Graph Transfer Learning

Andrey Gritsenko, Yuan Guo, Kimia Shayestehfard, Armin Moharrer, Jennifer Dy, Stratis Ioannidis

Department of Electrical and Computer Engineering, Northeastern University, Boston, MA
{agritsenko, yuanee20, kshayestehfard, amoharrer, jdy, ioannidis}@ece.neu.edu

***Abstract*—Graph embeddings have been tremendously successful at producing node representations that are discriminative for downstream tasks. In this paper, we study the problem of graph transfer learning: given two graphs and labels in the nodes of the first graph, we wish to predict the labels on the second graph. We propose a tractable, non-combinatorial method for solving the graph transfer learning problem by combining classification and embedding losses with a continuous, convex penalty motivated by tractable graph distances. We demonstrate that our method successfully predicts labels across graphs with almost perfect accuracy; in the same scenarios, training embeddings through standard methods leads to predictions that are no better than random.**

I. Introduction

WE consider a *graph transfer learning* problem, illustrated by the following motivating example. An epidemic spreading through a graph is observed by an analyst. The statistics governing the epidemic propagation are a priori unknown; nevertheless, the analyst wishes to use this trace to predict how the epidemic would spread over a new graph, potentially modeling a different population. More broadly speaking, we wish to solve the following abstract problem. A learner is presented with two structurally similar (but distinct) graphs G_A and G_B. Node labels such as, e.g., infection probabilities, community membership, etc., are provided only for nodes on G_A. A learner wishes to *use the labels on G_A to predict the labels on G_B*.

Intuitively, the success of such a transfer learning task relies on the fact that many interesting labels depend on *structural* or *topological* features of nodes. For example, membership in a cluster, susceptibility to an infection during a cascade, pagerank scores, etc., are all properties that depend on the relative position (w.r.t. clusters, weakly connected components, centrality, etc.) nodes have in a graph. A classifier trained over such labels in G_A should be transferable to a new, structurally similar graph G_B. In the extreme, when graphs G_A and G_B are isomorphic, G_B's labels should be fully recoverable; conversely, one expects transferability to degrade over highly dissimilar graphs.

A natural challenge that arises in this setting is in how to abstract (and transfer) topological information across the two graphs. In this paper, we address this challenge by leveraging *graph embeddings* [1]–[3]. Graph embeddings have been tremendously successful at producing compact representations of nodes in a graph, and have become a true workhorse of graph mining. In short, graph embeddings map nodes of a graph into a lower-dimensional space (e.g., $\mathbb{R}^d$, for some small d); this mapping concisely captures node connectivity, recovered from embeddings through an appropriate link function. Embeddings therefore naturally abstract structural information through the node's position in this lower-dimensional space. In addition, embeddings reduce graph transfer learning to classic transfer learning [4]: a classifier trained over labels and embeddings of nodes in graph G_A can be transferred to a new feature domain, namely, the embeddings of G_B's nodes.

Unfortunately, successfully transferring knowledge via state-of-the-art embeddings poses significant challenges. A classifier trained on embeddings of one graph *is generally no better than random guessing* when applied to embeddings of another graph: we provide a theoretical justification for this in Section IV-A, and demonstrate it also experimentally in Section V. In short, classifiers catastrophically fail to transfer across embeddings of different graphs because of an embedding misalignment: as designed, none of the popular graph embedding methods ensure that nodes of two distinct graphs are embedded over the *same* lower-dimensional subspace or manifold. In general, embeddings capture only the *relative*, rather than the *absolute*, position of nodes in $\mathbb{R}^d$. This is sufficient for inference tasks on nodes of the same graph (like, e.g., link prediction) but disastrous when transferring knowledge across graphs: the same embedding algorithms applied to two isomorphic graphs may generate vastly different embeddings, that are distorted via arbitrary shifts, rotations, or other transforms. This severely hampers the ability to transfer structural classifiers across graphs.

We directly address this issue by producing a tractable, non-combinatorial methodology for solving the graph transfer learning problem. We do so by *learning joint embeddings* across the two graphs. This allows us to successfully transfer a classifier trained on labels of one graph to another. We make the following contributions:

- We introduce novel methodology for solving the graph transfer learning problem in a non-combinatorial fashion. Our method is general, and can be applied to a broad array of graph embedding algorithms. Moreover, it combines classification and embedding losses with a continuous, convex coupling penalty motivated by tractable graph distances [5].
- Our continuous and convex coupling penalty seamlessly integrates with deep embedding methods. We propose and implement an alternating minimization algorithm that *jointly* embeds the two graphs. Our algorithm does so without solving the combinatorial (and hard) problem of aligning the two graphs: instead, it alternates between using SGD and solving a convex optimization problem constrained over the Birkhoff polytope [6].
- We extensively evaluate our proposed graph transfer learning methodology over several synthetic and real-life datasets. We demonstrate that it successfully predicts labels across graphs with almost perfect accuracy; in the same scenarios, training embeddings separately leads to predictions that are no better than random.

To the best of our knowledge, we are the first to study the graph transfer learning problem, and to propose a non-combinatorial method for its solution.

II. Related Work

Graph Embeddings and Graph Neural Networks. Graph embedding research has flourished recently [1]–[3], [7]. We thoroughly review techniques as well as specific algorithms in Section III, following the unifying framework of Hamilton et al. [8]. Typically, embeddings preserve node similarity in the embedding space, and thus require the definition of similarity on both the embedding space as well as on graph nodes [9], [10]. We list several examples in Table II. Graph neural networks (GNNs) [8], [11]–[13] produce graph embeddings by generalizing the notion of a convolution, aggregating information from neighboring nodes, in analogy to conventional convolutional neural networks. Our transfer learning approach is generic, and applies to the majority of the methods outlined above, including GNNs. Moreover, the challenges posed by graph transfer learning we outline in Section IV-A are pertinent to all these methods, and are exacerbated by deep models, as non-convexity increases the multiplicity of local minima.

Transfer Learning on Graphs. Transfer learning in the general machine learning setting aims to apply knowledge gained while solving one task to a different but related task [4]. A quintessential example is transferring a text classifier from language to another [14]. Transfer learning has been applied to graphs only recently; all current work however [15]–[17] considers classifying (and transferring labels across) graphs, as opposed to nodes. To the best of our knowledge, we are the first to tackle transferring structural node labels between graphs.

Notation	Description
G_A, G_B	Graphs
V	Node set of graphs G_A, G_B
E_A, E_B	Edge sets of graphs G_A, G_B
z_i^A, z_j^B	Embedding of nodes in G_A and G_B
$s_G(i,j)$	Topological similarity between nodes $i, j \in V$
$s_E(z_i, z_j)$	Similarity between node embeddings z_i, z_j
y_i^A	Label of node $v_i^A \in V_A$
`ACC`	Classification accuracy
`RMSE`	Root mean squares error
$\mathtt{R}^2$	Coefficient of determination
$\mathcal{L}_S$	Embedding loss – Eq. (3.2b)
$\mathcal{L}_C$	Classification loss – Eq. (3.6a)
$\mathcal{L}_P$	Penalty function (4.10)
$\mathcal{L}$	Aggregate loss – Eq. (4.8a)
W, W_A, W_B, W'	Neural network weights
P	Doubly stochastic matrix
$\mathbb{B}$	Birkhoff polytope – Eq. (4.12)
$\mathbb{P}$	Set of permutation matrices

TABLE I: Summary of notation.

