. With its generality, HMPNN is itself a base model for RGCN [26] and HGT [15], and is also widely used as a default HGNN model in popular GNN libraries (e.g., pyG [7], TF-GNN [6], DGL [34]).

[2] In our problem, source and target node types could have either (1) different distributions on the same attribute space or (2) entirely different attribute spaces

In a HMPNN, for any node j, the embedding of node j at the *l-th* layer is obtained with the following generic formulation:

$$h_j^{(l)} = \textbf{Transform}^{(l)}\left(\textbf{Aggregate}^{(l)}(\mathcal{E}(j))\right) \tag{1}$$

where $\mathcal{E}(j) = \{(i,j) \in \mathcal{E} : i,j \in \mathcal{V}\}$ denotes all the edges which connect (directionally) to j. The above operations typically involve type-specific parameters to exploit the inherent multiplicity of modalities in heterogeneous graphs. First, we define a linear **Message** function:

$$\textbf{Message}^{(l)}(i,j) = M^{(l)}_{\phi((i,j))} \cdot \left(h_i^{(l-1)} \parallel h_j^{(l-1)}\right) \tag{2}$$

where $M_r^{(l)}$ are the specific message passing parameters for each edge type $r \in \mathcal{R}$ and each of L HMPNN layers. Then defining $\mathcal{E}_r(j)$ as the set of edges of type r pointing to node j, the **Aggregate** function mean-pools messages by edge type, and concatenates:

$$\textbf{Aggregate}^{(l)}(\mathcal{E}(j)) = \mathop{\parallel}_{r\in\mathcal{R}} \frac{1}{|\mathcal{E}_r(j)|} \sum_{e\in\mathcal{E}_r(j)} \textbf{Message}^{(l)}(e) \tag{3}$$

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

$$\textbf{Transform}^{(l)}(j) = \alpha(W^{(l)}_{\tau(j)} \cdot \textbf{Aggregate}^{(l)}(\mathcal{E}(j))) \tag{4}$$

where $W_t^{(l)}$ are the specific transformation parameters for each node type $t \in \mathcal{T}$ and each of L HMPNN layers. By stacking L layers, HMPNN outputs highly contextualized final node representations, and the final node representations can be fed into another model to perform downstream heterogeneous network tasks, such as node classification or link prediction.

3.3 Problem definition

Using notations defined above, we formalize our novel transfer learning problem on HGs.

Problem Definition 1. Zero-shot cross-type transfer learning running on a HG:
In a given heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$ with node attributes $\mathcal{X} = \cup_{t\in\mathcal{T}}\mathcal{X}_t$, assume node types $\mathbf{s}$ *and* $\mathbf{t}$ *share a classification task $\{(i, y_i) : i \in \mathcal{V}_s, \mathcal{V}_t\}$. During the training phase, using labels $\{(i, y_i) : i \in \mathcal{V}_s\}$ only for source-type nodes, we train an HGNN model $\boldsymbol{f} : \boldsymbol{f}(\mathcal{G}, \mathcal{X}) = h_i$ to get node embeddings h_i for all nodes $i \in \mathcal{V}$ and a classifier $\boldsymbol{g} : \boldsymbol{g}(h_i) = \hat{y}_i$ to predict labels $\hat{y}_i$ from the node embeddings h_i. During the test phase, our task is to predict labels $\{(j, y_j) : j \in \mathcal{V}_t\}$ of target-type nodes where none of labels of target-type nodes were used for training.*

4 Cross-Type Feature Extractor Transformations in HGNNs

We define $f_t : \mathcal{G} \mapsto \mathbb{R}^d$ to be the "feature extractor" of a HGNN, which is composed of parameters participating to map input node attributes of type t into a shared feature space $\mathbb{R}^d$. In this section, we derive a strict transformation between feature extractors within a HMPNN. Specifically, for any two nodes i, j with types $\tau(i) = s$ and $\tau(j) = t$, we derive an expression for f_s in terms of f_t, and use that expression to inspire a trainable transfer learning module called KTN in the following section. For simplicity, throughout this section we ignore the activation $\alpha(\cdot)$ within the **Transform** function in Equation (4).

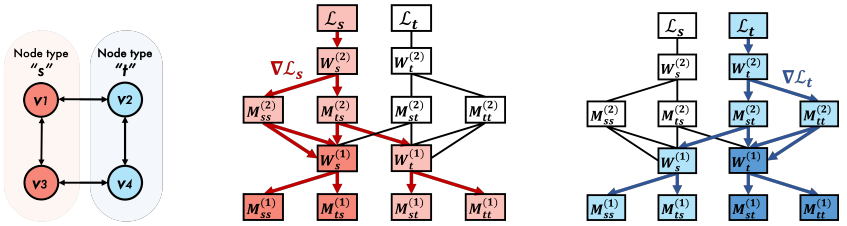
4.1 Feature extractors in HMPNNs

We first reason intuitively about the differences between f_s and f_t when $s \neq t$, using a toy heterogeneous graph shown in Figure 1(a). Consider nodes v_1 and v_2, noticing that $\tau(1) \neq \tau(2)$. Using HMPNN's equations (2)-(4) from Section 3.2, for any $l \in \{0, \ldots, L-1\}$ we have

$$h_1^{(l)} = W_s^{(l)}\left[M_{ss}^{(l)}\left(h_3^{(l-1)} \parallel h_1^{(l-1)}\right) \parallel M_{ts}^{(l)}\left(h_2^{(l-1)} \parallel h_1^{(l-1)}\right)\right] \tag{5}$$

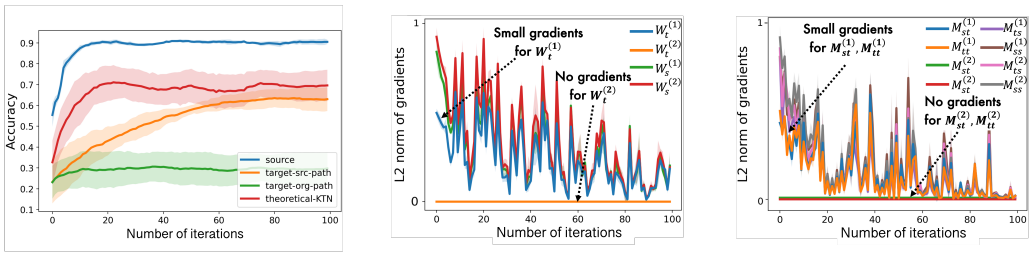
$$h_2^{(l)} = W_t^{(l)}\left[M_{st}^{(l)}\left(h_1^{(l-1)} \parallel h_2^{(l-1)}\right) \parallel M_{tt}^{(l)}\left(h_4^{(l-1)} \parallel h_2^{(l-1)}\right)\right] \tag{6}$$

where $h_j^{(0)} = x_j$. From these equations, we see that $h_1^{(l)}$ and $h_2^{(l)}$, which are features of different types, are extracted using *disjoint* sets of model parameters at l-th layer. In a 2-layer HMPNN, this creates unique gradient backpropagation paths between the two node types, as illustrated in Figures 1(b)-1(c). In other words, even though the same HMPNN 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.



(a) Toy graph (b) Gradient path for feature extractor f_s (c) Gradient path for feature extractor f_t

Figure 1: Illustration of a toy heterogeneous graph and the gradient paths for feature extractors f_s and f_t. Colored arrows in figures (b) and (c) show that the same HGNN nonetheless produces different gradient paths for each feature extractor. Color density of each box in (b) and (c) is proportional to the degree of participation of the corresponding parameter in each feature extractor.



(a) Test accuracy across various feature extractors (b) L2 norms of gradients of $W_{\tau(\cdot)}$ (c) L2 norms of gradients of $M_{\phi(\cdot)}$

Figure 2: HGNNs trained on a source domain underfit a target domain even on a "nice" heterogeneous graph. (a) Performance on the simulated heterogeneous graph for 4 kinds of feature extractors; (*source*: source extractor f_s on source domain, *target-src-path*: source extractor f_s on target domain, *target-org-path*: target extractor f_t on target domain, and *theoretical-KTN*: target extractor f_t on target domain using KTN.) (b-c) L2 norms of gradients of parameters $W_{\tau(\cdot)}$ and $M_{\phi(\cdot)}$ in HGNN models.

