NetTrans: Neural Cross-Network Transformation

Si Zhang[*], Hanghang Tong[*], Yinglong Xia[†], Liang Xiong[†], and Jiejun Xu[‡]
[*] University of Illinois at Urbana-Champaign, {sizhang2, htong}@illinois.edu;
[†] Facebook, {yxia, lxiong}@fb.com;
[‡] HRL Laboratories, jxu@hrl.com

ABSTRACT

Finding node associations across different networks is the cornerstone behind a wealth of high-impact data mining applications. Traditional approaches are often, explicitly or implicitly, built upon the linearity and/or consistency assumptions. On the other hand, the recent network embedding based methods promise a natural way to handle the non-linearity, yet they could suffer from the *disparate* node embedding space of different networks. In this paper, we address these limitations and tackle cross-network node associations from a new angle, i.e., cross-network transformation. We ask a generic question: *Given two different networks, how can we transform one network to another?* We propose an end-to-end model that learns a composition of nonlinear operations so that one network can be transformed to another in a hierarchical manner. The proposed model bears three distinctive advantages. First (*composite transformation*), it goes beyond the linearity/consistency assumptions and performs the cross-network transformation through a composition of nonlinear computations. Second (*representation power*), it can learn the transformation of both network structures and node attributes at different resolutions while identifying the cross-network node associations. Third (*generality*), it can be applied to various tasks, including network alignment, recommendation, cross-layer dependency inference. Extensive experiments on different tasks validate and verify the effectiveness of the proposed model.

KEYWORDS

Network transformation; node associations; network representation learning; network alignment; social recommendation

ACM Reference Format:
Si Zhang[*], Hanghang Tong[*], Yinglong Xia[†], Liang Xiong[†], and Jiejun Xu[‡]. 2020. NetTrans: Neural Cross-Network Transformation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403141

1 INTRODUCTION

In the era of big data, networks are often multi-sourced. Finding the node associations across different networks is a key stepping stone to explore deep insights from such multi-sourced networks. If nodes in different networks represent the same type of entities, finding the cross-network node associations is essentially the *(soft) network alignment* problem [38]. For example, aligning suspects across different transaction networks helps integrate the transaction histories of the suspects at different financial institutes, which in turn facilitates to uncover the complex financial fraud schema. On the other hand, if nodes in different networks represent different types of entities, finding the cross-network node associations is often referred to as the *cross-layer dependency inference* problem [1]. It indicates how entities of different types from different networks interact with each other. For instance, the user-product interactions across a social network and a product similarity network can be used for social recommendations [34]. In a biological system, the protein-protein interaction (PPI) networks are often coupled with the disease networks, and the cross-network node associations may indicate how diseases are related to different genes [22].

Traditional methods for cross-network node associations are often, explicitly or implicitly, built upon the linearity and/or consistency assumptions. For example, classic graph matching based network alignment methods often assume networks are noisy permutations of each other and minimize $\|\mathbf{B}_0 - \mathbf{P}\mathbf{A}_0\mathbf{P}^T\|_F^2$ where $\mathbf{A}_0, \mathbf{B}_0$ are the adjacency matrices of two networks and $\mathbf{P}$ is the permutation matrix [17, 37]. This formulation together with many of its variants (e.g., [38]) implicitly embraces a *linear operation*[1]. As for cross-layer dependency inference, a typical approach is based on network-regularized matrix factorization [1, 16, 32] under the *consistency/homophily* assumption. For example, in social recommendation, these methods typically assume similar users tend to share similar latent representations.

More recent efforts aim to approach the cross-network node association problem by learning node embedding vectors of different networks [3, 20]. These methods can potentially go beyond the linearity and/or the consistency assumptions behind the complex cross-network node associations by learning node embedding vectors through nonlinear functions. However, the node embedding vectors of different networks often lie in the *disparate vector spaces* which might be incomparable with each other. For instance, if we shift, rotate or scale the node embeddings of one network, it could significantly impair network alignment results [5].

In this paper, we address the above limitations and tackle cross-network node associations from a new angle, i.e., cross-network transformation. We ask a generic question: *Given two different networks, how can we transform one network to another?* We propose an end-to-end model *NetTrans* that bears three key advantages. First

[1]To see this, the objective function of graph matching based network alignment is equivalent to minimizing $\|\operatorname{vec}(\mathbf{B}_0) - \tilde{\mathbf{P}}\operatorname{vec}(\mathbf{A}_0)\|_2^2$ where $\operatorname{vec}(\mathbf{A}_0)$, $\operatorname{vec}(\mathbf{B}_0)$ denote the vectors of node pairs and $\tilde{\mathbf{P}} = \mathbf{P} \otimes \mathbf{P}$ is the Kronecker product of the permutation matrix. $\tilde{\mathbf{P}}$ is used as a *single* linear transformation across the node pairs of two networks.

(*composite transformation*), instead of learning a single linear transformation function underpinning graph matching based methods, the proposed model learns a composition of nonlinear functions to transform both network structures and node attributes, and in the meanwhile unveils the cross-network associations. Second (*representation power*), by exploiting the multi-resolution characteristics underlying the networks, the proposed pooling layer *TransPool* can learn the hierarchical representations of the networks and the unpooling layer *TransUnPool* learns the transformations at different resolutions. Third (*generality*), the proposed model is generic and it can be easily applied to numerous tasks, such as network alignment, social recommendation, cross-layer dependence inference, etc. The main contributions of this paper can be summarized as follows:

- **Problem Definition.** To our best knowledge, we are the first to address the cross-network transformation problem.
- **End-to-End Model.** We propose an end-to-end model *NetTrans* which composes of the novel pooling and unpooling operations to learn the transformation functions and find the node associations across different networks.
- **Experimental Results.** We perform extensive experiments in network alignment and social recommendation, which demonstrate the effectiveness of the proposed model to find the cross-network node associations.

The rest of the paper is organized as follows. Section 2 defines the cross-network transformation problem. Section 3 presents the proposed *NetTrans* model. Section 4 shows the experimental results. Related works and conclusion are given in Section 5 and Section 6.

2 PROBLEM DEFINITION

Table 1 summarizes the main symbols and notations used throughout the paper. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{a}$), and lowercase letters (e.g., α) for scalars. We use $\mathbf{A}(i, j)$ to denote the entry at the intersection of the i-th row and j-th column of the matrix $\mathbf{A}$, $\mathbf{A}(i, :)$ to denote the i-th row of $\mathbf{A}$ and $\mathbf{A}(:, j)$ to denote the j-th column of $\mathbf{A}$. We denote the transpose of a matrix by a superscript T (e.g., $\mathbf{A}^T$ is the transpose of $\mathbf{A}$). Furthermore, we use subscripts to index the matrices in different layers. For example, we use $\mathbf{A}_0$ to denote the input adjacency matrix and $\mathbf{A}_l$ as the adjacency matrix of the coarsened network in the l-th layer. In addition, we use u, v to index the nodes of two input networks and use u', v' to index the supernodes of the coarsened networks in each layer. Note that the supernode-u' of the output coarsened network obtained in the $(l-1)$-th layer is equivalent to the node-u' of the input network in the l-th layer. In this paper, we use 'graphs' and 'networks' interchangeably.

Multi-sourced networks are often associated with each other, in terms of the network structures, node attributes and cross-network node associations. In other words, there exist some functions that link the multi-sourced networks together. In this paper, we consider to learn a transformation function denoted by $g(\cdot)$ such that the source network $\mathcal{G}_1$ can be transformed to the target network $\mathcal{G}_2$, i.e., $g(\mathcal{G}_1) \simeq \mathcal{G}_2$. Figure 1 (a) presents an illustrative example in the network alignment scenario. As we can see, the source network $\mathcal{G}_1$ and target network $\mathcal{G}_2$ have different network structures as well as node attributes, and more importantly, the cross-network node associations (i.e., node correspondences) are unknown. Our goal is to

Table 1: Symbols and Notations

Symbols	Definition
$\mathcal{G}_1, \mathcal{G}_2$	input source and target networks
$\mathcal{V}_1, \mathcal{V}_2$	the sets of nodes of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{A}_0, \mathbf{B}_0$	input adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
$\mathbf{X}_0, \mathbf{Y}_0$	input node attribute matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$
n_0, m_0	# of nodes in $\mathbf{A}_0$ and $\mathbf{B}_0$
$\mathbf{A}_l, \mathbf{B}_l$	adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$ in the l-th layer
$\mathbf{X}_l, \mathbf{Y}_l$	node feature matrices in the l-th layer
$\mathbf{P}_l$	node-supernode assignment matrix in the l-th layer
n_l, m_l	# of nodes in $\mathbf{A}_l$ and $\mathbf{B}_l$ in the l-th layer
L	# of layers of both encoder and decoder
α, β, γ	parameters controlling the importance of loss terms
$[\cdot\Vert\cdot]$	concatenation operator of two vectors

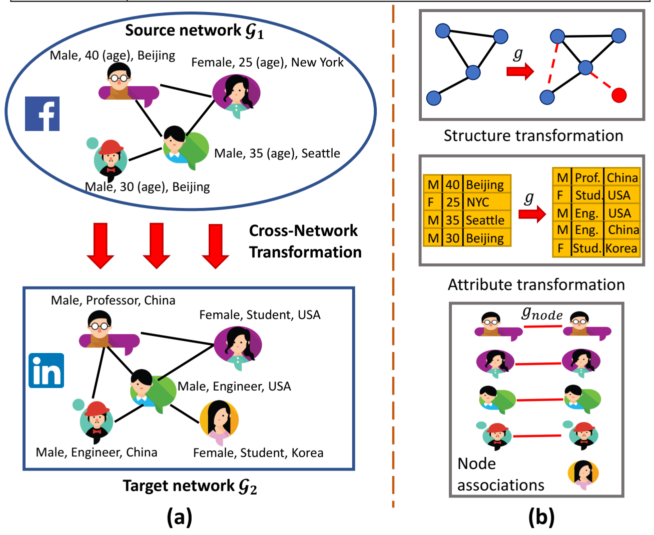


Figure 1: An illustrative example of cross-network transformation in the network alignment scenario. Figure 1 (a) shows the entire transformation across input networks. Figure 1 (b) shows the transformation on network structure, node attributes and cross-network node associations.

learn the transformation functions on both network structures and node attributes while identifying the cross-network associations.