Graph distances. There exist graph alignment heuristics (see, e.g., [18], [19]) that are tractable, but do not satisfy the metric property. Our tractable penalty is based on, and inspired by, recent work by Bento and Ioannidis [5]. The authors propose a family of graph distances that are (a) computable in polynomial time and (b) satisfy the metric property. We incorporate this formulation as a penalty into our framework and use it to couple the embeddings of two graphs in order to transfer the learned classifier.

Epidemic Learning. The seminal paper by Kempe et al. [20], has motivated learning the parameters of an epidemic spread (e.g., [21]–[23]). Typically, this is done via maximum likelihood estimation over a generative model, e.g., the independent cascades (IC) or linear threshold (LT) models [20]. We learn from cascades in one graph and transfer knowledge to another graph. We thus avoid intermediate parameter inference and modeling assumptions (such as the IC or LT model), that may not hold in practice.

III. Background

A. Node Embeddings. The goal of *node embedding* algorithms is to learn parsimonious node representations that are discriminative w.r.t. downstream tasks such as community detection, link prediction, etc. We follow the framework of Hamilton et al. [8] that unifies multiple different node embedding methods.

A Unifying Framework. Given a graph $G(V, E)$ with $n = |V|$ nodes, let $x_i \in \{0,1\}^n$ be the 1-hot encoding of a node $i \in V$ in the graph. An embedding is a parametric function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^d$, where $d \ll n$, mapping nodes to d dimensional vectors; that is,

$$z_i = f(x_i, W) \in \mathbb{R}^d \tag{3.1}$$

Method	$s_E(z_i, z_j)$	$s_G(i,j)$	Loss function ℓ_S
Laplacian Eigenmaps [2]	$-\|z_i - z_j\|_2^2$	n-neighborhood	$-s_E(z_i,z_j)\cdot s_G(i,j)$
Graph Factorization [1]	$z_i^\top z_j$	$A_{i,j}$	$(s_E(z_i,z_j) - s_G(i,j))^2$
GraRep [7]	$z_i^\top z_j$	$A_{i,j}, A_{i,j}^2, \ldots, A_{i,j}^k$	$(s_E(z_i,z_j) - s_G(i,j))^2$
node2vec [3]	$\frac{e^{z_i^\top z_j}}{\Sigma_{k\in V} e^{z_i^\top z_k}}$	$p(i\|j)$	$-s_G(i,j)\log(s_E(z_i,z_j))$

TABLE II: Different embedding methods expressed in the unifying framework of Hamilton at al. [8]. In node2vec, $p(i|j)$ is the probability of visiting node j on a fixed-lenght random walk from node i.

is the embedding z_i of node $i \in V$, and $W \in \mathbb{R}^m$, for some $m \in \mathbb{N}$, are weights parametrizing the embedding function. For example, f could be a neural network with by weights W, an affine (shallow) function, etc. Note that this representation can readily incorporate node attributes, that can be represented via features in input vectors x_i.

Keeping the exposition on one-hot encoding for concreteness, the parameters of the embedding can be trained as follows. Given a *topological similarity* function $s_G : V \times V \to \mathbb{R}$ between nodes as well as an *embedding similarity* function $s_E : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ between embeddings, the node embedding task can be formulated via the following minimization problem:

$$\min_{W\in\mathbb{R}^m} \mathcal{L}_S(W;G), \text{ where} \tag{3.2a}$$

$$\mathcal{L}_S(W;G) = \textstyle\sum_{i,j\in V} \ell_S(s_G(i,j), s_E(z_i,z_j)), \tag{3.2b}$$

$$z_i = f(x_i, W), \quad \forall i \in V, \tag{3.2c}$$

and $\ell_S : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ an appropriately defined loss function. Typically, Prob. (3.2) is solved via stochastic gradient descent over the nodes, although techniques like hierarchical softmax [24] and negative sampling [25] can be incorporated to accelerate computations.

Examples. The topological similarity s_G can be, e.g., node adjacency or proximity in path distance. That is, if A is the adjacency matrix of $G(V,E)$, and d_{ij} is the shortest path distance between $i, j \in V$ then two possible similarities are $s_G(i,j) = A_{ij}$ and $s_G(i,j) = 1/d_{ij}$. Other alternatives include, e.g., powers of the adjacency matrix, the probability that a random walk starting at i terminates at j after a small number of steps, etc. Several examples are provided in Table II (see also [8]). For example, Laplacian Eigenmaps [2] couple Euclidian distance with a product loss, yielding:

$$\mathcal{L}_S(W;G) = \textstyle\sum_{i,j\in V} \|z_i - z_j\|_2^2 \cdot s_G(i,j), \tag{3.3}$$

while Graph Factorization [1] couples an inner product with a quadratic loss, yielding:

$$\mathcal{L}_S(W,G) = \textstyle\sum_{i,j\in V} (z_i^\top z_j - s_G(i,j))^2. \tag{3.4}$$

B. Node Label Prediction. Node embeddings often serve as an intermediate step for downstream supervised learning tasks on graphs, such as community detection, link prediction, etc. For example, given binary labels $y_i \in \{0,1\}$ for nodes $i \in S \subseteq V$, learning embeddings that are discriminative w.r.t. these labels can be accomplished by extending Eq. (3.2) as follows:

$$\min_{W\in\mathbb{R}^m, W'\in\mathbb{R}^{m'}} \mathcal{L}_S(W;G) + \mathcal{L}_C(W,W';y_S,G), \tag{3.5}$$

where $\mathcal{L}_S(W;G)$ is the similarity loss (3.2b), while

$$\mathcal{L}_C(W,W';y_S,G) = \textstyle\sum_{i\in S} \ell_C\big(y_i, g(z_i,W')\big), \text{ and} \tag{3.6a}$$

$$z_i = f(x_i, W), \quad \forall i \in V. \tag{3.6b}$$

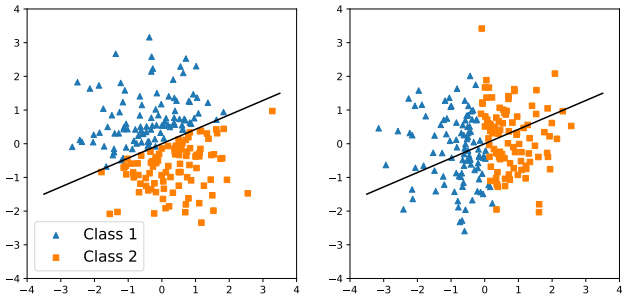
Here, $\ell_C : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function (such as, e.g., square error, logistic, or cross-entropy), $y_S \in \{0,1\}^{|S|}$ is the vector of labels, and $g : \mathbb{R}^d \times \mathbb{R}^{m'} \to \mathbb{R}$ is a function (i.e., a prediction model) parametrized by $W' \in \mathbb{R}^{m'}$, mapping node embeddings to labels. This can again be a deep or shallow model (e.g., logistic regression). Problem (3.5) can again be solved via stochastic gradient descent, where an epoch iterates over batches node pairs $i, j \in V$ *and* labeled nodes $i \in S$.

