Knowledge and Information Systems
https://doi.org/10.1007/s10115-022-01782-6

REGULAR PAPER

Graph transfer learning

Andrey Gritsenko[1] · Kimia Shayestehfard[1] · Yuan Guo[1] · Armin Moharrer[1] · Jennifer Dy[1] · Stratis Ioannidis[1]

Received: 26 January 2022 / Revised: 17 October 2022 / Accepted: 23 October 2022

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.

Keywords Graph mining · Transfer learning · Graph distance

1 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

✉ Andrey Gritsenko
agritsenko@ece.neu.edu

Kimia Shayestehfard
kshayestehfard@ece.neu.edu

Yuan Guo
yuanee20@ece.neu.edu

Armin Moharrer
amoharrer@ece.neu.edu

Jennifer Dy
jdy@ece.neu.edu

Stratis Ioannidis
ioannidis@ece.neu.edu

[1] Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA

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 infection probabilities and community membership, 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 and pagerank scores, are all properties that depend on the relative position (w.r.t. clusters, weakly connected components, centrality,...) 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 Sect. 4.1, and demonstrate it also experimentally in Sect. 6.2. 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 (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.

The rest of the paper is organized as follows. We review the previous work related to problems on graph and node embeddings, graph distances and transfer learning on graphs in Sect. 2. In Sect. 3, we focus on node embedding and node label prediction tasks that serve as the backbone of our framework, which we introduce in Sect. 4 and elaborate in Sect. 5, particularizing the options for both exact and inexact solutions. We briefly discuss two natural extensions of the framework: to graphs of different size and weighted graphs, in Sect. 5.4. We present our experiments in Sect. 6, and finally conclude in Sect. 7.

2 Related work

2.1 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 Sect. 3, following the unifying framework of Hamilton et al. [8] (Table 1). 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 2. 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. Aggregation methods vary from merely averaging of neighbor information [8] to more sophisticated aggregation functions [13, 14] to spectral convolutions on graph nodes [15]. Gated graph neural networks [12] propose a solution to vanishing/exploding gradients that allows up to 20 layers in GNNs, by adapting recurrent neural network techniques. Variational variants of GNNs also exist, e.g., [16]. 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 Sect. 4.1 are pertinent to all these methods, and are exacerbated by deep models, as non-convexity increases the multiplicity of local minima.

2.2 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, 17]. A quintessential example is transferring a text classifier from language to another [18–20]. Transfer learning has been applied to graphs only recently; all current work however [21–23] considers classifying (and transferring labels across) graphs, as opposed to nodes. Stone et al. [24, 25] measure similarity between rule graphs and then transfer a value function from one graph to another. Other explored ways of transferring knowledge between graphs include co-factorization [26, 27], learning graph representations via GNNs and mapping them via transfer matrix [22, 23], and performing classic transfer learning on graphs representations in the spectral

Table 1 Summary of notation

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 $v_i^A \in V_A$
$\mathtt{ACC}$	Classification accuracy
$\mathtt{RMSE}$	Root-mean-square error
$\mathtt{R}^2$	Coefficient of determination
$\mathcal{L}_S$	Embedding loss—Eq. (4b)
$\mathcal{L}_C$	Classification loss—Eq. (8)
$\mathcal{L}_P$	Penalty function—Eq. (15)
$\mathcal{L}$	Aggregate loss—Eq. (12a)
W, W_A, W_B, W'	Neural network weights
P	Doubly stochastic matrix
$\mathbb{B}$	Birkhoff polytope—Eq. (18)
$\mathbb{P}$	Set of permutation matrices

domain [21]. Regardless of different ways used for knowledge transfer, the main common feature across all the above works is that they consider classifying (and transferring labels of) graphs, as opposed to nodes. To the best of our knowledge, we are the first to tackle transferring structural node labels between graphs.

2.3 Graph distances and graph matching

Graph distance scores find applications in varied fields such as image processing [28], chemistry [29, 30] and social network analysis [31, 32], to name a few. Classic graph distance examples include the edit distance [33, 34], the maximum common subgraph distance [35, 36], the chemical distance [30], and the Chartrand–Kubiki–Shultz (CKS) distance [37]. Jain [38] proposes an extension of the chemical distance [30] that incorporates edge attributes but is limited to the Frobenius norm. The reaction distance devised by Koca et al. [39] is also directly related to the chemical distance [30] when edits are limited to edge additions and deletions. All six [30, 34, 36–39] are metrics but are hard to compute. Moreover, it is not immediately clear how to relax [38, 39] to attain tractability. 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 and can be seen as convex relaxations of the chemical distance between two graphs. 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.

A straightforward approach to induce a metric over unlabeled graphs is to embed graphs in a common metric space and then measure the distance between these embeddings. Riesen et al. [40, 41] embed graphs into real vectors by computing their edit distances to a set of *prototype* graphs. The same embedding approach is also used to compute a median of graphs [42]. Other works [43–45] map graphs to spaces determined by their spectral decomposition. Although all described methods satisfy metric properties, the resulting embeddings do not capture the full graph structure, and are thus not as discriminative as the metrics proposed in [5] and incorporated in our current work.