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 1(a) to have multiple nodes per-type, and multiple classes. To maximize the effects of having different feature extractors, we sample source and target nodes from the same feature distributions and each classes are well-separated in the both the graph and feature space (more details available in Appendix A.7.1).

On such a well-aligned heterogeneous 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 HMPNN model solely on s-type nodes, as shown in Figure 2(a) we find the test accuracy for s-type nodes to be high (90%, blue line) and the test accuracy for t-type nodes to be quite low (25%, green line). Now if instead we make the t-type nodes use the source feature extractor f_s, much more transfer learning is possible ($\sim 65\%$, orange line). This shows that the different feature extractors present in the HMPNN model result in the significant performance drop, and simply matching input data distributions can not solve the problem.

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

This case study 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.

4.3 Relationship between feature extractors in HMPNNs

We show that a HMPNN model provides different feature extractors for each node type. However, still, f_s and f_t are built inside one HMPNN model and interchange intermediate feature embeddings with each other. Both $H_s^{(L)}$ and $H_t^{(L)}$ (the output of f_s and f_t) are computed using the previous layer's intermediate embeddings $H_s^{(L-1)}, H_t^{(L-1)}$, and any other connected node type embeddings $H_x^{(L-1)}$ at the L-th HMPNN layer. Therefore $H_s^{(L)}$ and $H_t^{(L)}$ can be mathematically presented by each other using the $(L-1)$-th layer embeddings as connecting points, so do f_s and f_t. Based on this intuition, we derive a strict transformation between f_s and f_t, which will motivate the core domain adaptation component of our proposed KTN model.

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 HMPNN parameters as*

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

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

$$H_s^{(l)} = A_{ts}^* H_t^{(l)} Q_{ts}^* \tag{8}$$

where $a_{ts}(\cdot)$ and $q_{ts}(\cdot)$ are matrix functions that depend only on s, t.

The full proof of Theorem 1 can be found in Appendix A.1. Notice that in Equation 8, Q_{ts}^* acts as a macro-**Transform** operator that maps $H_t^{(L)}$ into the source domain, then A_{ts}^* aggregates the mapped embeddings into s-type nodes. In other words, Q_{ts}^* acts as a mapping matrix from the target domain to the source domain. To examine the implications, we run the same experiment as described in Section 4.2, while this time mapping the target features $H_t^{(L)}$ into the source domain by multiplying with Q_{ts}^* in Equation 8 before passing over to a task classifier. We see via the red line in Figure 2(a) that, with this mapping, the accuracy in the target domain becomes much closer to the accuracy in the source domain ($\sim$70%). Thus, we use this theoretical transformation as a foundation for our trainable HGNN domain adaptation module, introduced in the following section.

4.4 Generalized cross-type transformations for HGNNs

In this section we showed that a HMPNN feature extractor on the (label-abundant) source node type can be expressed in terms of the (label-scarce) target node type feature extractor, and this transformation enables cross-type zero-shot learning for the target node type. As most HGNNs have distinct feature extractors for each node types (even single-layer HGNNs, which have specialized parameters for each node/edge attribute projection layer), they will suffer from the under-trained target embeddings phenomena we showed in Section 4.2. For instance, in the meta-path based MAGNN model [8], meta-paths directing toward the target node types are generally less engaged in the source node feature computation and thus receive smaller gradients. While we cannot derive the exact cross-type transformation for all possible HGNNs, the core intuition in the HMPNN theory holds, namely that $H_s^{(L)}$ and $H_t^{(L)}$ are both computed using the previous layer's intermediate embeddings (see Section 4.3) across all HGNN models. This observation allows us to extend our KTN and apply it to almost any HGNN. We illustrate this by applying KTN to 6 different HGNN models in Section 6, where we see greatly increased target-type accuracy.

5 KTN: Trainable Cross-Type Transfer Learning for HGNNs

Inspired by these derivations we introduce our primary contribution, *Knowledge Transfer Networks*. We begin by noting Equation 8 in Theorem 1 has a similar form to a single-layer graph convolutional network [17] with a deterministic transformation matrix (Q_{ts}^*) and a combination of adjacency matrices directing from target node type t to source node type s (A_{ts}^*). Instead of hand-computing the mapping function Q_{ts}^* for arbitrary HGs and HGNNs (which would be intractable), we *learn* the mapping function by modelling Equation 8 as a trainable graph convolutional network, named the

Algorithm 1 Training step on a source domain

Require: heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$, node feature matrices $\mathcal{X}$, source node type s, target node type t, adjacency matrix A_{ts}, source node label matrix $\mathcal{Y}_s$.
Ensure: HGNN $\mathbf{f}$, classifier $\mathbf{g}$, KTN $\mathbf{t}_{\text{KTN}}$

$H_s^{(L)}, H_t^{(L)} = \mathbf{f}(\mathcal{G}, H^{(0)} = \mathcal{X})$
$H_t^* = \mathbf{t}_{KTN}(H_t^{(L)}) = A_{ts} H_t^{(L)} T_{ts}$
$\mathcal{L}_{\text{KTN}} = \left\| H_s^{(L)} - H_t^* \right\|_2$
$\mathcal{L} = \mathcal{L}_{\text{CL}}(\mathbf{g}(H_s^{(L)}), \mathcal{Y}_s) + \lambda \mathcal{L}_{\text{KTN}}$
Update $\mathbf{f}$, $\mathbf{g}$, $\mathbf{t}$ using $\nabla \mathcal{L}$

Algorithm 2 Test step on a target domain

Require: pretrained HGNN $\mathbf{f}$, classifier $\mathbf{g}$, KTN $\mathbf{t}_{\text{KTN}}$
Ensure: target node label matrix $\mathcal{Y}_t$

$H_t^{(L)} = \mathbf{f}(\mathcal{G}, H^{(0)} = \mathcal{X})$
$H_t^* = H_t^{(L)} T_{ts}$
return $\mathbf{g}(H_t^*)$

Knowledge Transfer Network, $\mathbf{t}_{\text{KTN}}(\cdot)$. KTN replaces Q_{ts}^* and A_{ts}^* in Equation 8 as follows:

$$\mathbf{t}_{\text{KTN}}(H_t^{(L)}) = A_{ts} H_t^{(L)} T_{ts} \tag{9}$$

$$\mathcal{L}_{\text{KTN}} = \left\| H_s^{(L)} - \mathbf{t}_{\text{KTN}}(H_t^{(L)}) \right\|_2 \tag{10}$$

where A_{ts} is an adjacency matrix from node type t to s, and T_{ts} is a trainable transformation matrix. By minimizing $\mathcal{L}_{\text{KTN}}$, T_{ts} is optimized to a mapping function of the target domain into the source domain.

5.1 Algorithm

We minimize a classification loss $\mathcal{L}_{\text{CL}}$ and a transfer loss $\mathcal{L}_{\text{KTN}}$ jointly with regard to a HGNN model $\mathbf{f}$, a classifier $\mathbf{g}$, and a knowledge transfer network $\mathbf{t}_{\text{KTN}}$ as follows:

$$\min_{\mathbf{f},\, \mathbf{g},\, \mathbf{t}_{\text{KTN}}} \mathcal{L}_{\text{CL}}(\mathbf{g}(\mathbf{f}(\mathcal{G}, \mathcal{X})_s), \mathcal{Y}_s) + \lambda \left\| \mathbf{f}(\mathcal{G}, \mathcal{X})_s - \mathbf{t}_{\text{KTN}}(\mathbf{f}(\mathcal{G}, \mathcal{X})_t) \right\|_2$$

where λ is a hyperparameter regulating the effect of $\mathcal{L}_{\text{KTN}}$; and $\mathbf{f}(\mathcal{G}, \mathcal{X})_s$ and $\mathbf{f}(\mathcal{G}, \mathcal{X})_t$ denote $H_s^{(L)}$ and $H_t^{(L)}$, respectively. Algorithm 1 describes a training step on the source domain. After computing the node embeddings $H_s^{(L)}$ and $H_t^{(L)}$, we map $H_t^{(L)}$ to the source domain using $\mathbf{t}_{\text{KTN}}$ and compute $\mathcal{L}_{\text{KTN}}$. Then, we update the models using gradients of $\mathcal{L}_{\text{CL}}$ (computed using only source labels) and $\mathcal{L}_{\text{KTN}}$. Algorithm 2 describes the test phase on the target domain. After we get node embeddings $H_t^{(L)}$ from the trained HGNN model, we map $H_t^{(L)}$ into the source domain using the trained transformation matrix T_{ts}. Finally we pass the transformed target embeddings H_t^* into the classifier 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 HGs via other node types (i.e., there is no A_{ts}). Appendix A.2 describes how we can easily extend KTN to cover domain adaption scenarios in this case.