IV. Graph Transfer Learning

A. Problem Formulation. In this paper, we wish to solve the *graph transfer learning* problem. Given two graphs and labels in the nodes of the first graph, we wish to predict the labels *on the second graph*. As discussed in the introduction, labels such as community membership, susceptibility to an infection, centrality, etc., may be functions of structural properties of a node and, as a result, may be transferable across graphs. Formally, we are given two unweighted graphs $G_A(V_A, E_B)$ and $G_B(V_B, E_B)$ of the same size (i.e., $|V_A| = |V_B| = n$), as well as a set of labels y_i for $i \in S \subseteq V_A$. For example, $y_i \in \{0,1\}$ for $i \in S$ in a binary classification task, $y_i \in \mathbb{R}$ in the case of a regression task, etc. We wish to train a neural network over labels in G_A, and use it to subsequently predict labels in G_B. We focus first on unweighted graphs of equal size for the sake of simplicity; we extend out method to weighted graphs and graphs of unequal size in Section IV-D.

A Naïve Solution. The node embedding and node label prediction algorithms we reviewed in Section III give a possible simple solution to the graph transfer learning problem. First, a discriminative embedding is trained on graph G_A, by solving Prob. (3.5): this gives both an embedding $f(\cdot, W_A)$ and a predictive model $g(\cdot, W')$. Second, an embedding $f(\cdot, W_B)$ is trained on graph G_B, by solving Prob. (3.2) on G_B alone. Finally, the predictive model $g(\cdot, W')$ is applied on the embeddings of nodes in graph G_B to predict their labels. Altogether, this naïve algorithm solves the following problem, which is separable over (W_A, W') and W_B:

$$\begin{aligned}\min_{W_A,W_B,W'} \mathcal{L}_S(W_A;G_A) + \mathcal{L}_C(W_A,W';y_S,G_A) \\ + \mathcal{L}_S(W_B;G_B),\end{aligned} \tag{4.7}$$



(a) Graph G_A (b) Graph G_B

Fig. 1: Example of two isomorphic embeddings and the failure to transfer a learned classifier across them. In (a) embeddings of a G_A are used to train an almost perfect classifier between two classes. Embeddings of G_B in (b) are identical to G_A but subject to a rotation; as a result, the classifier trained in G_A does not readily generalize to G_B.

where $\mathcal{L}_S, \mathcal{L}_C$ are given by Eqs. (3.2b) and (3.6a), respectively. Unfortunately, this approach is bound to fail; we extensively demonstrate this experimentally in Section V, and give some intuition as to why this is the case below.

Non-Uniqueness. It is easy to see that Eq. (4.7) fails to transfer the learned classifier by considering the case when the two graphs G_A and G_B are isomorphic. In this case, nodes that map to each other should have the same embeddings and, thereby, the same labels. Unfortunately, *none of the methods outlined in Table II are guaranteed to produce the same embeddings for nodes in V_A and V_B.* This is because of *non-uniqueness*: the non-convexity of the loss $\mathcal{L}_S$ for all of these methods implies that optimal embeddings (i.e., solutions to Prob. (3.2)) are non-unique. In turn, this non-uniqueness implies that the embeddings of the same graph, or two isomorphic graphs, can be vastly different at two different executions of the algorithm.

For several objectives, non-uniqueness manifests through arbitrary transformations of the latent space, via rotations, shifts, or other transforms. In turn, this "breaks" transferring a node classifier learned on one graph to another via the above naïve method. This is illustrated in Fig. 1: clearly, a classifier trained on a set of samples fails to correctly classify exactly the same samples when the latter are rotated. Simply put, the separating surface (e.g., hyperplane for a shallow linear classifier like logistic regression) is *not* invariant to the aforementioned transforms that relate embeddings between different graphs; as a result, embeddings trained across the two graphs can be misaligned. This suggests that *embeddings across graphs need to be trained jointly, maintaining an appropriate alignment.* We accomplish this, via a non-combinatorial method, in the next section.

The use of different random seeds or starting points, the use of deep neural networks, that may introduce additional local minima, and departures from perfect isomorphism (i.e., different edges in the two graphs), all further exacerbate the problem of non-uniqueness. Most importantly, as non-uniqueness is a consequence of the non-convexity of the objective, *it arises irrespective of whether embedding functions are shallow or deep, whether inputs x_i are features or one-hot encodings, or whether, e.g., graph neural networks are used.* In the latter case, it is tempting to think that embeddings are, by design, linked to topological properties of the position of a node in the graph, and thereby are invariant (at least if graphs are isomorphic). However, this is *not* true: the non-convexity of the objective makes such methods also susceptible to variations due to randomness, initialization conditions, and departures from perfect isomorphism. We also demonstrate this experimentally in Section V, exploring three direct encoding methods, viz. *Laplacian Eigenmaps* [2], *Graph Factorization* [1], and *node2vec* [3], as well as graph neural network *GraphSAGE* [13]: all four algorithms fail to transfer across graphs for the aforementioned reasons (see Table IV).

To make two of these examples concrete, non-uniqueness is quite easy to see for both Laplacian Eigenmaps and Graph Factorization (with objectives (3.3) and (3.4), respectively). Indeed, in Laplacian Eigenmaps, it is easy to see that if $\{z_i^*\}_{i\in V}$ is an optimal embedding, then so will be $\{R \cdot z_i^*\}_{i\in V}$, where $R \in \mathbb{R}^{d\times d}$ is a rotation matrix. Similarly, in the case of Graph Factorization, if $\{z_i^*\}_{i\in V}$ is an optimal embedding, then so is $\{Q \cdot z_i^*\}_{i\in V}$, where $Q \in \mathbb{R}^{d\times d}$ is an arbitrary orthogonal matrix. For exactly the same reason, other embeddings in Table II that use inner products (e.g., node2vec) are non-unique. Finally, we note that the above problem arises in the context of structural node label prediction, but *not for link prediction and, possibly, other pairwise classification tasks that depend only on the distance or angle between node embeddings.* This because the latter are not affected by rotation and the other transforms listed above. Indeed, embeddings learned via Prob. (4.7) may work well at predicting edges between two nodes in G_B, even though classifier $g(\cdot, W')$ fails.

B. Graph Transfer Learning via Coupling Penalty. We address the above challenges by training the embeddings of G_A and G_B jointly. We accomplish this by modifying Prob. (4.7) to incorporate a coupling penalty: this coupling penalty enforces that nodes that are structurally similar across graphs receive proximal embeddings. Our method is generic and applies to *all embeddings outlined in Table II,* irrespective of whether shallow or deep functions f are used, or whether inputs x_i represent features or one-hot encodings. Most importantly, our algorithm produces a "soft", non-combinatorial assignment between nodes of the two graphs, which can be computed with convex optimization methods; we neither assume nor compute a bijection between nodes in the two graphs.