To be specific, we use the following major notations to describe the cross-network transformation. First, we denote the input source network $\mathcal{G}_1$ and target network $\mathcal{G}_2$ by triplets, i.e., $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_0, \mathbf{X}_0\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{B}_0, \mathbf{Y}_0\}$[2] where $\mathcal{V}_1, \mathbf{A}_0, \mathbf{X}_0$ denote the set of nodes, adjacency matrix and node attributes of $\mathcal{G}_1$, respectively and similarly for $\mathcal{G}_2$. Second, we denote the transformation function on network structures and node attributes by g such that $(\mathbf{B}_0, \mathbf{Y}_0) \simeq g(\mathbf{A}_0, \mathbf{X}_0)$. Lastly, the transformation function induces the node associations across different networks and is denoted by g_{node}. Figure 1 (b) shows an example of the corresponding transformation in terms of network structure and node attributes, as well as the node associations. Given the above notations, we formally define the cross-network transformation problem as follows.

PROBLEM 1. *CROSS-NETWORK TRANSFORMATION.*

Given: *(1) input source network $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_0, \mathbf{X}_0\}$ and target network $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{B}_0, \mathbf{Y}_0\}$ where $\mathcal{V}_1, \mathcal{V}_2$ denote the nodes of $\mathcal{G}_1, \mathcal{G}_2$,*

[2]If the attributes are not available, one can simply set $\mathbf{X}_0, \mathbf{Y}_0$ as identity matrices or manually extract structure-dependent attributes.

$\mathbf{A}_0, \mathbf{B}_0$ *are the adjacency matrices and* $\mathbf{X}_0, \mathbf{Y}_0$ *are the node attribute matrices of* $\mathcal{G}_1, \mathcal{G}_2$*, and (2) partial knowledge of the cross-network node associations* $\mathbf{L}$ *where* $\mathbf{L}(u, v)$ *indicates whether node-u in* $\mathcal{G}_1$ *associates with node-v in* $\mathcal{G}_2$ *a priori.*

Output: *(1) the cross-network transformation function* $g = \mathcal{F} \circ \mathcal{H}$ *where* $\mathcal{F}$ *denotes a set of encoding functions and* $\mathcal{H}$ *denotes the decoding functions such that* $g(\mathcal{G}_1) \simeq \mathcal{G}_2$*, and (2) the cross-network association function* g_{node} *where* $g_{node}(u, v)$ *measures to what extent node-u in* $\mathcal{G}_1$ *is associated with node-v in* $\mathcal{G}_2$*.*

Let us take the graph matching based methods as an example to further illustrate the functions g and g_{node}, given two networks $\mathcal{G}_1 = \{\mathcal{V}_1, \mathbf{A}_0, \mathbf{X}_0\}$ and $\mathcal{G}_2 = \{\mathcal{V}_2, \mathbf{B}_0, \mathbf{Y}_0\}$ with the same type of nodes, graph matching based methods aim to learn a permutation matrix $\mathbf{P}$, indicating the node correspondence between $\mathcal{V}_1, \mathcal{V}_2$ (i.e., g_{node}). In the meanwhile, the matrix $\mathbf{P}$ is also used as a single transformation function such that $\text{vec}(\mathbf{B}_0) \simeq \tilde{\mathbf{P}}\,\text{vec}(\mathbf{A}_0)$ and $\mathbf{Y}_0 \simeq \mathbf{P}\mathbf{X}_0$ where $\tilde{\mathbf{P}} = \mathbf{P} \otimes \mathbf{P}$. Thus, the transformation function g can be written as $g(\text{vec}(\mathbf{A}_0), \mathbf{X}_0) = (\tilde{\mathbf{P}}\,\text{vec}(\mathbf{A}_0), \mathbf{P}\mathbf{X}_0)$. And the cross-network node association function $g_{\text{node}}(u, v) = \mathbf{P}(v, u)$. Despite its elegant mathematical formulation, such a single linear transformation might over-simplify the complex associations across networks.

Remarks. Note that the cross-network transformation problem is similar to but bears subtle differences from the graph-to-graph translation problem [10, 13]. For the latter, it implicitly assumes that nodes in different networks are of the same type and all node correspondences are perfectly *known a priori*. In contrast, the cross-network transformation problem that we study in this paper aims to *learn* such node associations from the networks (i.e., the function g_{node}), in addition to learning the transformation function g.

We envision that the learned transformation functions from Problem 1 can be applied in a variety of data mining tasks. In this paper, we focus on two such tasks, including (Task 1) network alignment and (Task 2) social recommendation. In *Task 1*, we consider two networks with the same type of nodes and the transformation function g_{node} measures to what extent that two nodes are aligned with each other. In *Task 2*, we consider a social network and a product similarity network, and the transformation function g_{node} predicts whether a user likes/buys a product.

3 THE NETTRANS MODEL

In this section, we present the proposed model *NetTrans*, an end-to-end semi-supervised model to solve Problem 1. We start by giving an overview of our model, and then detail the components of the model, followed by the discussions on the potential generalizations.

3.1 NetTrans Model Overview

The core challenge of cross-network transformation lies in how to design a model that can jointly learn the transformation function g and the cross-network node associations g_{node}. In this paper, we propose an encoder-decoder architecture that decomposes the transformation function g into two parts, including the encoder $\mathcal{F}$ and the decoder $\mathcal{H}$. We exploit the multi-resolution characteristics of real-world networks in both the encoder and the decoder. The overall architecture of the proposed *NetTrans* model is shown in Figure 2. The encoder $\mathcal{F}$ aims to coarsen the source network and learn the network structure and node representations at different resolutions. On the other hand, the decoder $\mathcal{H}$ reconstructs the structure and node representations of the target network at different resolutions. To make the source network and target network at different resolutions comparable to each other, we design the encoder and decoder to have the same number of layers (i.e., L). To learn the cross-network node associations g_{node}, we need to simultaneously learn the node assignments across two adjacent resolutions indicating which nodes at the finer resolution are merged into which node(s) in the next coarser resolution. To this end, we propose a pooling layer *TransPool* as the core component of the encoder and an unpooling layer *TransUnPool* in the decoder.

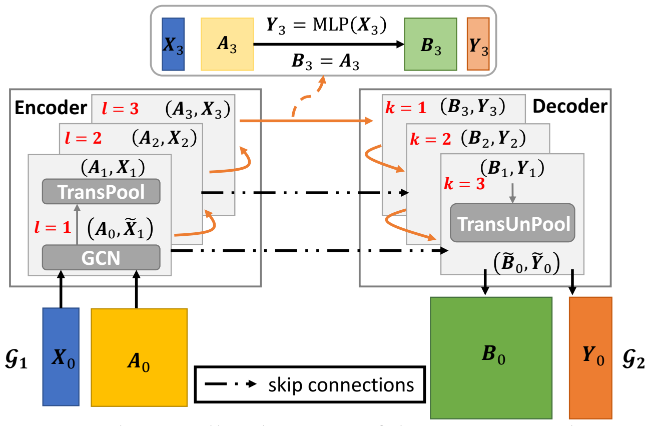


Figure 2: The overall architecture of the cross-network transformation model *NetTrans* ($L = 3$).

The intuition behind such a design is that we could simplify the cross-network transformation at the coarser resolutions, since the coarsened networks are likely to become more similar with each other. For instance, given a social network and a product similarity network, the nodes at the coarse resolutions might share similar latent meanings (e.g., a group of users who like to buy computers vs. a group of computer-related products). Moreover, the association between a group of users and a group of similar products will provide critical auxiliary information to infer the associations between the users and products in these groups. Finally, with this hierarchical learning, the proposed *TransUnPool* layers naturally learns the functions to model how network structures and node representations are transformed at different resolutions.