Graph matching (also sometimes referenced as graph alignment in literature) is inherently related to graph distance computations, as it is often a preprocessing step toward computing graph distances. The literature on graph matching heuristics is vast (see, e.g., [28, 32, 46–48]). Most are tractable, but distances induced by these mappings do not satisfy the metric property [5]. Nevertheless, several graph matching methods are related to the approach we take. Heimann [49] proposes an embedding-based graph matching that maps nodes with similar node degree or structural connectivity to each other with high probability. Chen [50] uses non-convex alternating optimization methods to match the nodes with embedding-based nearest-neighbor search. By using objectives that take into account distance in the embedding space, these methods are highly related to the trace penalty that Bento and Ioannidis [5] proposed that we also employ here.

2.4 Epidemic learning

The seminal paper by Kempe et al. [51] has motivated learning the parameters of an epidemic spread (e.g., [52–54]). Typically, this is done via maximum likelihood estimation over a generative model, e.g., the independent cascades (IC) or linear threshold (LT) models [51]. Though methodologically quite different from the approach we take here, these estimation methods can be used for the influence cascade prediction task we study as a motivating example, and that we also explore via our experiments in Sect. 6. In particular, the parameters of a propagation model can be trained on one graph, and then used to predict propagation on another graph with similar structural characteristics or known matched correspondences between nodes. In contrast, our approach can be used to perform this task *without any parameter inference.* In short, following our graph transfer learning method, we can learn how cascades behave in one graph and then transfer this knowledge directly to another graph. We thus avoid intermediate parameter inference and modeling assumptions (such presuming that the IC or LT generative models hold during the cascade) that may not hold in practice and introduce model bias in the prediction process.

3 Background

3.1 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 and link prediction. 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

Table 2 Different embedding methods expressed in the unifying framework of Hamilton at al. [8]

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

Here, $s_G(i,j)$ and $s_E(z_i,z_j)$ denote topological and embedding similarity, respectively, between nodes i and j, and for *node2vec*, $p(i|j)$ is the probability of visiting node j on a fixed-length random walk from node i

$f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^d$, where $d \ll n$, mapping nodes to d dimensional vectors, that is,

$$z_i = f(x_i, W) \in \mathbb{R}^d \tag{1}$$

is the embedding z_i of node $i \in V$, and $W \in \mathbb{R}^m$, for some $m \in \mathbb{N}$, are weights parameterizing the embedding function. For example, f could be a neural network with weights W or an affine (shallow) function. 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 between nodes with the following formulation

$$s_G : V \times V \to \mathbb{R} \tag{2}$$

as well as the following *embedding similarity* function between embeddings

$$s_E : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \tag{3}$$

the node embedding task can be formulated via the following minimization problem:

$$\min_{W\in\mathbb{R}^m} \mathcal{L}_S(W; G), \tag{4a}$$

where

$$\mathcal{L}_S(W; G) = \sum_{i,j\in V} \ell_S(s_G(i,j), s_E(z_i,z_j)), \text{ and} \tag{4b}$$

$$z_i = f(x_i, W), \quad \forall i \in V, \tag{4c}$$

and $\ell_S : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is an appropriately defined loss function. Typically, Problem (4) is solved via stochastic gradient descent over the nodes, although techniques like hierarchical softmax [55] and negative sampling [56] can be incorporated to accelerate computations.

Examples. The topological similarity s_G (2) can be 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 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 2 (see also [8]). For example, *Laplacian Eigenmaps* [2] couple Euclidean distance with a product loss, yielding:

$$\mathcal{L}_S(W; G) = \sum_{i,j\in V} \|z_i - z_j\|_2^2 \cdot s_G(i,j), \tag{5}$$

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

$$\mathcal{L}_S(W, G) = \sum_{i,j \in V} (z_i^\top z_j - s_G(i, j))^2. \tag{6}$$

3.2 Node label prediction

Node embeddings often serve as an intermediate step for downstream supervised learning tasks on graphs, such as community detection and link prediction. 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 Problem (4) as follows:

$$\min_{W \in \mathbb{R}^m, W' \in \mathbb{R}^{m'}} \mathcal{L}_S(W; G) + \mathcal{L}_C(W, W'; y_S, G), \tag{7}$$

where $\mathcal{L}_S(W; G)$ is the similarity loss (4b), while

$$\mathcal{L}_C(W, W'; y_S, G) = \sum_{i \in S} \ell_C\big(y_i, g(z_i, W')\big) \tag{8}$$

is the classification loss with z_i being the embedding (4c) of node i. Here, $\ell_C : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function (such as 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) parameterized 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 (7) can again be solved via stochastic gradient descent, where an epoch iterates over batches node pairs $i, j \in V$ and labeled nodes $i \in S$. Note that in this scenario, W and W' are trained jointly, i.e., over multiple epochs iterating over the two objectives; alternatively, embeddings z_i could be learned first via SGD on W, as in Prob. (4), and subsequently used to train W' with embeddings fixed.