Formally, we extend Prob. (4.7) to the following constrained optimization problem:

$$\min_{\substack{W_A,W_B\\W',P}} \mathcal{L}_S(W_A;G_A)+\mathcal{L}_C(W_A,W';y_S,G_A)+\mathcal{L}_S(W_B;G_B)$$
$$+\alpha\|AP-PB\|_2^2+\beta\,\mathrm{tr}\left(P^\top D(W_A,W_B)\right), \quad (4.8a)$$
$$\text{s.t.} \quad P\in\mathbb{R}^{n\times n}, P\mathbf{1}=\mathbf{1}, P^\top\mathbf{1}=\mathbf{1}, P\geq 0, \quad (4.8b)$$

where $\alpha,\beta>0$ are positive regularization parameters, $\mathrm{tr}(P^\top D)=\sum_{i\in V_A,j\in V_B}P_{ij}D_{ij}$ is the element-wise product between matrices $P,D\in\mathbb{R}^{n\times n}$, and $D=D(W_A,W_B)$ is a matrix comprising all the pairwise distances between the embeddings of nodes across the two graphs; that is:

$$D(W_A,W_B)=[D_{ij}]_{i\in V_A,j\in V_B}\in\mathbb{R}^{n\times n}, \text{ where} \quad (4.9a)$$
$$D_{ij}=\|z_i^A-z_j^B\|_2, \quad \forall i\in V_A, j\in V_B, \quad (4.9b)$$
$$z_i^A=f(x_i,W_A), \quad \forall i\in V_A, \text{ and} \quad (4.9c)$$
$$z_j^B=f(x_j,W_B), \quad \forall j\in V_B. \quad (4.9d)$$

Intuitively, Prob. (4.8) jointly determines **(a)** the embeddings of nodes in the two graphs, via parameters $W_A,W_B\in\mathbb{R}^m$, **(b)** the label classifier g, via parameters $W'\in\mathbb{R}^{m'}$, and **(c)** a doubly stochastic matrix $P\in[0,1]^{n\times n}$ that couples the nodes of the two graphs and their embeddings together through the penalty:

$$\mathcal{L}_P(P,W_A,W_B)\equiv\alpha\|AP-PB\|_2^2+\beta\,\mathrm{tr}(P^\top D). \quad (4.10)$$

The first term of this penalty learns a probabilistic mapping between nodes in the two graphs, via the doubly stochastic matrix P. Intuitively, if G_A, G_B are isomorphic, $\|AP-PB\|$ is zero under a mapping P that sends every node in G_A to its image in G_B with probability 1; the double-stochasticity of P, enforced via the constraints (4.8b), relaxes this to probabilistic mappings. The second term enforces that nodes that map to each other have similar embeddings. To see this, note that if $P_{ij}\in[0,1]$ is high for some $i\in V_A$, $j\in V_B$, minimizing the penalty in Eq. (4.10) forces $D_{ij}=\|z_i^A-z_j^B\|_2$ to be low.

Our approach has several advantages. It avoids finding a discrete, exact solution to the graph isomorphism/graph matching problem, which is notoriously hard [26]. The coupling penalty (Eq. (4.10)) is convex, making the optimization w.r.t. P tractable given the node embeddings. The coupling via continuous, smoothly evolving variables P translates to a smooth evolution of neural network weights, which is beneficial in practice during SGD. Finally, as embeddings are fine-tuned, the trace penalty helps discover better stochastic mappings P, as nodes with similar embeddings are mapped to each other. Our solution to Prob. (4.8), discussed next, exploits these properties.

C. Alternating Minimization. We solve Prob. (4.8) via alternating minimization. Denote the combined weights of

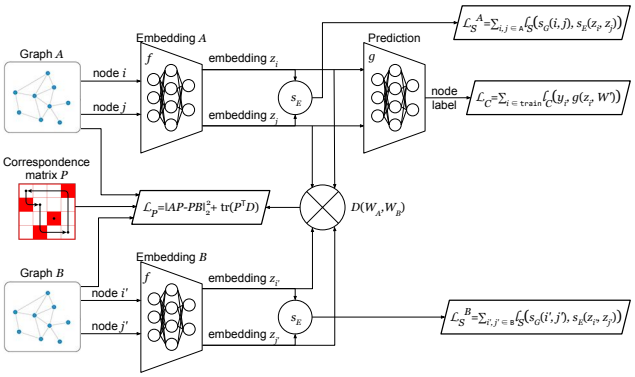


Fig. 2: Schematic portrayal of the proposed framework. Here, $\mathcal{L}_S^A$ and $\mathcal{L}_S^B$ are node embedding penalties (Eq. (3.2)) for graphs G_A, G_B, respectively, L_C is the prediction model loss (Eq. (3.6)), and L_P is the matching penalty (Eq. (4.10)).

the network embeddings f for each graph and the predictor g by $W=(W_A,W_B,W')\in\mathbb{R}^{2m+m'}$. We rewrite (4.8) as

$$\min_{W\in\mathbb{R}^{2m+m'},P\in\mathbb{B}}\mathcal{L}(W,P), \quad (4.11)$$

where $\mathcal{L}:\mathbb{R}^{2m+m'}\times\mathbb{R}^{n\times n}\to\mathbb{R}$ is the aggregate loss (4.8a), and $\mathbb{B}\subseteq\mathbb{R}^{n\times n}$ is the set of doubly stochastic matrices (a.k.a. the *Birkhoff polytope*):

$$\mathbb{B}\triangleq\{P\in[0,1]^{n\times n}: P\mathbf{1}=\mathbf{1}, P^\top\mathbf{1}=\mathbf{1}\}. \quad (4.12)$$

We solve Prob. (4.8) via alternating minimization as follows: at each iteration $k\in\mathbb{N}$ we update weights W and matrix P via

$$W^{(k+1)}=\arg\min_{W\in\mathbb{R}^{2m+m'}}\mathcal{L}(W,P^{(k)}), \quad (4.13a)$$
$$P^{(k+1)}=\arg\min_{P\in\mathbb{B}}\mathcal{L}(W^{(k+1)},P). \quad (4.13b)$$

We give a detailed description of these steps in Appendix A. In short, Eq. (4.13a) can be solved via standard SGD. Eq. (4.13b) is a convex optimization problem, and admits fast implementations via the Frank-Wolfe (FW) algorithm [27] and the Alternating Directions Method of Multipliers (ADMM) [28].

Overall Algorithm and Initialization. A summary of our overall framework for solving Prob. (4.8) is shown in Fig. 2. The embeddings f for both graphs and the predictor network g are neural networks. As the objective is not convex, it is important to start from a good initialization point. To do so, we first compute a matrix P ignoring embeddings (i.e., assuming that $D=0$). Then, we train the embedding and classifier for graph G_A ignoring P (i.e., solving Eq. (3.5) w.r.t W_A) for one epoch; then, using the existing embedding of G_A, and P we train the embedding of G_B (i.e., solving Eq. (4.13a) w.r.t. W_B alone). The remaining alternating minimization proceeds as in Eq. (4.13), with each step as described in Appendix A.

D. Extensions Our approach naturally extends to weighted graphs and graphs of different size.

Graphs of Different Size. Given two graphs G_A, G_B of different size, there are several ways of expanding them with "dummy" nodes such that the new graphs, G'_A and G'_B, have the same number of nodes (see, e.g., [5], [29]). A simple one is to expand graph G_A with $|V_{G_B}|$ dummy nodes and graph G_B with $|V_{G_A}|$ dummy nodes, resulting in two graphs of size $|V_A|+|V_B|$. Dummy nodes are handled in the coupling penalty (4.10) as follows. First, A, B are extended by adding edges of weight $1/2$ between dummy and normal nodes, as well as between dummy nodes: using $1/2$ differentiates such edges from edges in the original graph (that have weight 1), which in turn penalizes maps between dummy and normal nodes. Such maps can be further discouraged via D, by setting the distance between dummy nodes in G_A and non-dummy nodes G_B to a large value (e.g., $100\times$ the largest distance between normal node embeddings), and vice versa, while the distance between dummy nodes is set to 0. Note that dummy nodes have no embeddings, so W updates (Eq. (A.1)) remain unaltered.