3.2 NetTrans Encoder

Denote $\mathcal{F} = \{f_l\}$ as the functions in the encoder where f_l, $l = 1, \cdots, L$ represents the encoding function in the l-th encoder layer. In the l-th encoder layer, the function $f_l(\cdot, \cdot)$ on the network can be decomposed into learning the adjacency matrix and node attributes of the coarsened network. Denote $\mathbf{A}_l \in \mathbb{R}^{n_l \times n_l}$ and $\mathbf{X}_l \in \mathbb{R}^{n_l \times d_l}$ as the output adjacency matrix and node representations of the coarsened network in the l-th layer. Given the inputs $\mathbf{A}_{l-1} \in \mathbb{R}^{n_{l-1} \times n_{l-1}}$ ($n_l \leq n_{l-1}$) and $\mathbf{X}_{l-1} \in \mathbb{R}^{n_{l-1} \times d_{l-1}}$ which is the output coarsened network in the $(l-1)$-th layer, we can denote the encoding function by $(\mathbf{A}_l, \mathbf{X}_l) = f_l(\mathbf{A}_{l-1}, \mathbf{X}_{l-1})$. For example, the outputs of the first encoder layer can be computed by $(\mathbf{A}_1, \mathbf{X}_1) = f_1(\mathbf{A}_0, \mathbf{X}_0)$.

To learn the structure of the coarsened networks, one prevalent choice is to coarsen the network with an assignment matrix, e.g., $\mathbf{A}_l = \mathbf{P}_l \mathbf{A}_{l-1} \mathbf{P}_l^T$ where $\mathbf{P}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{P}_{l-1}(u', u)$ measures the strength of node-u in $\mathbf{A}_{l-1}$ being merged into the supernode-u'. Existing methods to compute $\mathbf{P}_l$ include the classic methods that

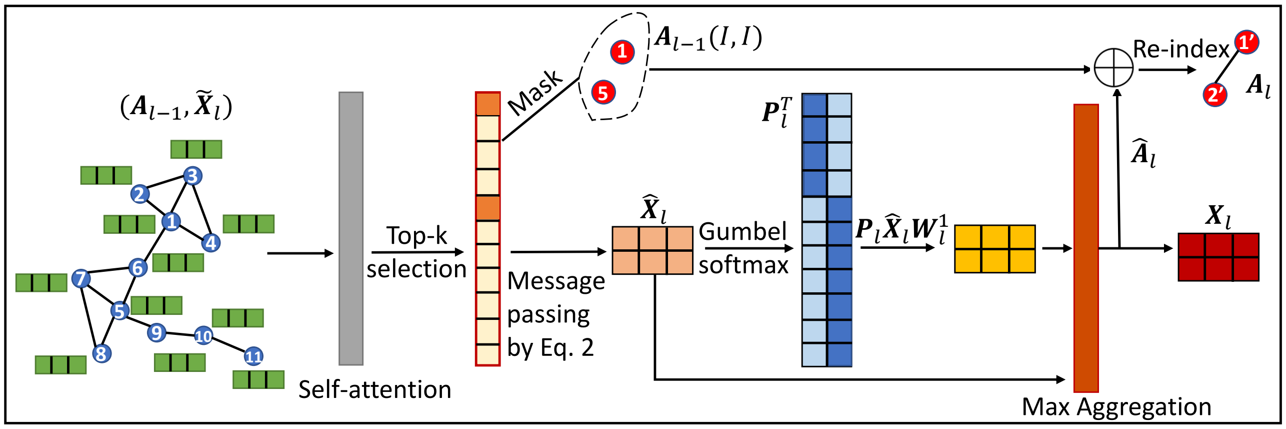


Figure 3: Description of the pooling layer (TransPool) in the l-th encoder layer.

calculate it deterministically (e.g., [26]) and graph neural networks based methods [35]. One advantage is that all nodes are assigned to certain supernodes based on the dense assignment matrix. However, these methods might lead to costly computations and the densely connected network structure [35]. We also remark in Proposition 1 that this coarsening process is built upon linear operations which might insufficiently capture the underlying hierarchical structures.

Proposition 1. *Given an assignment matrix* $\mathbf{P}_l$*, the coarsening process* $\mathbf{A}_l = \mathbf{P}_l\mathbf{A}_{l-1}\mathbf{P}_l^T$ *constructs edges by linear operations.*

Proof. See Appendix. □

Another way to coarsen networks is the learnable importance-based pooling operations [9, 19]. These methods basically select the top-k important nodes as *supernodes*, and preserve the original connections among the selected nodes as the edges among the corresponding supernodes. The advantages of these methods include the efficient computations and the sparse structure of the coarsened networks. Note that keeping the original connections can be viewed as a special case of assignment matrix based methods. Specifically, the assignment matrix $\mathbf{P}_l(u', u) = 1$ if and only if node-u is selected and re-indexed to supernode-u'. Thus, according to Proposition 2, the informativeness of the coarsened structure by these methods is also hindered by the underlying linear operations. In addition, it is unknown how those unselected nodes are assigned to the supernodes (i.e., $\mathbf{P}_l(:, u) = 0$ for all u that are not selected). To this end, we propose a new graph pooling operation *TransPool* (shown in Figure 3) that can balance between the above two types of pooling strategies and learn the assignments for all nodes.

Following [9, 19], in the l-th encoder layer, we first feed the inputs $(\mathbf{A}_{l-1}, \mathbf{X}_{l-1})$ to a graph convolutional layer (e.g., [15]) before the pooling layer. By denoting $\tilde{\mathbf{X}}_l = \text{GCN}(\mathbf{A}_{l-1}, \mathbf{X}_{l-1})$, we compute the self-attention scores $\mathbf{z}_l \in \mathbb{R}^{n_{l-1}}$ by a graph convolutional layer [15] to measure the node importance [19]. This is formulated by

$$\mathbf{z}_l = \sigma\left(\tilde{\mathbf{D}}_{l-1}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{l-1}\tilde{\mathbf{D}}_{l-1}^{-\frac{1}{2}}\tilde{\mathbf{X}}_l\mathbf{W}_l^{\text{self}}\right) \quad (1)$$

where $\sigma(\cdot)$ denotes the nonlinear activation function (e.g., *tanh*), $\tilde{\mathbf{A}}_{l-1} = \mathbf{A}_{l-1} + \mathbf{I}$, $\mathbf{A}_{l-1} \in \mathbb{R}^{n_{l-1}\times n_{l-1}}$ is the input adjacency matrix in the l-th encoder layer, $\tilde{\mathbf{D}}_{l-1} = \text{diag}(\tilde{\mathbf{A}}_{l-1}\mathbf{1})$ is the diagonal degree matrix of $\tilde{\mathbf{A}}_{l-1}$, $\tilde{\mathbf{X}}_l \in \mathbb{R}^{n_{l-1}\times d_l}$ is the input node feature matrix, and $\mathbf{W}_l^{\text{self}} \in \mathbb{R}^{d_l}$ contains the parameters to compute the self-attention scores. By using these self-attention scores to measure node importance, both network structure and node features are naturally encoded. Thus, nodes of the top-n_l scores are likely to be more important to capture the structural and feature information. In [9, 19], these selected top-n_l nodes are then used as masks to construct the adjacency matrix and node features of the coarsened network. Specifically, by denoting the indices of the selected nodes as $\mathcal{I} = \text{top-rank}(\mathbf{z}_l, n_l)$, [9, 19] compute $\mathbf{A}_l = \mathbf{A}_{l-1}(\mathcal{I}, \mathcal{I}) = \mathbf{P}_l\mathbf{A}_{l-1}\mathbf{P}_l^T$ where $\mathcal{I}_{u'}$ is the u'-th element of $\mathcal{I}$ and $\mathbf{P}_l(u', \mathcal{I}_{u'}) = 1$, $\forall u' = 1, \cdots, n_l$ are the only nonzero elements in $\mathbf{P}_l$. The coarsened node features are computed by $\mathbf{X}_l(u', :) = \tilde{\mathbf{X}}_l(\mathcal{I}_{u'}, :) \odot \mathbf{z}_l(u')$ where $\odot$ is element-wise product.

However, this simple masking-based pooling operation has two potential limitations. First, the computation of output feature matrix $\mathbf{X}_l$ in [9, 19] insufficiently leverages the representations of the unselected nodes by simply re-scaling $\tilde{\mathbf{X}}_l$ based on $\mathbf{z}_l$. Second, as mentioned before, edges in the coarsened adjacency matrix are constructed by linear operations. Besides, the coarsened network empirically may contain isolated nodes. For example, an isolated node-u' can occur when $\mathbf{A}_{l-1}(\mathcal{I}_{u'}, \mathcal{I} \setminus \mathcal{I}_{u'}) = 0$ for some u' (e.g., node-1 and node-5 in red in Figure 3). Note that the isolated nodes cannot be completely avoided by Eq. (1). Such an issue could further lead to the inability of the information propagation to the isolated nodes in the next encoder layer, making the network structure and node representation learning at the coarser resolutions even worse.