6 Experiments

6.1 Datasets

Open Academic Graph (OAG). A dataset introduced in [44] composed of five types of nodes: papers (P), authors (A), institutions (I), venues (V), fields (F) 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. We construct three field-specific subgraphs from OAG: computer science, computer networks, and machine learning academic graphs.

Table 1: **Open Academic Graph on Computer Science field**. The *gain* column shows the relative gain of our method over using no domain adaptation (*Base* column). *o.o.m* denotes *out-of-memory* errors.

Task	Metric	Base	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%)**
	MRR	0.297	0.361	0.314	0.179	0.129	0.111	0.138	0.041	0.363	**0.629 (112%)**
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%)**
	MRR	0.318	0.508	0.544	0.229	0.270	0.090	0.047	0.022	0.507	**0.711 (123%)**
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%)**
	MRR	0.364	0.413	0.458	0.341	0.205	0.253	0.327	0.011	0.389	**0.698 (92%)**
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%)**
	MRR	0.133	0.365	0.345	0.094	0.241	0.444	0.144	0.006	0.006	**0.586 (340%)**
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%)**
	MRR	0.121	0.118	o.o.m	0.140	o.o.m	o.o.m	0.020	o.o.m	0.143	**0.2248 (86%)**
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%)**
	MRR	0.095	0.052	o.o.m	0.106	0.032	0.045	0.017	o.o.m	0.132	**0.242 (155%)**
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 (16%)**
	MRR	0.310	0.296	0.198	0.223	0.128	0.097	0.110	o.o.m	0.227	**0.399 (29%)**
V-A (L2)	**NDCG**	0.235	0.249	0.251	0.214	0.197	0.205	0.217	o.o.m	0.119	**0.252 (7%)**
	MRR	0.129	0.157	0.161	0.090	0.044	0.068	0.085	o.o.m	0.000	**0.166 (28%)**

Table 2: **PubMed graph**. The *gain* column shows the relative gain over using *Base* column.

Task	Metric	Base	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%)**
	MRR	0.372	0.425	0.414	0.397	0.428	0.443	0.388	0.389	0.336	**0.499 (34%)**
G-D	**NDCG**	0.596	0.599	0.577	0.599	0.581	0.606	0.578	0.576	0.580	**0.662 (11%)**
	MRR	0.354	0.362	0.332	0.356	0.337	0.362	0.340	0.351	0.353	**0.445 (26%)**

PubMed.[39] A network composed of of four types of nodes: genes (G), diseases (D), chemicals (C), and species (S), and their corresponding relationships. Gene and chemical nodes have graph structure-based attributes, while disease and species nodes have text-based attributes. Each gene and disease is labeled with a set of diseases they belong to or cause.

Synthetic heterogeneous graphs. We generate stochastic block models [1] with multiple node/edge types. We label each node types with the same set of classes. Then we control feature/edge distributions within/between node types by manipulating feature/edge signal-to-noise ratio within/between classes. A complete definition of the generative model is given in Appendix A.7.

6.2 Baselines

We compare KTN with two MMD-based DA methods (DAN [18], JAN [20]), three adversarial DA methods (DANN [9], CDAN [19], CDAN-E [19]), one optimal transport-based method (WDGRL [27]), and two traditional graph mining methods (LP and EP [45]). For KTN and DA methods, we use HMPNN as their base HGNN model. More information of each method is in Appendix A.9.

6.3 Zero-shot transfer learning

We run 18 different zero-shot transfer learning tasks across three OAG and PubMed graphs. We run each experiment 3 times and report the average value. Due to the space limitation, we report the standard deviations and results on OAG-computer networks and OAG-machine learning in Appendix A.3. Each heterogeneous graph has the same node classification task for both source and target node types. During training, we are given 1) the heterogeneous graph structure information (i.e., adjacency matrices), 2) input node attribute matrices for all node types, and 3) labels on source-type nodes for the classification task. During the test phase, we predict labels on target-type nodes for the same classification task. The performance is evaluated by NDCG and MRR — widely adopted ranking metrics [14, 15].

In Tables 1 and 2, our proposed method 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 base accuracy using the model pretrained on the source domain without any domain adaptation (3rd column, *Base*), the results are even more impressive. We see our method 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 KTN on zero-shot transfer learning tasks on a heterogeneous graph. We mention that venue and author node types are not directly connected in the OAG graphs (Figure 5(b) in Appendix), but KTN successfully transfer knowledge by passing intermediate node types.

Table 3: **KTN on different HGNN models.** The *Source* column shows accuracy on for source node types. *Base* and *KTN* columns show accuracy for target node types without/with using KTN, respectively. The *Gain* column shows the relative gain of our method over using no domain adaptation.

HGNN type	Metric	P-A (L1) Source	P-A (L1) Base	P-A (L1) KTN	P-A (L1) Gain	A-P (L1) Source	A-P (L1) Base	A-P (L1) KTN	A-P (L1) Gain
R-GCN	**NDCG**	0.765	0.337	0.577	**71.12%**	0.648	0.388	0.647	**66.82%**
	MRR	0.757	0.236	0.587	**148.73%**	0.623	0.270	0.611	**126.18%**
HAN	**NDCG**	0.476	0.179	0.520	**190.56%**	0.515	0.182	0.512	**181.33%**
	MRR	0.416	0.047	0.497	**960.55%**	0.509	0.055	0.527	**850.90%**
HGT	**NDCG**	0.757	0.294	0.574	**95.07%**	0.670	0.283	0.581	**104.83%**
	MRR	0.749	0.178	0.563	**216.17%**	0.670	0.149	0.565	**279.52%**
MAGNN	**NDCG**	0.657	0.463	0.574	**24.01%**	0.676	0.557	0.622	**11.68%**
	MRR	0.631	0.378	0.556	**47.33%**	0.680	0.509	0.592	**16.14%**
MPNN	**NDCG**	0.602	0.443	0.590	**33.11%**	0.646	0.307	0.621	**101.92%**
	MRR	0.572	0.319	0.575	**80.10%**	0.660	0.145	0.595	**311.42%**
HMPNN	**NDCG**	0.789	0.399	0.623	**56.14%**	0.671	0.401	0.733	**82.88%**
	MRR	0.777	0.297	0.629	**111.86%**	0.661	0.318	0.711	**123.30%**

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 [19, 27]. These reversed results are a consequence of HGNN's unique feature extractors for each domains. Adversarial- and optimal transport-based methods define separate losses for source and target feature extractors (which are not separated in their shared feature extractor assumption), resulting in divergent gradients between different feature extractors and poor domain adaption performance. This shows again the importance of considering different feature extractors in HGNNs. More analysis can be found in Appendix A.4.

6.4 Generality of KTN

Here, we use 6 different HGNN models, R-GCN [26], HAN [35], HGT [15], MAGNN [8], MPNN [10], and HMPNN. MPNN maps all node types to the shared embedding space using projection matrices at the beginning then applies MPNN layers designed for homogeneous graphs. In Table 3, KTN improves accuracy on the target nodes across all HGNN models by up to 960%. This shows the strong generality of KTN. More results and analysis can be found in Appendix A.5.