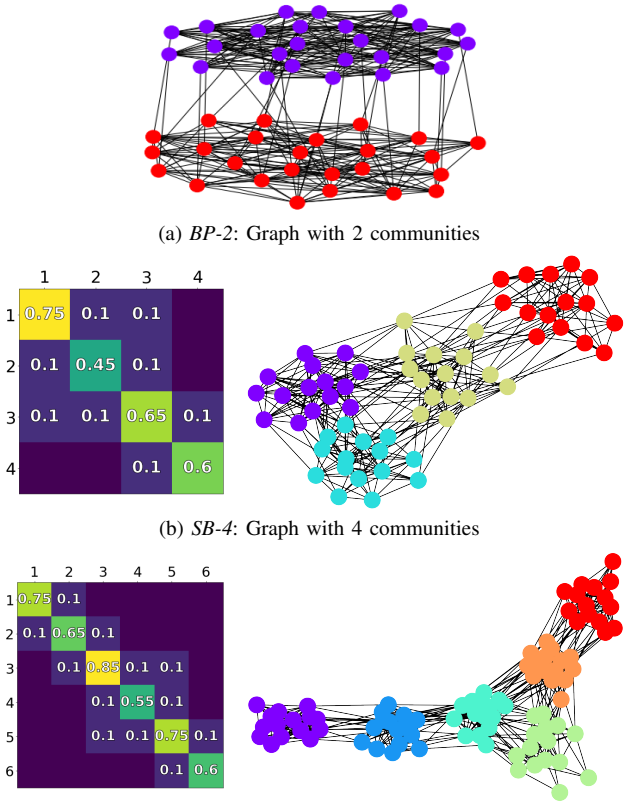
Weighted Graphs. The coupling penalty (Eq. (4.10)) remains the same under weighted graphs, with A, B being now weighted adjacency matrices in $\mathbb{R}^{n\times n}$. Handling weighted graphs thus only requires modifying the embedding functions, taking weights into account when computing graph similarities s_G; all methods outlined in Table II can be appropriately adjusted to do so.

V. Experiments

A. Experimental Setup We use 3 synthetic and 3 real-world datasets, summarized in Table III.

Synthetic datasets. We generate synthetic graphs with $C = 2, 4$ and 6 equal-sized clusters. The graph with 2 clusters, *BP-2*, contains one cluster generated via Erdős-Rényi model [30] $G(25, 0.5)$, while the second cluster is a complete bipartite graph $K_{12,13}$; these two clusters are connected via a bipartite graph with a one-to-one correspondence between nodes from the two clusters (see Fig. 3a). In the 4-cluster and 6-cluster datasets, *SB-4* and *SB-6*, graphs are generated via the stochastic block model. Each cluster is an Erdős-Rényi graph $G(n, p_i^{in})$ ($n = 25$ for the graph with 4 clusters and $n = 20$ for the graph with 6 clusters), and p_i^{in} varies for different clusters i. Clusters are connected as shown in Figs. 3b and 3c, which also provides the inter- and intra- connection probabilities for both SB models.

Real-world datasets. We use 3 real-world graphs: *Zachary Karate Club* (ZKC) [31], *Email* [32], and the *Infectious Disease Transmission Dataset* (IDTD) [33]. We use ground truth cluster labels for *ZKC* as provided. For *Email*, we reorganize ground truth labels provided with the dataset as follows: clusters with fewer than 10 nodes are dissolved, and their nodes are assigned to a cluster with more than 10 nodes by a majority vote across their neighbors. We use the *IDTD* dataset solely for epidemic experiments.



(a) *BP-2*: Graph with 2 communities

(b) *SB-4*: Graph with 4 communities

(c) *SB-6*: Graph with 6 communities

Fig. 3: Synthetic graphs with (a) 2, (b) 4 and (c) 6 communities. Each community is represented as a highly interconnected cluster of nodes. For *SB-4*, *SB-6* graphs, corresponding block adjacency matrices depict probabilities of intra- and inter-cluster connections; shallow inter-clusters connections produce asymmetric structure of a graph.

Labels. We predict two types of labels in our experiments: *clustering* labels and *epidemic spread/influence* labels. Both are structural (i.e., depend on the position of a node in the graph), can be inferred from latent embeddings, and, as we show below, are transferable across graphs. Clustering labels are standard: each node is assigned with a single integer-valued label to one of the clusters it belongs to, and influence labels are generated with `NDlib`'s Independent Cascade model [34]. We generate ground truth influence labels as follows for both synthetic and real-world graphs. We always select a center node, i.e. a node with eccentricity equal to the radius of a graph, to be the *infection seed*. We set the *transition probability*, i.e. the probability that a node will get infected by a neighbor, to $p_{\texttt{infected}} = 0.5$. We then run the independent cascade model [20] 1000 times using `Network Diffusion Library` [34]. For each run, the infection propagation process unfolds from *active* nodes in discrete steps according to the following rule:

a) When node v becomes *active* in step t, it is given

	BP-2	SB-4	SB-6	ZKC	Email	IDTD
$\|V\|$	50	100	120	34	986	788
$\|E\|$	∼331	∼985	∼1028	78	16064	118291
# of clusters	2	4	6	2	28	N/A

TABLE III: Dataset summary.

a single chance to activate each currently inactive, *susceptible*, neighbor w; it succeeds with a transition probability $p_{infected}$. At step $t = 0$, only the *infection seed* is active.

b) If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.

c) If v succeeds, then w will become *active* in step $t+1$, and v itself changes its status to *removed*. Whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds.

The process runs until no more activations are possible. All nodes that remain *susceptible* after the process halts are declared as *healthy*, and the rest of the nodes are considered as *infected*. We use the fraction of times a node was infected as ground truth.

Label Transfer Experiments. All of the datasets, both synthetic and real, contain only one graph G_A. We generate a second graph G_B by randomly permuting G_A via $B = P^\top AP$, where A, B are adjacency matrices of graphs G_A, G_B, respectively, and P is a permutation matrix, i.e., $P \in \mathbb{P} = \mathbb{B} \cap \{0,1\}^{n\times n}$. In the *BP-2* dataset, we additionally remove $\lfloor \frac{|V|}{2} \rfloor + 1$ edges from both graphs. For *SB-4* and *SB-6* datasets, we randomly remove $p \cdot |V|$ edges subsequently adding the same amount of new connections to a given graph G_B. Here, parameter p identifies the percentage of existing edges to be removed and new edges to be added, thus referred to as *perturbation factor*. The effect of this perturbation is studied in Section "Graph Perturbation", with the remaining results on *SB-4* and *SB-6* reported for $p = 0$. Though we train embeddings over the entire graphs G_A, G_B, we train predictor g (Eq. (3.6)) using a subset $S \subset V_A$ containing only 80% of the nodes G_A, selected so that cluster class ratios are preserved. The rest 20% of G_A's nodes are used as a test set (*Test A*). *All* of G_B nodes are used as a separate test set (*Test B*), to validate the success of our transfer learning algorithms. To ensure statistical significance, we repeat all experiments 100 times with random initializations and splits, and report averages and standard deviations of the metrics described below, except for large graphs *Email* and *IDTD*, where we only conduct one experiment.