To address the first issue, instead of directly rescaling the representation vectors, we allow message passing from n_{l-1} nodes to the selected n_l supernodes. In particular, we use attention-based message passing [30] formulated as below.

$$\hat{\mathbf{X}}_l(u', :) = \sigma\left(\tilde{\mathbf{X}}_l(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 + \sum_{u\in\mathcal{N}_{u'}} \alpha_{u'u}\tilde{\mathbf{X}}_l(u, :)\mathbf{W}_l^1\right) \quad (2)$$

$$\alpha_{u'u} = \frac{\exp\left(\mathbf{a}_l^T\left[\tilde{\mathbf{X}}_l(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 \| \tilde{\mathbf{X}}_l(u, :)\mathbf{W}_l^1\right]\right)}{\sum_{u_1\in\mathcal{N}_{u'}}\exp\left(\mathbf{a}_l^T\left[\tilde{\mathbf{X}}_l(\mathcal{I}_{u'}, :)\mathbf{W}_l^1 \| \tilde{\mathbf{X}}_l(u_1, :)\mathbf{W}_l^1\right]\right)} \quad (3)$$

where $\mathcal{N}_{u'}$ denotes the 1-hop neighborhood of supernode-u' in the bipartite graph $\mathcal{G}_b$ formed by $\mathbf{A}_{l-1}(:, \mathcal{I})$, $\mathbf{a}_l \in \mathbb{R}^{2d_l}$ and $\mathbf{W}_l^1 \in \mathbb{R}^{d_l\times d_l}$ are the parameters to be learned. However, Eq. (2) cannot aggregate the features from the nodes that are multi-hop away from the selected nodes (e.g., from node-10 to node-5). As a remedy, we propose to additionally aggregate the node features based on the assignment matrix $\mathbf{P}_l$. To efficiently learn $\mathbf{P}_l$, we propose the following mechanism to select supernode candidates. First, for a node-u that have some supernodes as their 1-hop neighbors in $\mathcal{G}_b$, they can be assigned only to their 1-hop neighboring supernodes, i.e., $C(u) = \{u' | \mathbf{A}_{l-1}(u, \mathcal{I}_{u'}) = 1, u' = 1, \cdots, n_l\}$. For example, in Figure 3, node-6 can be assigned to either node-1 or node-5. Second, for some node-u that connects to supernodes exactly in 2 hops (e.g., node-10 in Figure 3), we select the candidates of supernodes by $C(u) = \{u' | \mathbf{A}_{l-1}(u, :)\mathbf{A}_{l-1}(:, \mathcal{I}_{u'}) = 1, \mathbf{A}_{l-1}(u, \mathcal{I}_{u'}) = 0, u' = 1, \cdots, n_l\}$. For the rest of nodes (e.g., node-11), we simply set $C(u) = \{u' | u' = 1, \cdots, n_l\}$. In addition, a hard assignment matrix $\mathbf{P}_l$ often requires each column of $\mathbf{P}_l$ to be a one-hot vector, i.e., $\mathbf{P}_l(u', u) = 1$ if and only if node u is merged into supernode u'. However, it is very difficult to directly learn those one-hot vectors as they essentially involve discrete variables, making the computations non-differentiable. In our paper, we use the continuous Gumbel softmax [12] functions to approximate them which computes $\mathbf{P}_l$ by

$$\mathbf{P}_l(u', u) = \frac{\exp\left(\left[\log\left(\hat{\mathbf{X}}_l(u', :)\mathbf{W}_l^g \tilde{\mathbf{X}}_l^T\right) + g_{u'u}\right] / \tau\right)}{\sum_{c \in C(u)} \exp\left(\left[\log\left(\hat{\mathbf{X}}_l(c, :)\mathbf{W}_l^g \tilde{\mathbf{X}}_l^T\right) + g_{cu}\right] / \tau\right)}$$

where $u' \in C(u)$, $g_{u'u}$ is drawn from Gumbel(0, 1) distribution, $\mathbf{W}_l^g \in \mathbb{R}^{d_l \times d_l}$ is the parameter matrix and τ is the softmax temperature. According to the Gumbel softmax distribution [12], as $\tau \to +\infty$, $\mathbf{P}_l(:, u)$ becomes a uniform distribution. In contrast, as $\tau \to 0$, $\mathbf{P}_l(:, u)$ is close to a one-hot vector but the variance of the gradients is large. Thus, we learn the parameters by starting with a large temperature and annealing to a small temperature τ. We then aggregate the information from distant nodes into the supernodes by $\mathbf{P}_l \tilde{\mathbf{X}} \mathbf{W}_l^1$, which can aid the representation learning of supernodes to better summarize the local neighborhoods. The output representations of the supernodes are computed by

$$\mathbf{X}_l = \text{Aggregate}(\hat{\mathbf{X}}_l, \mathbf{P}_l \tilde{\mathbf{X}}_l \mathbf{W}_l^1) \quad (4)$$

where Aggregate$(\cdot, \cdot)$ is a layer-wise aggregation. In our paper, we use the max aggregation which simply takes the element-wise max.

To go beyond the linearity behind the coarsening process and address the issue of isolated supernodes, our key idea is to leverage the representations of supernodes to add auxiliary weighted edges. Specifically, we add weights to the existing edges among the supernodes in $\mathbf{A}_{l-1}(\mathcal{I}, \mathcal{I})$ to measure the edge strengths. For isolated supernodes, we compensate the edges by adding weighted edges only between isolated supernodes and the rest of supernodes. Note that the isolated supernodes $\mathcal{I}_l^s$ can be simply detected by whether there exist edges connecting to them in $\mathbf{A}_{l-1}(\mathcal{I}, \mathcal{I})$ [8]. Then, under the classic assumption that nodes with similar representations are likely to be connected, the auxiliary edges to be added are computed by a sigmoid function, i.e., $\mathbf{A}_l = \frac{1}{2}(\mathbf{A}_{l-1}(\mathcal{I}, \mathcal{I}) + \hat{\mathbf{A}}_l)$ where

$$\hat{\mathbf{A}}_l(u_1', u_2') = \begin{cases} 2\sigma_s\left(\mathbf{X}_l(u_1', :)\mathbf{X}_l(u_2', :)^T\right) & \text{if } u_1' \in \mathcal{I}_l^s \text{ or } u_2' \in \mathcal{I}_l^s \\ \sigma_s\left(\mathbf{X}_l(u_1', :)\mathbf{X}_l(u_2', :)^T\right) & \text{if } u_1' \notin \mathcal{I}_l^s \text{ and } u_2' \notin \mathcal{I}_l^s \\ 0 & \text{otherwise} \end{cases}$$

and $\sigma_s(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. In summary, we have the encoding function on adjacency matrices and node features as

$$(\mathbf{A}_l, \mathbf{X}_l) = f_l(\mathbf{A}_{l-1}, \mathbf{X}_{l-1}) = \text{TransPool}\left(\text{GCN}(\mathbf{A}_{l-1}, \mathbf{X}_{l-1})\right) \quad (5)$$

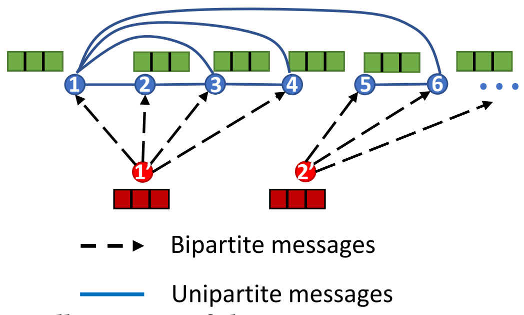


Figure 4: Illustrations of the message passing in TransUnPool layer corresponding to the pooling layer in Figure 3.

3.3 NetTrans Decoder

Our goal of the decoder is to learn node representations and the edges among the nodes in the context of the target network $\mathcal{G}_2$. We denote the decoder by a set of functions $\mathcal{H} = \{h_k\}$, $k = 1, \cdots, L$ where $h_k(\cdot, \cdot)$ represents the decoding function at the k-th decoder layer. At the k-th decoder layer, the decoding function takes $\mathbf{B}_{L-k+1}, \mathbf{Y}_{L-k+1}$ as inputs and outputs $\mathbf{B}_{L-k}, \mathbf{Y}_{L-k}$ as the adjacency matrix and node representations of the target network at the next finer resolution. Note that we have $\mathbf{Y}_{L-k+1} \in \mathbb{R}^{m_{L-k+1} \times d_{L-k+1}}$ and $\mathbf{Y}_{L-k} \in \mathbb{R}^{m_{L-k} \times d_{L-k}}$ where m_{L-k}, m_{L-k+1} ($m_{L-k} \geq m_{L-k+1}$) denote the numbers of nodes. Similar to the encoder, we denote the k-th decoder layer as $(\mathbf{B}_{L-1}, \mathbf{Y}_{L-1}) = h_k(\mathbf{B}_L, \mathbf{Y}_L)$.

Existing unpooling operator includes *gUnPool* [9] for a single network that restores the structure and node hidden representations of the input network $\mathcal{G}_1$ obtained in different encoder layers. However, it is restricted to a single network and cannot be applied to the cross-network scenario due to the following reasons. First (*node associations*), nodes in different networks can have different types. And even for the networks of the same node type, the cross-network node correspondences are unknown. Thus, without the knowledge of the cross-network node associations, it is inappropriate to use the node ordering of the source network as the reference of the target network. Second (*network structure*), networks from different sources might have different structural patterns. In this way, it might mislead learning the structures of the target network at different resolutions. Third (*node representations*), nodes in different networks, either of the same type or of different types, can carry different structural and attribute information.