6.5 Sensitivity analysis

Using our synthetic heterogeneous graph generator, we generate non-trivial 2-type heterogeneous graphs to examine how the feature and edge distributions affect the performance of KTN and other baselines. We generate a *range* of test-case scenarios by manipulating (1) signal-to-noise ratio σ_e of within-class edge distributions and (2) signal-to-noise ratio σ_f of within-class feature distributions across all of the (a) source-source ($s \leftrightarrow s$), (b) target-target ($t \leftrightarrow t$), and (c) source-target ($s \leftrightarrow t$) relationships.

For instance, in Figure 3, for each edge type ($s \leftrightarrow s$, $t \leftrightarrow t$, and $s \leftrightarrow t$, differentiated by colors), there are two different types of edges, edges within the same class (plain line) and edges across different classes (dotted line). For each edge type, we manipulate σ_e by changing the ratio of within-class and cross-class edges, and σ_f by diverging feature distributions between classes. Thus there will be 6 signal-to-noise ratios in total. A higher signal-to-noise ratio for a particular data dimension (edges or features) across a particular relationship $r \in \{s \leftrightarrow s,\ t \leftrightarrow t,\ s \leftrightarrow t\}$ means that classes are more *separable* in that data dimension, when comparing within r, and hence easier for HGNNs. Note that while tuning one $\sigma_{(\cdot)}$ on the range $[1.0, 10.0]$, the remaining five $\sigma_{(\cdot)}$ are held at 10.0. Additionally, we vary $\sigma_{(\cdot)}$ across two scenarios: (I) "easy": source and target node types have same number of clusters and same feature dimensions, (II) "hard" source and target node types have different number of clusters and feature dimensions. Note that clusters and classes are different concepts in this experiment; several clusters could have the same class label.

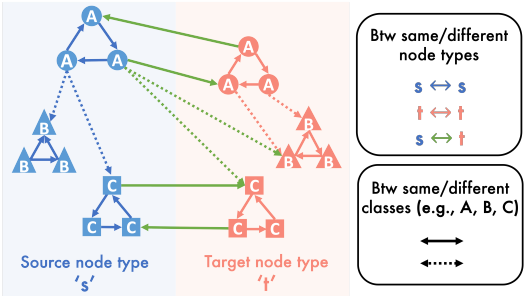
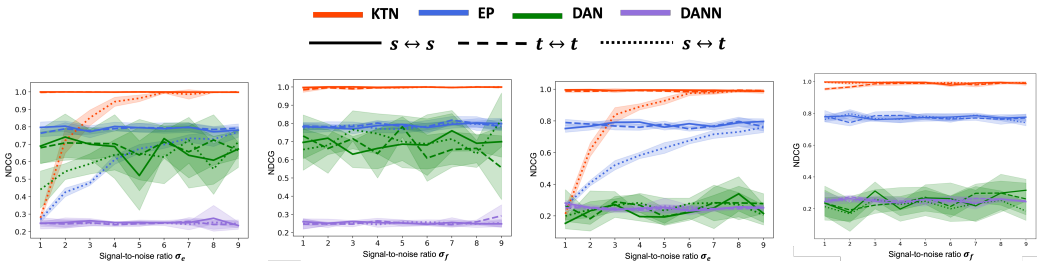


Figure 3: Synthetic HG generator



(a) Edge dist. (easy) (b) Feature dist. (easy) (c) Edge dist. (hard) (d) Feature dist. (hard)

Figure 4: Effects of edge and feature distributions across classes and types in heterogeneous graphs.

Figures 4(a) and 4(c) show results from changing σ_e across the three relation types. We see that KTN is affected only by σ_e across the $s \leftrightarrow t$ (cross-types) relationship, which accords with our theory, since KTN exploits the between-type adjacency matrix. Surprisingly, as seen in Figures 4(b) and 4(d), we do not find a similar dependence of KTN on σ_f, which shows that KTN is robust by learning purely from edge homophily in the absence of feature homophily. Regarding the performance of other baselines, EP shows similar tendencies as KTN— only affected by cross-type σ_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 exploit the (unlabelled) target features. DAN and DANN, which do not exploit cross-type edges, are not affected by cross-type σ_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.

7 Conclusion

In this work, we proposed the first cross-type zero-shot transfer learning method for heterogeneous graphs. Our method, Knowledge Transfer Networks (KTN) for Heterogeneous Graph Neural Networks, transfers knowledge from *label-abundant* node types to *label-scarce* node types. We illustrate KTN handily improves HGNN performances up to 960% for zero-labeled node types across 6 different HGNN models and outperforms many challenging baselines up to 73% higher in MRR. Future work in the area includes filtering noisy edges between source and target domains and making KTN more robust and less dependent on structure of given noisy heterogeneous graphs.

Limitation Statement Our transfer learning method is limited to node types sharing the same task (i.e., the same classifier). We plan to extend our work to transfer knowledge between different tasks running on different node types on a heterogeneous graph.

Societal Impact Statement KTN allows organizations to learn better from their own graph data, leveraging its structure without requiring external information. We believe this has a number of positive applications (preserving model quality without needing extra datasets). However like all modeling improvements, its true impact depends on what modeling tasks the technique is applied to.

8 Acknowledgement

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

References

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

[2] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

[3] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19:137, 2007.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[5] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[6] Oleksandr Ferludin, Arno Eigenwillig, Martin Blais, Dustin Zelle, Alvaro Sanchez-Gonzalez, Sibon Li, Sami Abu-el Haija, Peter Battaglia, Neslihan Bulut, Jonathan Halcrow, Filipe Miguel Gonçalves de Almeida, Silvio Lattanzi, André Linhares, Brandon Mayer, Vahab Mirrokni, John Palowitch, Mihir Paradkar, Jennifer She, Anton Tsitsulin, Kevin Villela, Lisa Wang, David Wong, and Bryan Perozzi. Tf-gnn: Graph neural networks in tensorflow. 2022.

[7] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[8] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020.

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[11] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[12] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[13] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[14] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1857–1867, 2020.

[15] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[16] Tiancheng Huang, Ke Xu, and Donglin Wang. Da-hgt: Domain adaptive heterogeneous graph transformer. *arXiv preprint arXiv:2012.05688*, 2020.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[19] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. *arXiv preprint arXiv:1705.10667*, 2017.

[20] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *International conference on machine learning*, pages 2208–2217. PMLR, 2017.

[21] Yadan Luo, Zijian Wang, Zi Huang, and Mahsa Baktashmotlagh. Progressive graph learning for open-set domain adaptation. In *International Conference on Machine Learning*, pages 6468–6478. PMLR, 2020.

[22] Xinhong Ma, Tianzhu Zhang, and Changsheng Xu. Gcan: Graph convolutional adversarial network for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8266–8276, 2019.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[24] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.

[25] Ievgen Redko, Amaury Habrard, and Marc Sebban. Theoretical analysis of domain adaptation with optimal transport. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 737–753. Springer, 2017.

[26] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[27] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246, 2015.

[29] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[30] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159, 2012.

[31] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998, 2008.

[32] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

[33] Anton Tsitsulin, Benedek Rozemberczki, John Palowitch, and Bryan Perozzi. Synthetic graph generation to benchmark graph learning. *WWW'21, Workshop on Graph Learning Benchmarks*, 2021.

[34] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[35] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

[36] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, 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*, pages 38–45, 2020.

[37] Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *Proceedings of The Web Conference 2020*, pages 1457–1467, 2020.

[38] Qitian Wu, Chenxiao Yang, and Junchi Yan. Towards open-world feature extrapolation: An inductive graph learning approach. *Advances in Neural Information Processing Systems*, 34:19435–19447, 2021.

[39] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[40] Shuwen Yang, Guojie Song, Yilun Jin, and Lun Du. Domain adaptive classification on heterogeneous information networks. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1410–1416, 2021.