Metrics. We use *accuracy*, i.e., the fraction of correct predictions $\hat{y}_i$ in the test set, given by $\texttt{ACC} = \frac{\sum_{i\in\texttt{test}} \mathbb{1}_{y_i=\hat{y}_i}}{|\texttt{test}|}$ to assess performance in experiments on clustering labels. For influence labels, we use the *root mean squared error* $\texttt{RMSE} = \sqrt{\frac{\sum_{i\in\texttt{test}}(y_i-\hat{y}_i)^2}{|\texttt{test}|}}$ and *coefficient of determination* [35] given by $\texttt{R}^2 = 1 - \frac{\sum_{i\in\texttt{test}}(y_i-\hat{y}_i)^2}{\sum_{i\in\texttt{test}}(\overline{y}-y_i)^2} \in (-\infty, 1]$, where $\overline{y} = \frac{1}{|\texttt{train}|}\sum_{i\in\texttt{train}} y_i$ is the mean label in a training set `train`.

Architectures and Solvers. We implement *Laplacian Eigenmaps* [2], *Graph Factorization* [1], and *node2vec* [3], whose loss and similarity functions are given in Table II and briefly discussed in Section III-A. We additionally employ *GraphSAGE* [13], a graph neural network node-classification framework, using an open-source implementation distributed by algorithm's authors. For reproducibility purposes, we provide all hyperparameter settings in Appendix B. We perform update Eq. (4.13b) via both exact solution (*optP*) as well as via one iteration of projected gradient descent (*iterP*). We implemented both via the CVX OPT solver, ADMM, and FW. We compared these in efficiency and use the best solver (FW for *optP* and ADMM for *iterP*, respectively) for the rest of our experiments.

Graph Transfer Algorithms. We compare the two versions of our graph transfer learning algorithm (*optP*, using a full constrained optimization solver, and *iterP*, using one iteration of projected gradient descent for Eq. (4.13b), respectively) to the following baselines. First, we implement the naïve algorithm (4.7) that ignores the coupling penalty; we refer to this algorithm as *noP*. We also solve Prob. (4.8) w.r.t W, assuming the true permutation matrix $P \in \mathbb{P}$ mapping G_A to G_B is fixed and entered in the objective of (4.8); we call this algorithm *trueP*. We also construct a doubly stochastic $P \in \mathbb{B}$ that maps every node in one cluster in G_A uniformly to every node in the corresponding cluster in G_B; with this P fixed, we solve again Prob. (4.8) w.r.t. W; we call this algorithm *dsP*. Note that both *trueP* and *dsP* are powerful benchmarks, as they exploit a priori knowledge of the ground truth cluster maps across G_A and G_B. Our code is publicly available.[1]

B. Results

Evaluating Architectures. We first evaluate four embedding algorithms (*Laplacian Eigenmaps* [2], *Graph Factorization* [1], *node2vec* [3], and *GraphSAGE* [13]) to solve the node label prediction Prob. (3.5) on the *SB-4* and *SB-6* datasets. Table IV reports performance on train (*tr*) and test (*tA*) subsets of graph G_A, as well test graph G_B (*tB*), w.r.t. `ACC` and `RMSE` metrics for *clustering* and *influence* labels, respectively, as described in Section V-A. As expected, *all examined embedding methods, including the GNN GraphSAGE, fail to transfer across graphs*. This is evident by the close to random guess accuracy for the classification task and high `RMSE` for the regression task over graph G_B (*tB*) on both datasets. However, for graph G_A, we clearly see that *node2vec* algorithm has superior prediction performance, for both train (*tr*) and test (*tA*) subsets. Thus,

[1] https://github.com/neu-spiral/GraphTransferLearning-NEU

Dataset	Label/ Metric	LE [2]			GF [1]			N2V [3]			GS [13]		
		tr	*tA*	*tB*	*tr*	*tA*	*tB*	*tr*	*tA*	*tB*	*tr*	*tA*	*tB*
SB-4	cl./ACC	0.62	0.53	0.31	0.79	0.69	0.33	**0.99**	**0.95**	0.32	0.96	0.85	0.27
	inf./RMSE	0.13	0.13	0.16	0.10	0.11	0.15	**0.09**	**0.09**	0.15	**0.09**	0.12	0.15
SB-6	cl./ACC	0.59	0.48	0.23	0.54	0.42	0.24	0.97	**0.93**	0.21	**0.98**	0.63	0.17
	inf./RMSE	0.15	0.16	0.16	0.09	0.13	0.19	**0.07**	**0.12**	0.23	0.08	0.13	0.22

TABLE IV: Performance of embedding algorithms w.r.t. solving node label prediction optimization problem (3.5). We evaluate *Laplacian Eigenmaps* (LE), *Graph Factorization* (GF), *node2vec* (N2V) and *GraphSAGE* (GS) methods w.r.t. ACC and RMSE metrics predicting clustering and influence/epidemic spread labels, respectively. We report both training and test accuracy on graph G_A (*tr* and *tA*, respectively), and test accuracy on graph G_B (*tB*), to demonstrate that none of the examined embedding methods succeeds to accurately transfer a learned predictor across two graphs, even when G_A and G_B are isomorphic.

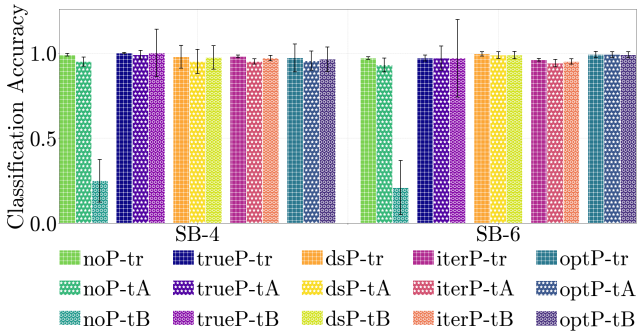


Fig. 4: Classification accuracy, ACC, w.r.t. clustering labels of different transfer learning algorithms (*noP*, *trueP*, *dsP*, *iterP*, *optP*) on two synthetic datasets (*SB-4* and *SB-6*). Each group of 3 ACC values is for training (*tr*) and testing (*tA*) subsets of graph G_A, and testing subset of graph G_B (*tB*). We observe that (a) ACC under naïve scenario (*noP*) is no better than random on *tB*, while (b) ACC when P is learned (both using projected gradient descent (*iterP*) and constrained optimization (*optP*) methods) on *tB* is almost 1, which is on par with *tB* accuracy when true permutation (*trueP*) and doubly stochastic (*dsP*) matrices are used, and on par with train/test accuracies (*tr, tA*) on G_A.

in all further experiments, we focus on transfer learning using this embedding method. More implementation details and model hyperparameters are presented in Appendix B.

Clustering Results. Fig. 4 shows the performance of the five graph transfer algorithms, *noP*, *trueP*, *dsP*, *iterP*, *optP*, described in Section V-A on two synthetic datasets, *SB-4* and *SB-6*. Algorithms are compared w.r.t. transfer test accuracy on G_B (*tB*); for reference purposes, we also show the training and testing accuracy on G_A as well (*tr* and *tA*, respectively). We make three important observations. First, the naïve algorithm (*noP*, Eq. (4.7)) *fails to accurately predict node labels for graph G_B for both topologies, doing almost no better than a random guess*. This is anticipated, for the reasons illustrated in Fig. 1. Second, our two transfer algorithms (*iterP*, *optP*) attain almost the same test accuracy on G_B (*tB*) as in G_A (*tA*): this indicates that the classifier trained on G_A is *successfully transferred* to G_B. Finally, our two transfer methods perform equally well as the powerful benchmarks (*trueP*, *dsP*), that have full access

Dataset	*noP*			*iterP*		
	tr	*tA*	*tB*	*tr*	*tA*	*tB*
BP-2	0.99	0.99	0.53	0.99	0.99	0.97
SB-4	0.99	0.95	0.32	0.98	0.95	0.97
SB-6	0.97	0.93	0.21	0.96	0.94	0.95
ZKC	1.0	0.85	0.5	0.98	0.88	0.96
Email	0.52	0.44	0.02	0.55	0.48	0.49

TABLE V: Classification label accuracy, ACC, on *BP-2, SB-4, SB-6, ZKC,* and *Email* datasets for the *noP* and *iterP* transfer algorithms. We report ACC on training (*tr*) and testing (*tA*) sets of G_A, as well as on the test set of graph G_B (*tB*); *iterP* significantly outperforms *noP* on *tB*.

to the ground truth mappings, yielding accuracies that are comparable to both training (*tr*) and test (*tA*) accuracies observed on G_A.