In our paper, instead of *copying* the structure and node representations of source network, we design a novel unpooling layer (*TransUnPool*) to decode the target network at different resolutions. To address the first issue, since learning the cross-network node associations in different layers is nested together leading to a sophisticated learning process and costly computations, we simplify it by assuming the supernodes in the encoder layers represent the same set of latent entities as those in the corresponding decoder layers. For example, suppose a supernode-u' of the source social network in an encoder layer represents a group of users who like computers, then this supernode in its symmetric decoder layer may represent a set of products related to computers. That is, the supernode-u' in both encoder and decoder layers represents the same latent entity 'computer'. By doing this, the supernodes in the encoders and decoders are naturally one-to-one mapped and the assignment matrix $\mathbf{P}_l$ ($l \geq 2$) can be shared with the k-th ($k = L - l + 1$) decoder layer.

To address other issues, we hypothesize that two networks are close to each other at the coarsest resolution such that they share the same network structure and the node representations $\mathbf{Y}_L$ can be transformed from $\mathbf{X}_L$ via a multilayer perceptron (MLP), i.e.,

$$\mathbf{B}_L = \mathbf{A}_L,\ \mathbf{Y}_L = \text{MLP}_1(\mathbf{X}_L). \tag{6}$$

Then, given the input supernode representations $\mathbf{Y}_{L-k+1}$ ($k < L$) in the k-th decoder layer, to learn the structure and node representations of the target network at the corresponding resolution, we propose the following message passing module (shown in Figure 4) as a building block. Specifically, we define two types of messages that propagate to the nodes. The first type of messages are those that propagate from supernodes to nodes via the bipartite edges $\mathbf{P}_{L-k+1}$ (denoted by black dashed lines in Figure 4). Mathematically, these bipartite messages are formulated by

$$\mathbf{m}^k_{v'\to v} = \mathbf{P}_{L-k+1}(v', v) \odot (\mathbf{Y}_{L-k+1}(v', :)\mathbf{W}^2_k) \tag{7}$$

where $\mathbf{W}^2_k \in \mathbb{R}^{d_{L-k+1}\times d_{L-k}}$ is the parameter matrix and $\mathbf{P}_{L-k+1}(v', v)$ is used to weigh the importance of the message based on to what extent that the node-v is merged into supernode-v' in the $(L-k+1)$-th encoder layer. Another type of messages are passed among the nodes in the unipartite graph. Our intuition is that the structures and node representations of the coarsened source networks in the encoder layers can provide some initial information, based on which we aim to calibrate the structure and node representations to fit the target network $\mathcal{G}_2$. Specifically, we first transfer the adjacency matrix $\mathbf{A}_{L-k}$ and node representations $\mathbf{X}_{L-k}$ through the skip connections (denoted by the black dash dotted lines in Figure 2). Then, we define the messages along the edges in the unipartite graph by

$$\mathbf{m}^k_{v_1\to v} = \frac{1}{\sqrt{|\mathcal{N}_v|}\sqrt{|\mathcal{N}_{v_1}|}}\mathbf{X}_{L-k}(v_1, :)\mathbf{W}^3_k \tag{8}$$

where $\mathbf{W}^3_k \in \mathbb{R}^{d_{L-k+1}\times d_{L-k}}$ is the parameter matrix and $|\mathcal{N}_v|$ denotes the number of neighbors of node-v according to $\mathbf{A}_{L-k}$. In this way, the representations of nodes in the k-th decoder layer can be computed by combining both types of messages as

$$\mathbf{Y}_{L-k}(v, :) = \sum_{\substack{v',\ s.t.\\ \mathbf{P}_{L-k+1}(v',v)>0}} \mathbf{m}^k_{v'\to v} + \sum_{v_1\in\mathcal{N}_v} \mathbf{A}_{L-k}(v_1, v)\odot \mathbf{m}^k_{v_1\to v} \tag{9}$$

where $\mathbf{A}_{L-k}(v_1, v)$ denotes the weight of the edge (v_1, v).

To calibrate the network structure to learn the structural pattern of $\mathcal{G}_2$ at different resolutions, we use $\mathbf{Y}_{L-k}$ to compute to what extent we need to add/delete edges upon $\mathbf{A}_{L-k}$, written as

$$\mathbf{B}_{L-k}(v, v_1) = \frac{1}{2}\max\{0, \mathbf{A}_{L-k}(v, v_1) + \sigma_t(\mathbf{Y}_{L-k}(v, :)\mathbf{Y}_{L-k}(v_1, :)^T)\} \tag{10}$$

where $\sigma_t(x) \in (-1, 1)$ denotes the *tanh* activation function and we use a ReLU function to make $\mathbf{B}_{L-k}$ only contain nonnegative entries. However, Eq. (10) calculates $O(m^2_{L-k})$ number of values, which is computationally costly. To make it more efficient, we only compute $\sigma_t(\mathbf{Y}_{L-k}(v, :)\mathbf{Y}_{L-k}(v_1, :)^T)$ for the (v, v_1) such that $\mathbf{A}_{L-k}(v, v_1) \neq 0$, which in practice performs well in our experiments.

In the last decoder layer (i.e., $k = L$), we cannot directly use $\mathbf{P}_1$ as in Eq. (7) given the fact that nodes in $\mathcal{G}_1$ might either (1) have a different type from nodes in $\mathcal{G}_2$ or (2) have the same type but the correspondences to nodes in $\mathcal{G}_2$ are unknown. Fortunately, we have a partial knowledge of the cross-network node associations across $\mathcal{G}_1$ and $\mathcal{G}_2$ based on $\mathbf{L}(u, v)$ indicating whether node-u in $\mathcal{G}_1$ associates with node-v in $\mathcal{G}_2$ a priori. Note that $(\mathbf{P}_1\mathbf{L})(v', v) = \sum_{i=1}^{n_1}\mathbf{P}_1(v', u_i)\mathbf{L}(u_i, v)$ measures the strength of the assignment between node-v in $\mathcal{G}_2$ and the supernode-v' based on how many nodes in $\mathcal{G}_1$ that are associated with node-v in $\mathcal{G}_2$ a priori and also assigned to supernode-v'. In this way, we can use $\mathbf{Q}_1 = \mathbf{P}_1\mathbf{L}$ as the partially existing edges for bipartite message passing (i.e., dashed lines in Figure 4). In addition, we can construct bipartite messages at the last decoder layer from the nodes in $\mathcal{G}_1$ to nodes in $\mathcal{G}_2$ at the finest resolution through the prior knowledge $\mathbf{L}$ similarly as

$$\begin{aligned}\mathbf{m}^L_{v'\to v} &= \mathbf{Q}_1(v', v)\odot(\mathbf{Y}_1(v', :)\mathbf{W}^2_L)\\ \mathbf{m}^L_{v_1\to v} &= \frac{1}{\sqrt{|\mathcal{N}_v|}\sqrt{|\mathcal{N}_{v_1}|}}\mathbf{Y}_0(v_1, :)\mathbf{W}^3_L\\ \mathbf{m}^L_{u\to v} &= \mathbf{L}(u, v)\odot(\tilde{\mathbf{X}}_1(u, :)\mathbf{W}^4_L)\end{aligned} \tag{11}$$

where $\mathcal{N}_v$ denotes the neighborhood of node-v and the node-v itself in the original target network $\mathcal{G}_2$. The final node representations of $\mathcal{G}_2$ can be computed by aggregating the messages in Eq. (11) as

$$\tilde{\mathbf{Y}}_0(v, :) = \sum_{\substack{v',\ s.t.\\ \mathbf{Q}_1(v',v)>0}} \mathbf{m}^L_{v'\to v} + \sum_{v_1\in\mathcal{N}_v}\mathbf{B}_0(v_1, v)\odot\mathbf{m}^L_{v_1\to v} + \sum_{\substack{u,\ s.t.\\ \mathbf{L}(u,v)>0}}\mathbf{m}^L_{u\to v} \tag{12}$$

In summary, the k-th decoder layer can be computed by Eq. (9) and Eq. (10) (Eq. (12) in the L-th decoder layer) and

$$h_k = \text{TransUnPool}\,(\mathbf{Y}_{L-k+1}, \mathbf{A}_{L-k}, \mathbf{X}_{L-k}, \mathbf{P}_{L-k+1}) \tag{13}$$

3.4 NetTrans Model Training

With L encoder layers and L decoder layers, we can write the transformation function g of network structure and node attributes as

$$g = h_L \circ \cdots \circ h_1 \circ f_L \circ \cdots \circ f_1. \tag{14}$$

To learn the model parameters, our objectives are to reconstruct the target network $\mathcal{G}_2$ in terms of both structure and node attributes, while reflecting the observed cross-network node associations $\mathbf{L}$. To reconstruct the structure of $\mathcal{G}_2$, we minimize the binary cross-entropy loss over edges written as follows.

$$\mathcal{L}_{\text{adj}} = -\frac{1}{|\mathcal{E}|}\sum_{(v,v_1)\in\mathcal{E}}[y_{v,v_1}\log p_{v,v_1} + (1-y_{v,v_1})\log(1-p_{v,v_1})] \tag{15}$$

where $p_{v,v_1} = \sigma_s(\tilde{Y}_0(v, :)\tilde{Y}_0(v_1, :)^T)$, $\mathcal{E} = \mathcal{E}_2 \cup \bar{\mathcal{E}}_2$ denotes the set of existing edges and samples of non-existent edges of $\mathcal{G}_2$ respectively, and $y_{v,v_1} = 1$ if $(v, v_1) \in \mathcal{E}_2$ otherwise $y_{v,v_1} = 0$.