[41] Yaming Yang, Ziyu Guan, Jianxin Li, Wei Zhao, Jiangtao Cui, and Quan Wang. Interpretable and efficient heterogeneous graph convolutional network. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[42] Jiaxuan You, Xiaobai Ma, Yi Ding, Mykel J Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. *Advances in Neural Information Processing Systems*, 33:19075–19087, 2020.

[43] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 793–803, 2019.

[44] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, et al. Oag: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2585–2595, 2019.

[45] Xiaojin Zhu. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We describe it in Section 7. We also show which environments are ideal for our method via sensitivity test in Section 6.5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We describe it in Section 7. We also mention about possible private information leakage in graph-to-graph transfer learning in Section 1. By proposing transfer learning between node types in one heterogeneous graph, we can alleviate this issue.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We mention the full set of assumptions for Theorem 1 in Section 4.3.
   (b) Did you include complete proofs of all theoretical results? [Yes] The proof for Theorem 1 is described in Appendix A.1.

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide a URL to our code in Appendix A.11.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] They are specified in Appendix A.8 and A.11.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] They are specified in Appendix A.3 and A.5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] They are specified in Appendix A.11.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite public domain adaptation library ADA and heterogeneous graph neural network library OpenHGNN that we have used in our experiment in Appendix A.11.
   (b) Did you mention the license of the assets? [N/A] All codes and data we used are public.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide a URL to our code in Appendix A.11.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We do not use any personal data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not use any personal data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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:

$$\textbf{Message}^{(l)}(i,j) = M^{(l)}_{\phi(i,j)} h^{(j)}_i. \tag{11}$$

This lets the Theorem match the experimental results shown in Figure 2, 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 [12], 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 networks 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 (through with more algebraically-dense forms of a_{ts} and q_{ts}).

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

$$\mathcal{E}_r(j) := \{(i,j) : i \in \mathcal{V}, (i,j) = r\}. \tag{12}$$

We write $A_{x_1 x_2}$ 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 $\bar{A}_{x_1 x_2}$ to denote the same matrix divided by its sum. $H^{(l)}_x$ is the (row-wise) $n_x \times d_l$ embedding matrix of x-type nodes at layer l.

We first compute the *l-th* output $g^{(l)}_j$ of the **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 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 $\mathcal{R}$. Thus we can write $g^{(l)}_j$ as

$$\begin{aligned}
g^{(l)}_j &= \mathop{\Big\|}_{r \in \mathcal{R}} \tfrac{1}{|\mathcal{E}_r(j)|} \sum_{e \in \mathcal{E}_r(j)} \textbf{Message}^{(l)}(e) \\
&= \mathop{\Big\|}_{x \in \mathcal{T}} \tfrac{1}{|\mathcal{E}_{xs}(j)|} \sum_{e \in \mathcal{E}_{xs}(j)} \textbf{Message}^{(l)}(e) \\
&= \mathop{\Big\|}_{x \in \mathcal{T}} \tfrac{1}{|\mathcal{E}_{xs}(j)|} \sum_{(i,j) \in \mathcal{E}_{xs}(j)} M^{(l)}_{xs} h^{(l-1)}_i \\
&= \mathop{\Big\|}_{x \in \mathcal{T}} \tfrac{1}{|\mathcal{E}_{xs}(j)|} M^{(l)}_{xs} \sum_{(i,j) \in \mathcal{E}_{xs}(j)} h^{(l-1)}_i \\
&= \mathop{\Big\|}_{x \in \mathcal{T}} M^{(l)}_{xs} \left(H^{(l-1)}_x\right)' \bar{A}^{(j)}_{xs},
\end{aligned}$$

where $\bar{A}^{(j)}_{xs}$ denotes the *j-th* column of $\bar{A}_{xs}$. Notice that

$$h^{(l)}_j = \textbf{Transform}^{(l)}(j) = W^{(l)}_s g^{(l)}_j, \tag{13}$$

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

$$h^{(l)}_j = \sum_{x \in \mathcal{T}} W^{(l)}_{xs} M^{(l)}_{xs} \left(H^{(l-1)}_x\right)' \bar{A}^{(j)}_{xs}. \tag{14}$$

Algorithm 3 Training step for one minibatch (indirect version)

Require: heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R})$, node feature matrices X, adjacency matrices $A_{xy} \forall (x, y) \in \mathcal{R}$, source node type s, target node type t, source node label matrix Y_s.
Ensure: HGNN **f**, classifier **g**, KTN $\mathbf{t}_{\text{KTN}}$

$H_s^{(L)}, H_t^{(L)} = \mathbf{f}(H^{(0)} = X, \mathcal{G})$, $H_t^* = \mathbf{0}$
for each meta-path $p = t \to s$ **do**
  $x = t$, $Z = H_t^{(L)}$
  for each node type $y \in p$ **do**
    $Z = A_{xy} Z T_{xy}$
    $x = y$
  end for
  $H_t^* = H_t^* + Z$
end for
$\mathcal{L}_{\text{KTN}} = \left\| H_s^{(L)} - H_t^* \right\|_2$
$\mathcal{L} = \mathcal{L}_{\text{CL}}(\mathbf{g}(H_s^{(L)}), Y_s) + \lambda \mathcal{L}_{\text{KTN}}$
Update **f**, **g**, $\mathbf{t}_{\text{KTN}}$ using $\nabla \mathcal{L}$

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

Require: pretrained HGNN **f**, classifier **g**, KTN $\mathbf{t}_{\text{KTN}}$
Ensure: target node label matrix Y_t

$H_t^{(L)} = \mathbf{f}(H^{(0)} = X, \mathcal{G})$, $H_t^* = \mathbf{0}$
for each meta-path $p = t \to s$ **do**
  $x = t$, $Z = H_t^{(L)}$
  for each node type $y \in p$ **do**
    $X = Z T_{xy}$
    $x = y$
  end for
  $H_t^* = H_t^* + Z$
end for
return $\mathbf{g}(H_t^*)$

Defining the operator $Q_{xs}^{(l)} := \left(W_{xs}^{(l)} M_{xs}^{(l)} \right)'$, this implies that

$$
\begin{aligned}
H_s^{(l)} &= \sum_{x \in \mathcal{T}} \bar{A}_{xs} H_x^{(l-1)} Q_{xs}^{(l-1)} \\
&= [\bar{A}_{x_1 s}, \dots, \bar{A}_{x_T s}] \begin{bmatrix} H_{x_1}^{(l-1)} & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & H_{x_T}^{(l-1)} \end{bmatrix} \begin{bmatrix} Q_{x_1 s}^{(l-1)} \\ \dots \\ Q_{x_T s}^{(l-1)} \end{bmatrix} \\
&= \bar{A}_{\cdot s} H_{\cdot}^{(l-1)} Q_{\cdot s}^{(l-1)}
\end{aligned}
$$

Similarly we have $H_t^{(l)} = \bar{A}_{\cdot t} H_{\cdot}^{(l-1)} Q_{\cdot t}^{(l-1)}$. Since $H_s^{(l)}$ and $H_t^{(l)}$ share the term $H_{\cdot}^{(l-1)}$, we can write

$$
H_s^{(l)} = \bar{A}_{\cdot s} \bar{A}_{\cdot t}^{-1} H_t^{(l)} (Q_{\cdot t}^{(l-1)})^{-1} Q_{\cdot s}^{(l-1)}, \tag{15}
$$

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 $\mathbf{t}_{\text{KTN}}(H_t^{(L)})$ to $(A_{xs}(A_{tx} H_t^{(L)} T_{tx}) T_{xs})$ where $T_{tx} T_{xs}$ becomes a mapping function from target to source domains. Algorithms 3 and 4 show how to extend KTN. For every step $(x \to y)$ in a meta-path $(t \to \dots \to s)$ connecting 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 5(b)). As

Table 4: **Open Academic Graph on Computer Science field**. The **gain** column shows the relative gain of our method over using no domain adaptation (*Base* column). **o.o.m** denotes *out-of-memory* errors.