Table V presents the accuracy for naïve (*noP*) and projected gradient descent (*iterP*) graph transfer algorithms on the *BP-2*, *SB-4*, *SB-6*, *ZKC*, *Email* datasets. Our earlier observations carry over to these graphs as well: *noP* fails to transfer across graphs, yielding low ACC on *tB*, no better than a random guess. On the other hand *iterP* universally performs as well on G_B (*tB*) as on G_A (*tA*). We note that these observations persist on *BP-2*, where graphs G_A and G_B *are not* isomorphic. We observe also that clusters are harder to learn on *Email* (on both G_A and G_B), but the accuracy is considerably better than random guess ($1/28 \approx 0.04$, for 28 clusters); moreover, transfer accuracy (0.49 on *tB*) is comparable to both train and test accuracy on G_A (0.55 and 0.48, respectively), indicating that the poorer performance is inherent to the embedding method and the trained classifier, as opposed to the transfer method.

Epidemic Spread Results. Table VI shows the prediction RMSE and R^2 under *noP* and *iterP* transfer algorithms on *SB-4*, *SB-6*, *ZCK*, *Email*, *IDTD* datasets. We also show the training RMSE (*tr*) as a baseline for comparison purposes. Our observations align perfectly with our earlier clustering results; test RMSE and R^2 on G_B (*tB*) indicate that *noP* fails to transfer, being sometimes worse than predicting based on the training mean ($R^2 < 0$), while prediction on G_B under *iterP* is almost as good as prediction on G_A, sometimes even better (e.g., for *SB-6* and *Email*).

Graph Perturbation. Fig. 5 illustrates results on the *SB-4* and *SB-6* datasets obtained for different percentage of perturbed edges. Here, we use results on graph G_A, *tA*, which does not have any edges removed or added, and results on graph G_B obtained with naïve method, *noP*, as upper and lower bounds when assessing the influence of the amount of perturbed edges on *tB* prediction performance. From all four plots, we can observe a consistent behavior: performance on both clustering and regression tasks remain largely unaffected when the perturbation factor does not exceed 10% (recall that this corresponds to 10% of edges removed and the same amount of new edges added): up until this level, performance is close to *tA* and *trueP*

Dataset	*noP*			*iterP*		
	tr	*tA*	*tB*	*tr*	*tA*	*tB*
	RMSE	RMSE/R^2	RMSE/R^2	RMSE	RMSE/R^2	RMSE/R^2
SB-4	0.09	0.09/0.29	0.15/–1.49	0.10	0.10/0.38	0.11/0.14
SB-6	0.07	0.12/0.42	0.23/–1.91	0.07	0.11/0.37	0.08/0.65
ZKC	0.09	0.09/0.49	0.26/–2.17	0.10	0.11/0.48	0.11/0.45
Email	0.08	0.10/0.22	0.20/–1.66	0.07	0.08/0.23	0.08/0.32
IDTD	0.10	0.10/0.41	0.17/–3.44	0.10	0.10/0.25	0.10/0.16

TABLE VI: Influence/epidemic spread label prediction performance of *noP* and *iterP* transfer learning algorithms on *SB-4, SB-6, ZKC, Email* and *IDTD* datasets. We compare prediction performance on training (*tr*) and testing (*tA*) sets on G_A, and the test set graph G_B (*tB*), w.r.t RMSE (the lower the better) and R^2 (the higher the better); note that the latter only applies to test sets (*tA*, *tB*). We observe that prediction accuracy fails to transfer to G_B under *noP*, even attaining negative R^2 values. In contrast, *iterP* successfully transfers labels, with a predictive power that is comparable to the one over G_A (*tA*).

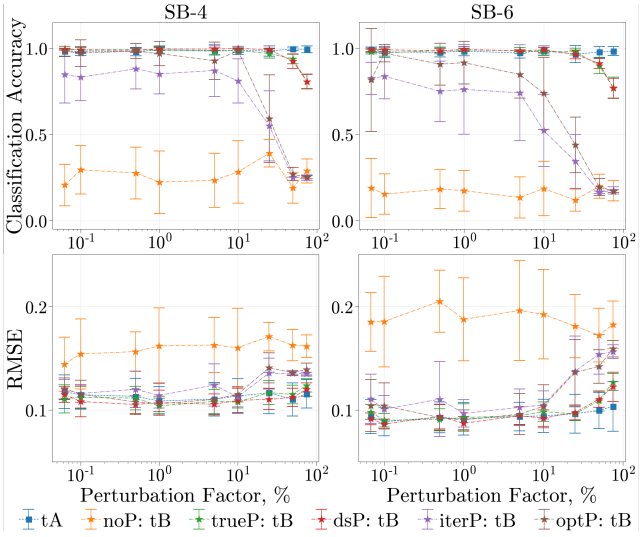


Fig. 5: Effects of edge perturbations between G_A and G_B on the label prediction performance (in ACC and RMSE for clustering and influence labels, respectively) studied on synthetic datasets, *SB-4* and *SB-6*. Transferability is possible even with a 25% perturbation factor, with almost no impact in the $< 10\%$ range.

performance. A degradation happens beyond this point; however, some level of transferability is possible even with a 25% perturbation factor (prediction *tB* for both $iterP$ and $optP$ scenarios is still better than for noP scenario).

VI. Conclusion

Our work offers strong evidence that structural labels can be successfully transferred across graphs using embeddings. This can have important implications, such as learning epidemics on one graph and transferring this knowledge on another. Exploring this on real epidemics is an exciting direction. Accelerating our methods, and scaling them to larger graphs, is an important open problem. The invariance of embeddings to rotations and orthogonal matrices suggests optimizations in which matrix P is orthogonal, rather than doubly stochastic; exploring efficient algorithms for such constraints is also an interesting future direction.

Acknowledgment

The authors gratefully acknowledge support by the National Science Foundation (grants IIS-1741197, CCF-1750539) and Google via GCP credit support.

References

[1] A. Ahmed *et al.*, "Distributed large-scale natural graph factorization," in *TheWebConf*, 2013, pp. 37–48.

[2] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[3] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE TKDE*, vol. 22, no. 10, pp. 1345–1359, 2009.

[5] J. Bento and S. Ioannidis, "A family of tractable graph metrics," *Applied Network Science*, vol. 4, p. 107, 2019.

[6] G. Birkhoff, "Tres observaciones sobre el algebra lineal [three observations on linear algebra]," *Univ. Nac. Tucuman, Ser. A*, vol. 5, pp. 147–151, 1946.

[7] S. Cao *et al.*, "Grarep: Learning graph representations with global structural information," in *ICIKM*, 2015.