To reconstruct node attributes $\mathbf{Y}_0$ of $\mathcal{G}_2$, we further feed the output node representations $\tilde{\mathbf{Y}}_0$ to an MLP and minimize the mean squared error with the input node attributes $\mathbf{Y}_0$, that is,

$$\mathcal{L}_{\text{attr}} = \frac{1}{m_0}\|\mathbf{Y}_0 - \text{MLP}_2(\tilde{\mathbf{Y}}_0)\|^2_F \tag{16}$$

In addition, we minimize the error of the known cross-network node associations by a margin ranking loss in the network alignment task and by a Bayesian personalized ranking loss [25] in social recommendation. Specifically in network alignment, given a set of triplets $\mathcal{O} = \{(u, v, v_1)|(u, v) \in \mathcal{R}^+, (u, v_1) \in \mathcal{R}^-\}$ where $\mathcal{R}^+ = \{(u, v)|\mathbf{L}(u, v) = 1\}$ denotes the observed node associations, and $\mathcal{R}^- = \{(u, v_1)|\mathbf{L}(u, v_1) = 0, \exists v,\ s.t.\ \mathbf{L}(u, v) = 1\}$ denotes a set of sampled negative associations, the margin ranking loss is

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{O}|}\sum_{(u,v,v_1)\in\mathcal{O}}\max\{0, \lambda - (g_{\text{node}}(u, v) - g_{\text{node}}(u, v_1))\} \tag{17}$$

where λ is the margin size and $g_{\text{node}}(u, v)$ is computed by

$$g_{\text{node}}(u, v) = \left[\mathbf{P}_1^T(\mathbf{Y}_1\tilde{\mathbf{Y}}_0^T)\right](u, v) = \sum_{u'} \mathbf{P}_1(u', u)\left(\mathbf{Y}_1\tilde{\mathbf{Y}}_0\right)(u', v). \quad (18)$$

The overall loss function can be now formulated as below.

$$\mathcal{L} = \alpha\mathcal{L}_{\text{adj}} + \beta\mathcal{L}_{\text{attr}} + \gamma\mathcal{L}_{\text{rank}} \quad (19)$$

3.5 NetTrans: Variants and Generalizations

The proposed *NetTrans* is flexible and can be generalized in multiple aspects. Due to the space limit, we only give a few examples.

- **Bi-directional cross-network transformation.** *NetTrans* can be generalized to a bi-directional transformation model. That is, in addition to transforming from the source network to the target network, the bi-directional model also learns the transformation functions in the reverse direction.
- **Graph-to-subgraph transformation.** When input source network is a large data graph and the target network is a small query graph, *NetTrans* can be tailored to learn the transformation from the data graph to the query graph and the node associations may indicate the subgraph matching.
- **Dynamic network transformation.** When we have a dynamic network $\mathcal{G}$, we can consider $\mathcal{G}^t$ at timestamp t as the source network and $\mathcal{G}^{t+1}$ as the target network. In this case, we can generalize our transformation model to handle dynamic networks and learn how networks evolve over time.
- **Single network auto-encoder.** When the source network and target network are the same network in which case the node associations are naturally known, the proposed *NetTrans* model degenerates to an auto-encoder which captures the hierarchical structure of the network.

4 EXPERIMENTAL RESULTS

We apply the proposed model to network alignment and one-class social recommendation. We evaluate it in the following aspects:

- How accurate is our proposed transformation model for network alignment and recommendation?
- How does our model benefit from the proposed TransPool and TransUnPool layers?

4.1 Experimental Setup

Datasets. The statistics of datasets are summarized in Table 2. Detailed descriptions and experimental settings are in Appendix.

Baseline methods. For network alignment, the baseline methods include: (1) *FINAL-N* [39], (2) *FINAL-P* [38], (3) *REGAL* [11] which is an embedding-based method for attributed networks, (4) *IONE* [21] and (5) *CrossMNA* [3] that are embedding-based methods without attributes. For fair comparisons, we modify *FINAL-N* and *FINAL-P* to semi-supervised setting using $\mathbf{L}$ as the prior alignment matrices. For one-class social recommendation, the baseline methods include (1) *NGCF* [31], (2) *GraphRec* [6] in which we use the BPR loss instead of the default mean square loss, (3) *SamWalker* [2], (4) *wpZAN* [33] that factorizes the node association matrix regularized by social network and product similarity network, and (5) *BPR* which is a classic method based on Bayesian personalized ranking loss [25].

4.2 Performance on Network Alignment

In this subsection, we compare our method *NetTrans* with the baseline methods in scenarios *S1-S3*. We evaluate the effectiveness in terms of Hits@K and alignment accuracy. Given the testing node correspondence (e.g., u, v) across two networks, if $g_{\text{node}}(u, v)$ is among the highest top-K values within all the nodes in $\mathcal{G}_2$, then we say there is a *hit*. We count the number of hits, divided by the total number of testing node correspondences. Besides, the alignment accuracy measures the accuracy of the node one-to-one mappings obtained by using a greedy matching as a postprocess [38]. The results are summarized in Table 3. We have the following observations. First, our proposed method *NetTrans* outperforms both *FINAL-N* and FINAL-P. Specifically, it achieves an up to 6.5% improvement in Hits@30 and an up to 3% improvement in alignment accuracy, compared to *FINAL-N*. Note that both *FINAL-N* and *FINAL-P* can be viewed as the variants of the graph matching-based methods which, as mentioned before, are built upon the linearity/consistency assumptions. Second, our method achieves a better performance than other embedding-based network alignment methods (i.e., *REGAL*, *IONE* and *CrossMNA*). In particular, our method can achieve an at least 20% improvement in alignment accuracy on attributed networks (i.e., scenarios *S1, S2*). This demonstrates the improvements of our method compared to the embedding-based methods that suffer from the embedding space disparity. Third, we observe that even on the networks without node attributes, our proposed model still outperforms the baseline methods. Note that *FINAL-N* and *FINAL-P* have the same performance in scenario *S3* because they are essentially equivalent without attributes.

Table 2: Data Statistics.

Tasks	Networks	# of nodes	# of edges	# of attributes
Network Alignment	Cora-1	2,708	5,806	1,433
	Cora-2	2,708	4,547	1,433
	ACM	9,872	39,561	17
	DBLP	9,916	44,808	17
	Foursquare	5,313	54,233	1
	Twitter	5,120	130,575	1
Recommendation	Ciao-user	3,719	65,213	1
	Ciao-product	4,612	49,136	28

Ablation study on the TransPool layer. To show the effectiveness of the proposed TransPool layer in identifying cross-network associations, we compare with two variants by replacing TransPool with the existing pooling layers *UNetPool* [9] and *SAGPool* [19]. Since these two pooling layers originally do not learn the assignment matrices $\mathbf{P}_l$, we calculate the inner products between the node representations and the supernode representations in the l-th pooling layer, followed by a softmax as $\mathbf{P}_l$. From Figure 5 we can see that using *TransPool* as the pooling layer significantly outperforms the other two variants. This implies the *TransPool* layer can learn better node representations and assignment matrices at different resolutions to help identify the cross-network node associations.

Ablation study on the TransUnPool layer. To show the effectiveness of the proposed unpooling layer, we compare with the different variants in learning the structure and node representations of the target network at different resolutions. These variants include: (1) *NetTrans-Skip* that directly uses the coarsened source network at the same resolution (i.e., $\mathbf{A}_{L-k} = \mathbf{B}_{L-k}$ and $\mathbf{X}_{L-k} = \mathbf{Y}_{L-k}$), (2) *NetTrans-w/o-Attr* that only calibrates the structure (i.e., without calculating Eq. (9)), and (3) *NetTrans-w/o-Adj* that in contrast only calibrates the node representations (i.e., without calculating Eq. (10)). From Figure 6 we can see that the TransUnPool layer significantly outperforms other variants indicating the importance of the calibrations to earn the transformation across different networks.

Table 3: (Higher is better.) Effectiveness results on network alignment.

	Cora1-Cora2			ACM-DBLP			Foursquare-Twitter		
	Hits@10	Hits@30	Accuracy	Hits@10	Hits@30	Accuracy	Hits@10	Hits@30	Accuracy
NetTrans	**90.98%**	**97.51%**	**89.89%**	**84.09%**	**94.52%**	**58.21%**	**24.68%**	**34.58%**	**9.17%**
FINAL-N	88.73%	90.77%	87.58%	82.91%	90.71%	54.39%	24.09%	33.80%	8.47%
FINAL-P	62.28%	80.01%	54.34%	69.70%	83.12%	36.34%	24.09%	33.80%	8.47%
REGAL	60.90%	69.20%	46.26%	63.68%	71.80%	41.78%	0.15%	2.20%	0.11%
IONE	73.03%	79.92%	42.29%	58.93%	84.19%	33.00%	13.44%	28.17%	4.13%
CrossMNA	59.06%	68.62%	33.26%	42.54%	49.69%	21.04%	3.37%	14.79%	2.48%

Table 4: (Higher is better.) Effectiveness results on social recommendation.