Task	Metric	Base	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	0.26	0.178	0.425	**0.623 (56)**
	std	0.010	0.012	0.032	0.009	0.021	0.014	0.021	0.000	0.006	0.004
	MRR	0.297	0.361	0.314	0.179	0.129	0.111	0.138	0.041	0.363	**0.629 (112)**
	std	0.024	0.006	0.041	0.011	0.032	0.031	0.033	0.000	0.005	0.004
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)**
	std	0.003	0.012	0.014	0.034	0.049	0.046	0.025	0.000	0.002	0.007
	MRR	0.318	0.508	0.544	0.229	0.27	0.09	0.047	0.022	0.507	**0.711 (123)**
	std	0.001	0.029	0.028	0.093	0.117	0.037	0.029	0.000	0.003	0.009
A-V (L1)	**NDCG**	0.459	0.457	0.47	0.382	0.346	0.359	0.403	0.207	0.461	**0.671 (46)**
	std	0.030	0.033	0.036	0.015	0.029	0.109	0.024	0.000	0.002	0.004
	MRR	0.364	0.413	0.458	0.341	0.205	0.253	0.327	0.011	0.389	**0.698 (92)**
	std	0.079	0.08	0.093	0.05	0.098	0.143	0.044	0.000	0.004	0.003
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)**
	std	0.045	0.012	0.007	0.004	0.048	0.039	0.004	0.000	0.006	0.005
	MRR	0.133	0.365	0.345	0.094	0.241	0.444	0.144	0.006	0.006	**0.586 (340)**
	std	0.074	0.027	0.017	0.011	0.103	0.115	0.018		0.007	0.010
P-A (L2)	**NDCG**	0.229	0.23	o.o.m	0.239	o.o.m	o.o.m	0.168	o.o.m	0.215	**0.282 (23)**
	std	0.005	0.003	-	0.006	-	-	0.007	-	0.004	0.002
	MRR	0.121	0.118	o.o.m	0.14	o.o.m	o.o.m	0.02	o.o.m	0.143	**0.2248 (86)**
	std	0.019	0.004	-	0.01	-	-	0.006	-	0.003	0.003
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)**
	std	0.006	0.009	-	0.006	0.019	0.022	0.012	-	0.004	0.001
	MRR	0.095	0.052	o.o.m	0.106	0.032	0.045	0.017	o.o.m	0.132	**0.242 (155)**
	std	0.009	0.022	-	0.016	0.018	0.027	0.008	-	0.005	0.002
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 (16)**
	std	0.003	0.034	0.014	0.013	0.011	0.058	0.011	-	0.002	0.003
	MRR	0.310	0.296	0.198	0.223	0.128	0.097	0.11	o.o.m	0.227	**0.399 (29)**
	std	0.004	0.109	0.047	0.065	0.003	0.096	0.034	-	0.001	0.015
V-A (L2)	**NDCG**	0.235	0.249	0.251	0.214	0.197	0.205	0.217	o.o.m	0.119	**0.252 (7)**
	std	0.002	0.002	0.006	0.004	0.008	0.004	0.002	-	0.001	0.007
	MRR	0.130	0.157	0.161	0.09	0.044	0.068	0.085	o.o.m	0.000	**0.166 (28)**
	std	0.010	0.011	0.009	0.015	0.007	0.009	0.005	-	0.000	0.012

shown in Tables 4, 6, and 7, 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 KTN varying the length of meta-paths. In Table 8, 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 KTN fails to transfer anything with a meta-path shorter than the minimum. Using the minimum length of meta-paths is enough for KTN.

A.3 More results for Zero-shot Domain Adaption in Section 6.3

We show the zero-shot domain adaptation results with error bars on OAG-computer science and Pubmed in Tables 4 and 5. We also present the results with error bars on OAG-computer networks and OAG-machine learning in Tables 6 and 7, respectively. Across all tasks and graphs, our proposed method KTN consistently outperforms all baselines.

A.4 Analysis for Baselines in Section 6.3

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 [19, 27]. These reversed results are a consequence of HGNN's unique feature extractors for source and target domains. When f_s and f_t denote feature extractors for source and target domains, respectively, DANN and CDAN define their adversarial losses as a cross entropy loss ($\mathbb{E}[\log f_s] - \mathbb{E}[\log f_t]$) where gradients of the subloss $\mathbb{E}[\log f_s]$ are passed only back to f_s, while gradients of the subloss $\mathbb{E}[\log f_t]$ are passed only back to f_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, WDGRL 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

Table 5: **PubMed**

Task	Metric	Base	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)**
	std	0.004	0.013	0.028	0.008	0.078	0.015	0.022	0.000	0.004	0.005
	MRR	0.372	0.425	0.414	0.397	0.428	0.443	0.388	0.389	0.336	**0.499 (34)**
	std	0.003	0.007	0.054	0.013	0.066	0.027	0.035	0.000	0.003	0.006
G-D	**NDCG**	0.596	0.599	0.577	0.599	0.581	0.606	0.578	0.576	0.580	**0.662 (11)**
	std	0.007	0.020	0.032	0.011	0.054	0.019	0.019	0.000	0.011	0.003
	MRR	0.354	0.362	0.332	0.356	0.337	0.362	0.340	0.351	0.353	**0.445 (26)**
	std	0.005	0.015	0.019	0.008	0.023	0.031	0.015	0.000	0.008	0.002

Table 6: **Open Academic Graph on Computer Network field**

Task	Metric	Base	DAN	JAN	DANN	CDAN	CDAN-E	WDGRL	LP	EP	KTN (gain%)
P-A (L2)	**NDCG**	0.331	0.344	o.o.m	0.335	o.o.m	o.o.m	0.287	0.221	0.270	**0.382 (16)**
	std	0.004	0.005	-	0.004	-	-	0.012	0.000	0.003	0.004
	MRR	0.250	0.277	o.o.m	0.280	o.o.m	o.o.m	0.199	0.130	0.270	**0.360 (44)**
	std	0.024	0.012	-	0.007	-	-	0.004	0.000	0.003	0.010
A-P (L2)	**NDCG**	0.313	0.290	o.o.m	0.250	0.234	0.168	0.266	0.114	0.319	**0.364 (17)**
	std	0.002	0.023	-	0.021	0.041	0.025	0.030	0.000	0.004	0.003
	MRR	0.250	0.233	o.o.m	0.130	0.116	0.051	0.212	0.038	0.296	**0.368 (47)**
	std	0.015	0.039	-	0.051	0.069	0.037	0.061	0.000	0.005	0.004
A-V (L2)	**NDCG**	0.539	0.521	0.519	0.510	0.467	0.362	0.471	0.232	0.443	**0.567 (5)**
	std	0.012	0.031	0.008	0.022	0.008	0.045	0.024	0.000	0.002	0.008
	MRR	0.584	0.528	0.461	0.510	0.293	0.294	0.365	0.000	0.406	**0.628 (8)**
	std	0.042	0.015	0.011	0.054	0.013	0.088	0.019	0.000	0.004	0.016
V-A (L2)	**NDCG**	0.256	0.343	0.345	0.265	0.328	0.316	0.263	0.133	0.119	**0.341 (33)**
	std	0.006	0.012	0.005	0.005	0.005	0.003	0.003	0.000	0.001	0.005
	MRR	0.117	0.296	0.286	0.151	0.285	0.275	0.147	0.000	0.000	**0.281 (141)**
	std	0.020	0.009	0.004	0.009	0.006	0.008	0.009	0.000	0.000	0.014

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.

JAN, CDAN, and CDAN-E often show out of memory issues in Tables 4, 6, and 7. 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 KTN up to 60% absolute points of MRR, showing the limitation of not using target node attributes.