[8] W. L. Hamilton *et al.*, "Representation learning on graphs: methods and applications," *Data Engineering Bulletin*, 2017.

[9] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[10] H. Cai *et al.*, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE TKDE*, vol. 30, no. 9, pp. 1616–1637, 2018.

[11] F. Scarselli *et al.*, "The graph neural network model," *IEEE T. on Neural Networks*, vol. 20, pp. 61–80, 2008.

[12] Y. Li *et al.*, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[13] W. L. Hamilton *et al.*, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[14] Z. Lu *et al.*, "Source free transfer learning for text classification," in *AAAI*, 2014, pp. 122–128.

[15] J. Lee *et al.*, "Transfer learning for deep learning on graph-structured data," in *AAAI*, 2017.

[16] K. Gong *et al.*, "Graphonomy: Universal human parsing via graph transfer learning," in *ICCV*, 2019.

[17] S. Verma and Z.-L. Zhang, "Learning universal graph neural network embeddings with aid of transfer learning," *arXiv preprint arXiv:1909.10086*, 2019.

[18] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *GBR*, 2011, pp. 102–111.

[19] D. Koutra *et al.*, "Deltacon: A principled massive-graph similarity function," in *SDM*, 2013, pp. 162–170.

[20] D. Kempe *et al.*, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.

[21] S. Myers and J. Leskovec, "On the convexity of latent social network inference," in *NeurIPS*, 2010.

[22] M. Gomez-Rodriguez *et al.*, "Inferring networks of diffusion and influence," *ACM TKDD*, vol. 5, 2012.

[23] B. Abrahao *et al.*, "Trace complexity of network inference," in *KDD*, 2013, pp. 491–499.

[24] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model." in *AISTATS*, 2005.
[25] T. Mikolov *et al.*, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.
[26] L. Babai, "Graph isomorphism in quasipolynomial time," in *STOC*, 2016, pp. 684–697.
[27] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *NRL*, vol. 3, no. 1-2, pp. 95–110, 1956.
[28] S. Boyd *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, 2011.
[29] S. Gold *et al.*, "Softmax to softassign: Neural network algorithms for combinatorial optimization," *Journal of Artificial Neural Networks*, vol. 2, pp. 381–399, 1996.
[30] P. Erdös and A. Rényi, "On random graphs i," *Publicationes Mathematicae Debrecen*, vol. 6, 1959.
[31] W. W. Zachary, "An information flow model for conflict and fission in small groups," *J. Anthropol. Res.*, 1977.
[32] J. Leskovec *et al.*, "Graph evolution: Densification and shrinking diameters," *ACM TKDD*, vol. 1, 2007.
[33] M. Salathé *et al.*, "A high-resolution human contact network for infectious disease transmission," *PNAS*, vol. 107, no. 51, pp. 22 020–22 025, 2010.
[34] G. Rossetti *et al.*, "Ndlib: Studying network diffusion dynamics," in *DSAA*, 2017, pp. 155–164.
[35] S. Wright, "Correlation and causation," *Journal of Agricultural Research*, vol. 20, pp. 557–585, 1921.
[36] M. Andersen *et al.*, "Interior-point methods for large-scale cone programming," *Optimization for machine learning*, vol. 5583, 2011.

A. Alternating Minimization

Updating W. Given P, minimizing $\mathcal{L}$ w.r.t. W becomes:

$$\min_{W_A,W_B,W'} \mathcal{L}_S(W_A;G_A) + \mathcal{L}_C(W_A,W';y_S,G_A) + \mathcal{L}_S(W_B;G_B) + \beta\,\mathrm{tr}(P^\top D(W_A,W_B)). \tag{A.1}$$

This is almost the naïve problem formulation in Eq. (4.7) save for the linear trace term $\mathrm{tr}(P^\top D)$, that indeed depends on the embeddings via Eq. (4.9b). We minimize this objective via stochastic gradient descent (SGD) w.r.t. the weights $W = (W_A, W_B, W')$. In practice, we only run a single epoch of SGD per iteration before switching to optimizing P, rather than executing SGD until convergence.

Updating P. Given W and, thereby, embeddings z_i^A, $i \in V_A$, and z_j^B, $j \in V_B$, Eq. (4.13b) amounts to:

$$\min_P \quad \mathcal{L}_P(P) = \|AP - PB\|_2 + \beta\,\mathrm{tr}(P^\top D), \tag{A.2a}$$

$$\text{s.t.} \quad P \in \mathbb{B}, \tag{A.2b}$$

where $D = D(W_A, W_B)$ is fully determined by the (fixed) embeddings. This is a convex optimization problem and can thus be solved via standard optimization toolboxes, such as, e.g., CVX OPT [36]. Nevertheless, because Prob. (A.2) is constrained over the Birkoff polytope, it can be solved efficiently via, e.g., the Frank-Wolfe (FW) algorithm [27] and the Alternating Directions Method of Multipliers (ADMM) [28]. We note that FW and ADMM are generally faster than generic solvers in our setting (see also Section V). Though an optimal P can be obtained efficiently through the algorithms discussed above, combining it with stochastic gradient descent steps used to update W has some drawbacks. In particular, different steps may oscillate across different values of P; this, combined with the non-convexity of the objective (A.1) may hinder the convergence of alternating minimization (4.13). For this reason, we also consider the following alternative for updating P in (4.13b). Rather than solving Prob. (A.2), we execute one step of projected gradient descent, instead:

$$P^{(k+1)} = \Pi_{\mathbb{B}}(P^k - \gamma \nabla_P \mathcal{L}_P(P^{(k)})), \tag{A.3}$$

where $\Pi_{\mathbb{B}}$ is the orthogonal projection to the Birkhoff polytope $\mathbb{B}$. This projection involves a quadratic objective subject to Birkhoff constraints; it can again be solved via FW, ADMM, or a standard solver such as CVX OPT.

B. Implementation Details

We implement our framework on `Python 3.6`, using the `Keras 2.2` neural network interface with `TensorFlow 1.10` backend.

Node embedding. The *node2vec* embedding algorithm is deployed with the following parameters: 20 random walks of length 10 are generated for each explored node with the window size equal to 4, return parameter $p = 0.25$ and in-out parameter $q = 4$, and negative sampling with $n = 5$. For *Laplacian Eigenmaps* and *Graph Factorization* algorithms, we use default parameters proposed by authors. For *GraphSAGE* method, we use the official implementation by algorithm's authors with default parameters, except for batch size: it is set to 4 in all of our experiments, due to small size of tested graphs.

Label prediction. In order to ensure the adequate minimization of the label prediction loss (Eq. (3.6)), we design the prediction branch of the framework to consist of 7 fully-connected hidden layers when learning node labels of the *ZKC*, *Email* and *IDTD* datasets. A sole fully-connected hidden layer was exploited in the branch's design when we trained a framework on synthetic datasets *BP-2*, *SB-4* and *SB-6*. Each hidden fully-connected layer contains 10 neurons with a hyperbolic tangent activation function. For each dataset, we solve the *graph transfer learning* optimization problem (4.8) with a stochastic gradient descent optimizer with Nesterov momentum and learning rate $\eta = 0.025$. Regularization parameters α, β, employed in the coupling penalty (Eq. (4.10)), are both set to $\alpha = \beta = 1$. The proposed framework is trained till convergence on the training subset. The convergence is declared when the *early stopping* criterion with the patience equal to 5 epochs is met. All stated parameter values were selected through the exploration of the corresponding parameter spaces.