	Ciao-0.2			Ciao-0.3			Ciao-0.5		
	Prec@10	Rec@10	Rec@50	Prec@10	Rec@10	Rec@50	Prec@10	Rec@10	Rec@50
NetTrans	**13.87%**	**11.08%**	**29.90%**	**11.01%**	**13.23%**	**28.15%**	**10.87%**	**12.43%**	**39.02%**
BPR	1.37%	0.6%	20.25%	1.38%	0.62%	20.18%	1.00%	0.37%	14.97%
wpZAN	11.99%	9.19%	20.77%	9.88%	10.33%	23.22%	9.85%	11.64%	26.04%
GraphRec	8.65%	6.62%	17.56%	8.42%	6.60%	18.07%	6.94%	6.63%	18.08%
SamWalker	4.94%	1.97%	5.98%	4.39%	2.07%	5.67%	2.48%	1.58%	4.05%
NGCF	2.77%	1.21%	3.26%	2.77%	1.48%	3.61%	3.17%	1.99%	4.77%

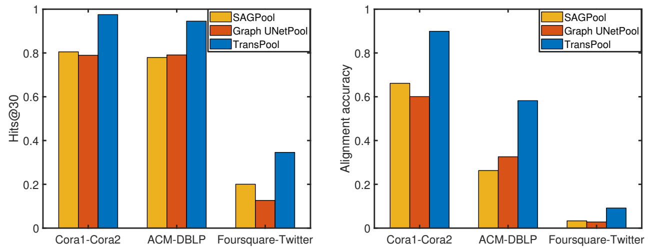


Figure 5: Ablation study on the pooling layer.

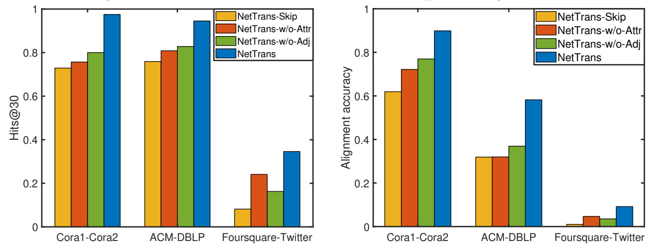


Figure 6: Ablation study on the unpooling layer.

4.3 Performance on Recommendation

In addition to network alignment, we apply our proposed model to one-class social recommendation to predict whether users interact with certain products. In our experiments, we evaluate the performance by precision@K and recall@K. The results are summarized in Table 4. We have the following observations. First, our proposed method outperforms all the baseline methods. Specifically, our method achieves an at least 1% improvement in Precision@10 and an at least 5% improvement in Recall@50 compared to the best baseline method. Second, our method and *wpZAN* that is dual-regularized by both social and product networks outperform other baseline methods, which implies leveraging the network structure, especially the product network, is indeed helpful.

5 RELATED WORKS

Network alignment. Many traditional graph matching based network alignment methods and their variants often assume networks are noisy permutations of each other. For example, Umeyama proposes to minimize $\|\mathbf{B}_0 - \mathbf{P}\mathbf{A}_0\mathbf{P}^T\|_F^2$ by eigen-decomposition [29]. Koutra et al. generalize the graph matching based methods to align bipartite graphs [17] and Zhang et al. propose to solve the multiple network alignment problem with transitivity constraints [37]. Besides, Zhang et al. propose an attributed alignment algorithm *FINAL* based on the alignment consistency [38]. Du et al. accelerate the algorithm based on Krylov subspace [4]. These graph matching based methods are often built on the linearity/consistency assumptions.

Moreover, there exist many network embedding based methods, including *IONE* [21] and *CrossMNA* [3]. However, these methods suffer from the space disparity as the unobserved anchor nodes could be incomparable in the embedding space. To address this issue, Zhang et al. propose to leverage the non-rigid point-set registration [40]. *MrMine* mitigates this issue by forcing embedding vectors in the same space, but it needs networks with similar structures [5].
Social Recommendation. Matrix factorization-based methods [14, 16] are widely used for recommendation. Classic social recommendation approaches are typically based on network-regularized matrix factorization. For example, *SoRec* co-factorizes both social network and the rating matrix with the shared user feature matrix [24]. *TrustMF* factorizes the social trust network to capture both the truster and trustee relations [32]. These methods implicitly embrace the linear operations. Graph neural networks based methods are recently proposed, including *GraphRec* [6] and *DSCF* [7]. These methods only exploit the social relations and user-product interactions. Yao et al. propose to regularize the matrix factorization by the homophily in both social and product similarity networks [34].
Graph neural networks. Many graph neural network models have been proposed [15, 23] and a detailed review can be found in [41]. In addition, to learn the hierarchical representations of networks, many graph pooling operations have been proposed including the differentiable pooling [35] and [9, 19] based on top-k important node selection. Graph-to-graph translation [10, 13] also relate to our model, but they implicitly require node correspondence and assume nodes in different networks are of same type.

6 CONCLUSION

Finding the node associations across different networks is a key step to many real-world applications. In this paper, we tackle the cross-network node associations by studying a novel cross-network transformation problem. To address this problem, we propose an end-to-end model *NetTrans* that learns a composition of nonlinear functions to transform one network to another. In details, we propose a novel graph pooling operation *TransPool* and an unpooling operation *TransUnPool*. We perform extensive experiments in network alignment and social recommendation that validate the effectiveness of our model to find the cross-network associations.

7 ACKNOWLEDGEMENTS

This work is supported by National Science Foundation under grant No. 1947135, and 1715385, by the NSF Program on Fairness in AI in collaboration with Amazon under award No. 1939725, by the United States Air Force and DARPA under contract number FA8750-17-C-0153 [3], Department of Homeland Security under Grant Award Number 17STQAC00001-03-03. The content of the information in this document does not necessarily reflect the position or the policy of the Government or Amazon, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] Chen Chen, Hanghang Tong, Lei Xie, Lei Ying, and Qing He. 2016. FASCINATE: fast cross-layer dependency inference on multi-layered networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 765–774.

[2] Jiawei Chen, Can Wang, Sheng Zhou, Qihao Shi, Yan Feng, and Chun Chen. 2019. Samwalker: Social recommendation with informative sampling strategy. In *The World Wide Web Conference*. 228–239.

[3] Xiaokai Chu, Xinxin Fan, Di Yao, Zhihua Zhu, Jianhui Huang, and Jingping Bi. 2019. Cross-Network Embedding for Multi-Network Alignment. In *The World Wide Web Conference*. 273–284.

[4] Boxin Du and Hanghang Tong. 2018. FASTEN: Fast Sylvester equation solver for graph mining. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1339–1347.

[5] Boxin Du and Hanghang Tong. 2019. MrMine: Multi-resolution Multi-network Embedding. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 479–488.

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*. 417–426.

[7] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep social collaborative filtering. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 305–313.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[9] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. *arXiv preprint arXiv:1905.05178* (2019).

[10] Xiaojie Guo, Liang Zhao, Cameron Nowzari, Setareh Rafatirad, Houman Homayoun, and Sai Manoj Pudukotai Dinakarrao. 2019. Deep Multi-attributed Graph Translation with Node-Edge Co-evolution. In *he 19th International Conference on Data Mining (ICDM 2019), pp. to appear.*

[11] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. 2018. Regal: Representation learning-based graph alignment. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 117–126.

[12] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[13] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. 2018. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv preprint arXiv:1812.01070* (2018).

[14] Jian Kang and Hanghang Tong. 2019. N2N: Network Derivative Mining. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 861–870.

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

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[17] Danai Koutra, Hanghang Tong, and David Lubensky. 2013. Big-align: Fast bipartite graph alignment. In *2013 IEEE 13th International Conference on Data Mining*. IEEE, 389–398.

[18] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. 1188–1196.

[19] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082* (2019).

[20] Jundong Li, Chen Chen, Hanghang Tong, and Huan Liu. 2018. Multi-layered network embedding. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 684–692.

[21] Li Liu, William K Cheung, Xin Li, and Lejian Liao. 2016. Aligning Users across Social Networks Using Network Embedding.. In *Ijcai*. 1774–1780.

[22] Qiao Liu, Chen Chen, Annie Gao, Hang Hang Tong, and Lei Xie. 2017. VariFunNet, an integrated multiscale modeling framework to study the effects of rare non-coding variants in genome-wide association studies: Applied to Alzheimer's disease. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2177–2182.

[23] Zhining Liu, Dawei Zhou, and Jingrui He. 2019. Towards Explainable Representation of Time-Evolving Graphs via Spatial-Temporal Graph Attention Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2137–2140.

[24] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 931–940.

[25] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[26] Dorit Ron, Ilya Safro, and Achi Brandt. 2011. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation* 9, 1 (2011), 407–423.

[27] Jiliang Tang, Huiji Gao, and Huan Liu. 2012. mTrust: discerning multi-faceted trust in a connected world. In *Proceedings of the fifth ACM international conference on Web search and data mining*. 93–102.

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

[29] Shinji Umeyama. 1988. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence* 10, 5 (1988), 695–703.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[31] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[32] Bo Yang, Yu Lei, Jiming Liu, and Wenjie Li. 2016. Social collaborative filtering by trust. *IEEE transactions on pattern analysis and machine intelligence* 39, 8 (2016), 1633–1647.

[33] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[34] Yuan Yao, Hanghang Tong, Guo Yan, Feng Xu, Xiang Zhang, Boleslaw K Szymanski, and Jian Lu. 2014. Dual-regularized one-class collaborative filtering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 759–768.