A.5 More results for Generality of KTN in Section 6.4

We show KTN performance on 6 different types of HGNN models across 4 different zero-shot domain adaptation tasks on the OAG-computer science dataset in Tables 9 and . Descriptions of each HGNN model can be found in Appendix A.10. While KTN consistently improves all HGNN models' performance on zero-labeled node types using labels rooted at other node types, the magnitude of improvements varies. While HAN sees up to 4958% (V-A (L1) task in Table), MAGNN is improved by up to 47% (P-A(L1) task in Table3) or sees no improvement in A-V(L1) task in Table. This gap stems from how many parameters each HGNN model shares across node types. HAN does not share any parameters during message-passing operations (every parameters are specialized to each meta-path), while MAGNN shares the transformation matrices across all node types at every layer. By sharing more parameters with other node types, the gradients are more likely passed to target node type-specific parameters, leaving less room for improvement by KTN. However, KTN is still necessary for any HGNN models. MPNN who shares all parameters except the projection matrices that map different input attributes into the same embedding space at the beginning still sees improvements by up to 311%. Again, these experimental results show the impact of having different feature extractors for each node type in HGNN models.

Table 7: **Open Academic Graph on Machine Learning field**

Task	Metric	Base	DAN	JAN	DANN	CDAN	CDAN-E	WDGRL	LP	EP	KTN (gain%)
P-A (L2)	**NDCG**	0.268	0.290	o.o.m	0.291	o.o.m	0.249	0.232	0.272	0.215	**0.318 (19)**
	std	0.002	0.009	-	0.004	-	0.005	0.004	0.000	0.002	0.004
	MRR	0.134	0.220	o.o.m	0.222	o.o.m	0.095	0.098	0.195	0.143	**0.269 (102)**
	std	0.006	0.020	-	0.026	-	0.003	0.037	0.000	0.003	0.006
A-P (L2)	**NDCG**	0.261	0.225	o.o.m	0.234	0.228	0.241	0.241	0.119	0.267	**0.319 (22)**
	std	0.002	0.009	-	0.004	0.005	0.011	0.002	0.000	0.001	0.005
	MRR	0.207	0.127	o.o.m	0.155	0.152	0.095	0.182	0.035	0.214	**0.287 (39)**
	std	0.018	0.042	-	0.008	0.009	0.003	0.017	0.000	0.012	0.011
A-V (L2)	**NDCG**	0.465	0.493	0.463	0.477	0.408	0.422	0.393	0.224	0.424	**0.538 (16)**
	std	0.006	0.004	0.003	0.003	0.006	0.013	0.005	0.000	0.005	0.004
	MRR	0.469	0.542	0.537	0.519	0.412	0.240	0.213	0.001	0.391	**0.632 (35)**
	std	0.039	0.008	0.005	0.003	0.015	0.008	0.009	0.000	0.021	0.006
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)**
	std	0.006	0.011	0.009	0.004	0.003	0.002	0.004	0.000	0.003	0.007
	MRR	0.131	0.212	0.199	0.086	0.085	0.129	0.118	0.000	0.000	**0.227 (73)**
	std	0.016	0.023	0.013	0.005	0.021	0.007	0.012	0.000	0.000	0.015

Table 8: **Meta-path length in KTN:** increasing the meta-path longer than the minimum does not bring significant improvement to KTN. Note that the minimum length of meta-paths in the A-V (L1) task is 2.

Task	**P-A (L1)**		**A-V (L1)**	
Meta-path length	**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

A.6 Effect of trade-off coefficient λ

We examine the effect of λ on domain adaptation performance. In Table 10, as λ decreases, target accuracy decreases as expected. Source accuracy also sees small drops since $\mathcal{L}_{\text{KTN}}$ functions as a regularizer; by removing the regularization effect, source accuracy decreases. When λ becomes large, both source and target accuracy drop significantly. Source accuracy drops since the effect of $\mathcal{L}_{\text{KTN}}$ becomes larger than the classification loss $\mathcal{L}_{\text{CL}}$. Even the effect of transfer learning become larger by having larger λ, since the source accuracy which will be transferred to the target domain is low, the target accuracy is also low. Thus we set λ to 1 throughout the experiments.

A.7 Synthetic Heterogeneous Graph Generator

Our synthetic heterogeneous graph generator is based on attributed Stochastic Block Models (SBM) [32, 33], 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 $K_T := \min_t\{k_t\}$-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 6.5, we describe the manipulation of multiple σ_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 σ_f, while keeping the within-cluster variance fixed. Note that features of

Table 9: **KTN on different HGNN models**: The *Source* column shows accuracy on source node types. *Base* and KTN columns show accuracy on target node types without/with using KTN, respectively. The *Gain* column shows the relative gain of our method over using no domain adaptation.

		P-A (L1)				A-P (L1)			
HGNN type	**Metric**	**Source**	**Base**	**KTN**	**Gain%**	**Source**	**Base**	**KTN**	**Gain%**
R-GCN	**NDCG**	0.765	0.337	0.577	**71.12**	0.648	0.388	0.647	**66.82**
	std	0.004	0.005	0.002		0.006	0.007	0.004	
	MRR	0.757	0.236	0.587	**148.73**	0.623	0.270	0.611	**126.18**
	std	0.002	0.003	0.001		0.005	0.008	0.004	
HAN	**NDCG**	0.476	0.179	0.520	**190.56**	0.515	0.182	0.512	**181.33**
	std	0.004	0.006	0.003		0.004	0.009	0.011	
	MRR	0.416	0.047	0.497	**960.55**	0.509	0.055	0.527	**850.90**
	std	0.001	0.002	0.002		0.005	0.004	0.005	
HGT	**NDCG**	0.757	0.294	0.574	**95.07**	0.670	0.283	0.581	**104.83**
	std	0.002	0.003	0.004		0.001	0.003	0.009	
	MRR	0.749	0.178	0.563	**216.17**	0.670	0.149	0.565	**279.52**
	std	0.005	0.007	0.001		0.002	0.007	0.006	
MAGNN	**NDCG**	0.657	0.463	0.574	**24.01**	0.676	0.557	0.622	**11.68**
	std	0.003	0.001	0.003		0.001	0.001	0.003	
	MRR	0.631	0.378	0.556	**47.33**	0.680	0.509	0.592	**16.14**
	std	0.003	0.002	0.004		0.001	0.002	0.005	
MPNN	**NDCG**	0.602	0.443	0.590	**33.11**	0.646	0.307	0.621	**101.92**
	std	0.002	0.002	0.001		0.005	0.013	0.004	
	MRR	0.572	0.319	0.575	**80.10**	0.660	0.145	0.595	**311.42**
	std	0.001	0.003	0.005		0.002	0.024	0.003	
H-MPNN	**NDCG**	0.789	0.399	0.623	**56.14**	0.671	0.401	0.733	**82.88**
	std	0.001	0.005	0.001		0.003	0.005	0.009	
	MRR	0.777	0.297	0.629	**111.86**	0.661	0.318	0.711	**123.30**
	std	0.003	0.001	0.002		0.007	0.004	0.008	

		V-A (L1)				A-V (L1)			
HGNN type	**Metric**	**Source**	**Base**	**KTN**	**Gain%**	**Source**	**Base**	**KTN**	**Gain%**
R-GCN	**NDCG**	0.664	0.426	0.530	**24.36**	0.660	0.599	0.744	**24.26**
	std	0.003	0.006	0.002		0.001	0.008	0.004	
	MRR	0.683	0.325	0.514	**58.39**	0.656	0.524	0.785	**49.87**
	std	0.003	0.008	0.004		0.011	0.009	0.005	
HAN	**NDCG**	0.618	0.153	0.510	**232.35**	0.515	0.546	0.689	**26.21**
	std	0.005	0.007	0.003		0.008	0.003	0.005	
	MRR	0.634	0.010	0.516	**4958.82**	0.508	0.511	0.758	**48.28**
	std	0.002	0.005	0.002		0.001	0.008	0.007	
HGT	**NDCG**	0.615	0.234	0.536	**128.95**	0.694	0.367	0.735	**100.22**
	std	0.002	0.005	0.002		0.006	0.007	0.009	
	MRR	0.638	0.095	0.537	**464.88**	0.699	0.267	0.772	**189.21**
	std	0.006	0.002	0.005		0.002	0.005	0.012	
MAGNN	**NDCG**	0.536	0.513	0.513	**0.00**	0.684	0.676	0.692	**2.37**
	std	0.005	0.001	0.001		0.001	0.002	0.001	
	MRR	0.586	0.522	0.522	**0.00**	0.686	0.751	0.752	**0.13**
	std	0.004	0.001	0.002		0.002	0.001	0.004	
MPNN	**NDCG**	0.578	0.380	0.532	**40.03**	0.639	0.578	0.794	**37.19**
	std	0.008	0.008	0.004		0.007	0.007	0.005	
	MRR	0.603	0.253	0.505	**100.12**	0.652	0.584	0.847	**44.96**
	std	0.001	0.003	0.007		0.006	0.001	0.006	
H-MPNN	**NDCG**	0.670	0.283	0.584	**106.50**	0.676	0.459	0.671	**46.22**
	std	0.002	0.002	0.006		0.005	0.004	0.003	
	MRR	0.689	0.133	0.586	**339.76**	0.677	0.364	0.698	**91.92**
	std	0.003	0.003	0.005		0.01	0.005	0.002	