[35] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.

[36] Jiawei Zhang and S Yu Philip. 2015. Integrated anchor and social link predictions across social networks. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[37] Jiawei Zhang and S Yu Philip. 2015. Multiple anonymized social networks alignment. In *2015 IEEE International Conference on Data Mining*. IEEE, 599–608.

[38] Si Zhang and Hanghang Tong. 2016. Final: Fast attributed network alignment. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1345–1354.

[39] Si Zhang and Hanghang Tong. 2018. Attributed Network Alignment: Problem Definitions and Fast Solutions. *IEEE Transactions on Knowledge and Data Engineering* 31, 9 (2018), 1680–1692.

[40] Si Zhang, Hanghang Tong, Jiejun Xu, Yifan Hu, and Ross Maciejewski. 2019. Origin: Non-rigid network alignment. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 998–1007.

[41] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6, 1 (2019), 11.

[3] Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

Reproducibility

Dataset descriptions. The datasets that we used in the experiments include:

- *Cora citation network*: This dataset contains a citation network where nodes represent documents and edges represent the citations among documents. Each document has a binary feature vector represented by bag-of-words [33].
- *ACM co-author network*: This dataset was collected in 2016 including 2,381,688 papers with the author and venue information of each paper [28]. A co-author network was then extracted based on the papers published in four areas (DM, ML, DB and IR) in [39]. Nodes in the co-author network represent authors and edges indicate the co-authorship. The numbers of papers published by an author in 17 venues are used as the node attributes.
- *DBLP co-author network*: This dataset was collected in 2016 and it contains 3,272,991 papers. A co-author network was extracted in [39] similarly to the ACM dataset.
- *Foursquare*: This dataset contains a social network with nodes as users and edges as the friendships [36].
- *Twitter*: This contains a social network where nodes represent users and edges represent the friendships [36].
- *Ciao*: This dataset contains a social network whose edges indicate the trust relationships among users and a set of user-product ratings with rich attribute information of product [27]. We extract 3,719 users who like more than 10 products and 4,612 products that are liked by these users. We use the product categories as the attributes. We consider ratings greater than or equal to 3 as interactions (i.e., likes) and obtain 105,900 user-product interactions.

With these datasets, we construct the scenarios of network alignment *S1-S3* and the recommendation scenario *S*4 for evaluations:

- *S1. Cora-1 vs. Cora-2*: Given the cora citation network, we generate two permuted networks $\mathcal{G}_1, \mathcal{G}_2$ and add noises by first inserting 10% edges to $\mathcal{G}_1$ and remove 15% edges from $\mathcal{G}_2$, and then adding 10% noises to $\mathbf{X}_0, \mathbf{Y}_0$ (i.e., by randomly changing $0.1 \times \mathbf{1}^T\mathbf{X}_0\mathbf{1}$ entries from 0 to 1). In this scenario, we aim to align the nodes in $\mathcal{G}_1, \mathcal{G}_2$. The permutation matrix is used as the ground-truth node correspondences.
- *S2. ACM vs. DBLP*: We aim to find the node correspondences across two co-author networks. There exist 6,325 common authors across two networks used as the ground-truth.
- *S3. Foursquare vs. Twitter*: In this scenario, we aim to align nodes in Foursquare and Twitter networks. There are 1,609 common users which are used as the ground-truth.
- *S4. Ciao users vs. product*: Different from the above scenarios, here we aim to predict the node associations between users and products indicating whether a user likes a product. To construct the product similarity network, similar to [34], we compute the similarities based on the embedding vectors of product reviews. The embedding vectors are learned by doc2vec [18]. Then we consider there exists an edge between two products if their similarity is larger than 0.5.

Besides, in *S1-S3*, we use 20% of the ground-truth as the training data (i.e., $\mathbf{L}$) and test on the rest of the ground-truth. In *S4*, we evaluate the performance in three sub-scenarios Ciao-r ($r \in \{0.2, 0.3, 0.5\}$) with different training ratios 20%, 30% and 50%, respectively.

Machine. The proposed model is implemented in Pytorch. We use one Nvidia GTX 1080 with 8G RAM as GPU.

Hyperparameters settings. We use Adam optimizer with a learning rate 0.005 to train the model. For network alignment, we set the margin size $\lambda = 0.1$, $\alpha = \beta = 1$, $\gamma = 10$ and the dimension of hidden representations in all layers to 256. As for the model architecture, in *S1*, we use $L = 2$, $n_1 = 2000$ and $n_2 = 1000$. In *S2*, due to the GPU memory limit, we use $L = 1$ and $n_1 = 5000$. In *S3*, we use $L = 2$, $n_1 = 5000$ and $n_2 = 2500$. For one-class social recommendation, we use the classic Bayesian personalized ranking loss to replace Eq. (17) and set $\alpha = \beta = 1$, $\gamma = 100$ and the dimensions of the representations to 128. The model architecture that we use is $L = 2$, $n_1 = 3000$ and $n_2 = 1500$. In both tasks, we set the negative sample size in Eq. (15) to 5 and that in the ranking loss Eq. (17) to 100. We use the same embedding dimensions for embedding-based methods, and other parameters in the baseline methods are set to default.

Proof of Proposition 1

Proposition 2. *Given an assignment matrix* $\mathbf{P}_l$*, the coarsening process* $\mathbf{A}_l = \mathbf{P}_l\mathbf{A}_{l-1}\mathbf{P}_l^T$ *constructs edges by linear operations.*

Proof. By eigenvalue decomposition on $\mathbf{A}_{l-1}$, $\mathbf{A}_{l-1} = \mathbf{U}\Sigma\mathbf{U}^T$, we have

$$\mathbf{A}_l(u_1', u_2') = (\mathbf{P}_l\mathbf{U}\Sigma\mathbf{U}^T\mathbf{P}_l^T)(u_1, u_2) = [\mathbf{P}_l(u_1', :)\mathbf{U}]\Sigma[\mathbf{P}_l(u_2', :)\mathbf{U}]^T$$

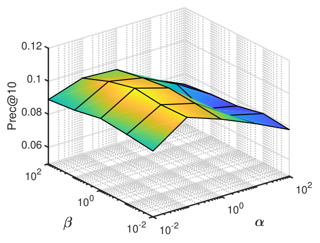
where $\mathbf{U} \in \mathbb{R}^{n_{l-1}\times r}$ and $r < n_{l-1}$ only if $\mathbf{A}_{l-1}$ is low-rank. In this way, by considering U as node representations, the existence of an edge and its weight between supernode u_1' and supernode u_2' is determined equivalently by first computing supernodes' representations by linear aggregations based upon assignment matrix $\mathbf{P}_l$, andthen a weighted inner product between the representations of supernodes u_1', u_2'. Both steps only involve linear operations on node representations. □

Model size.

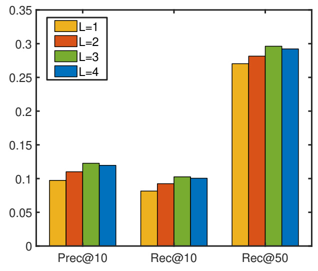
The parameters of TransPool layer are $\Theta_l^p = \{\mathbf{W}_l^{\text{self}}, \mathbf{W}_l^1, \mathbf{W}_l^g, \mathbf{a}_l\}$. The total number of parameters are $2d_l^2 + 3d_l$. Besides, in the k-th TransUnPool layer, the parameters include $\Theta_k^u = \{\mathbf{W}_k^2, \mathbf{W}_k^3\}$ which have $2d_{L-k+1}d_{L-k}$ parameters. In the last TransUnPool layer, we additionally have d_1d_0 more parameters introduced by $\mathbf{W}_L^4$. In this way, the proposed pooling and unpooling layers have an affordable number of parameters.

Discussions on the Vector Space Comparability

Note that most of the existing embedding-based methods compute the node association scores based on the similarities of embedding vectors across different networks [3, 21, 31], which basically compare all node embedding vectors of different networks in the same vector space. Different from those methods, in Eq. (18), we do not attempt to compare node representations of different networks in the same space. Instead, we first compute the node-supernode assignment matrix of the target network by the inner products between the representations of the nodes $\tilde{\mathbf{Y}}_0$ and those of supernodes $\mathbf{Y}_1$. Note that here we hypothesize that $\tilde{\mathbf{Y}}_0$ and $\mathbf{Y}_1$ are more comparable



(a) Sensitivity of α and β.



(b) Sensitivity of L.

Figure 7: Sensitivity study on hyperparameters.

as they are both in the context of target network but at different resolutions. In addition, under our assumption that two supernodes across different networks represent the same latent entity (e.g., same supernode-u'), it could be more comparable to compare the association score between u and v through the supernodes.

Hyperparamter sensitivity study

We conduct the hyperparameter sensitivity studies for recommendation on the loss coefficients α, β and on the number of layers. From Figure 7 (a) we can see that our model is robust to different choices of α, β ranging from 0.01 to 100. In addition, from Figure 7 (b), we can observe that our model achieves the best performance with 3 encoder layers and 3 decoder layers (i.e., $L = 3$). This demonstrates that learning hierarchical representations of the networks at different resolutions can lead to a better performance.