Table 10: **Effect of λ**

	P-A (L1)				A-V (L1)			
Metric	**NDCG**		**MRR**		**NDCG**		**MRR**	
λ	**Source**	**Target**	**Source**	**Target**	**Source**	**Target**	**Source**	**Target**
10^{-5}	0.780	0.587	0.772	0.595	0.689	0.626	0.690	0.642
10^{-3}	0.788	0.58	0.779	0.576	0.687	0.654	0.689	0.677
10^{0}	0.792	0.621	0.788	0.633	0.689	0.670	0.692	0.696
10^{2}	0.750	0.617	0.757	0.623	0.654	0.644	0.659	0.668
10^{4}	0.143	0.177	0.007	0.031	0.411	0.432	0.373	0.421

Table 11: **Statistics of Open Academic Graph**

Domain	**#papers**	**#authors**	**#fields**	**#venues**	**#institues**	
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 12: **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

different types are allowed to have different dimensions, as we generate different mixture-model cluster centers for each cluster *within each type*. Cross-type feature homophily is not necessary, since HGNN-KTN learns a transformation function between the type feature spaces.

A.7.1 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 2. We generate the graph with two node types and four edge types as described in Figure 1(a), then we divide each node type into 4 classes of 400 nodes. To generate an easy-to-transfer scenario, signal-to-noise ratio σ_f between means of feature distributions are all set to 10. The ratio σ_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.

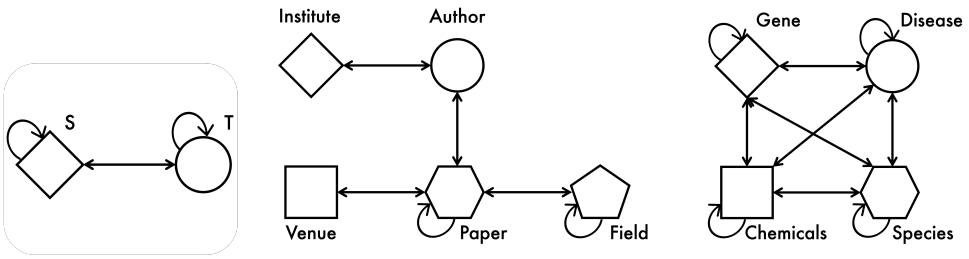
A.7.2 Sensitivity test in Section 6.5

Figure 5(a) shows the structures of graphs we used in Section 6.5. 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 σ_e and σ_f are described in Section 6.5. For each unique value of σ across the six (σ, r) pairs, we generate 5 heterogeneous graphs.

A.8 Real-world Dataset

Open Academic Graph (OAG) [28, 31, 44] is the largest publicly available heterogeneous graph. It is composed of five types of nodes: papers, authors, institutions, venues, 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.



(a) Synthetic graph (b) OAG (c) PubMed

Figure 5: Schema of synthetic and real-world heterogeneous graphs

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 11, 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 5(b).

For paper nodes, features are generated from each paper's title using a pre-trained XLNet [36]. 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 [5].

PubMed [39] 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 12, in which G, D, C, and S denote genes, diseases, chemicals, and species node types. The graph structure is described in Figure 5(c).

For gene and chemical nodes, features of dimension 200 are generated from related PubMed papers using word2vec [23]. For diseases and species nodes, features of dimension 50 are generated based on their graph structures using TransE [4].

A.9 Baselines

Zero-shot domain adaptation can be categorized into three groups — MMD-based methods, adversarial methods, and optimal-transport-based methods. MMD-based methods [18, 29, 20] minimize the maximum mean discrepancy (MMD) [11] between the mean embeddings of two distributions in reproducing kernel Hilbert space. DAN [18] enhances the feature transferability by minimizing multi-kernel MMD in several task-specific layers. JAN [20] aligns the joint distributions of multiple domain-specific layers based on a joint maximum mean discrepancy (JMMD) criterion.

Adversarial methods [9, 19] are motivated by theory in [3, 2] 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. DANN [9] learns domain invariant features by a min-max game between the domain classifier and the feature extractor. CDAN [19] exploits discriminative information conveyed in the classifier predictions to assist adversarial adaptation. CDAN-E [19] extends CDAN to condition the domain discriminator on the uncertainty of classifier predictions, prioritizing the discriminator on easy-to-transfer examples.

Optimal transport-based methods [27] estimate the empirical Wasserstein distance [25] between two domains and minimizes the distance in an adversarial manner Optimal transport-based method are based on a theoretical analysis [25] that Wasserstein distance can guarantee generalization for domain adaptation. WDGRL [27] estimates the empirical Wasserstein distance between two domains and minimizes the distance in an adversarial manner.

A.10 HGNN models

We briefly describe 6 heterogeneous graph neural networks (HGNN) models we used in the experiments. MPNN (message passing neural networks) [10] is originally designed for homogeneous graphs. We extend MPNN to process heterogeneous graphs by adding projection matrices that project input attributes of different node types into the same feature space before running the original MPNN. R-GCN [26] extends MPNN by specializing message matrices in each edge type, while HMPNN specializes all transformation and message matrices in each node/edge type in MPNN. HGT [15] extends HMPNN by adding attention modules. The attention modules have node-type-specific key/query projection matrices and edge-type-specific key-query similarity matrices, following the transformer architecture.

HAN [35] is a meta-path-based model who specializes parameters in each meta-path. HAN exploits meta-path-specific attention modules to aggregate features of neighboring nodes connected by each meta-path. Then HAN aggregates embeddings of different meta-paths with semantic-level attention modules. MAGNN [8] is another meta-path-based HGNN model. MAGNN aggregates features of all nearby nodes sitting on each meta-path using intra-meta-path attention modules. Then MAGNN aggregates features of different meta-paths using inter-meta-path attention modules.

All HGNN models we describe above have layer-wise parameters. As all HGNN models have parameters specialized in either node/edge/meta-path types, they all have distinct feature extractors for each node types, thus, they will suffer from the under-trained target node phenomena we showed in Section 4.2. Also, because the core intuition in KTN — namely that embeddings of any node types at the last layer are computed using the same set of the previous layer's intermediate embeddings (see Section 4.3) — holds across all HGNN models, KTN can be applied to any HGNN models and show greatly increased target-type accuracy.

A.11 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 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 KTN and other baselines. Learning rate is set to 5×10^{-5}.

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

We implement LP, EP and KTN using Pytorch. For the domain adaptation baselines (DAN, JAN, DANN, CDAN, CDAN-E, and WDGRL), we use a public domain adaptation library ADA [3]. We match the numbers of layers and dimensions of hidden embeddings across all HGNN models. We implement MPNN and HMPNN using Pytorch. For other HGNN models (R-GCN, HAN, HGT, and MAGNN), we use an open-source toolkit for Heterogeneous Graph Neural Network (OpenHGNN) [4]. Our code is publicly available [5].

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

[4] https://github.com/BUPT-GAMMA/OpenHGNN

[5] https://github.com/minjiyoon/KTN