4 Graph transfer learning

4.1 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 and centrality, 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 and $y_i \in \mathbb{R}$ in the case of a regression task. 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 our method to weighted graphs and graphs of unequal size in Sect. 5.4.

A Naïve Solution. The node embedding and node label prediction algorithms we reviewed in Sect. 3 give a possible simple solution to the graph transfer learning problem. First, a discriminative embedding is trained on graph G_A, by solving Prob. (7): 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. (4) on G_B alone. Finally, the predictive model $g(\cdot, W')$

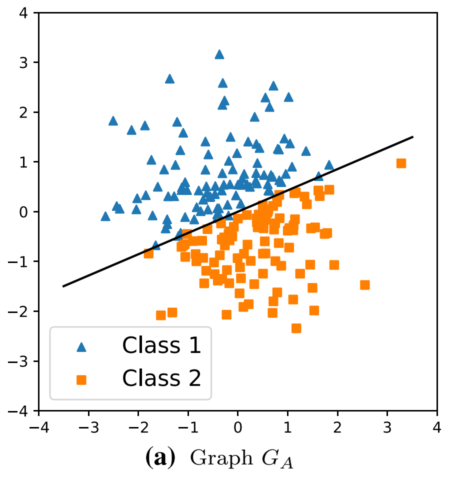


Fig. 1 Example of two isomorphic embeddings and the failure to transfer a learned classifier across them. (**a**) Embeddings of a G_A are used to train an almost perfect classifier between two classes. Embeddings of G_B in (**b**) are identical to G_A but subject to a rotation; as a result, the classifier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

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:

$$\min_{W_A, W_B, W'} \mathcal{L}_S(W_A; G_A) + \mathcal{L}_C(W_A, W'; y_S, G_A) + \mathcal{L}_S(W_B; G_B), \tag{9}$$

where $\mathcal{L}_S$, $\mathcal{L}_C$ are given by Eqs. (4b) and (8), respectively. Unfortunately, this approach is bound to fail; we extensively demonstrate this experimentally in Sect. 6.2, and give some intuition as to why this is the case below.

Non-Uniqueness. It can be noticed that Eq. (9) 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 2 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. (4)) 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 shown in Fig. 1: noticeably, 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 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 Sect. 6.2, 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 4).

To make two of these examples concrete, non-uniqueness for both *Laplacian Eigenmaps* and *Graph Factorization* (with objectives (5) and (6), respectively) is quite easy to see, as demonstrated below. Indeed, if $\{z_i^*\}_{i\in V}$ is an optimal embedding obtained by *Laplacian Eigenmaps*, 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 2 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 is because the latter are not affected by rotation and the other transforms listed above. Indeed, embeddings learned via Prob. (9) may work well at predicting edges between two nodes in G_B, even though classifier $g(\cdot, W')$ fails.

4.2 Graph transfer learning via coupling penalty

Graph Matching. To transfer graph embeddings across graphs, one would need to know the correspondence between the nodes in the two networks. This is generally known as the graph matching problem [28, 57, 58], and several algorithms (mostly heuristics) exist for solving it. Assuming that graphs have the same size, we would like to find a permutation that maps nodes from graph G_A to G_B that minimizes edge discrepancies. Formally, let us denote the set of permutation matrices by

$$\mathbb{P} = \{P \in \{0, 1\}^{n\times n} : P\mathbf{1} = \mathbf{1}, P^\top \mathbf{1} = \mathbf{1}\} \tag{10}$$

and the Frobenius matrix norm by $\|\cdot\|$, then we seek P such that:

$$\min_{P\in\mathbb{P}} \|AP - PB\|, \tag{11}$$

where A and B are the corresponding adjacency matrices for G_A and G_B, respectively. The permutation P indeed then captures the node mapping between the two graphs that minimizes edge discrepancies; in particular, the minimum value (11) is zero if and only if the two graphs are isomorphic. Unfortunately, no poly-time algorithm is known for solving (11) [59], though research on approximation algorithms and heuristics is vast (see, e.g., [28]).

Coupling Penalty. Rather than solving (11), we focus on its convex relaxation, which was proposed by Bento and Ioannidis in [5]. We incorporate this relaxed version of (11) into the optimization problem (9). In particular, we propose solving the following optimization problem:

$$\min_{\substack{W_A, W_B \\ W', P}} \mathcal{L}_S(W_A; G_A) + \mathcal{L}_C(W_A, W'; y_S, G_A) + \mathcal{L}_S(W_B; G_B)$$

$$+ \alpha \|AP - PB\|_2^2 + \beta \operatorname{tr}\left(P^\top D(W_A, W_B)\right), \tag{12a}$$

$$\text{s.t.} \quad P \in \mathbb{R}^{n \times n}, P\mathbf{1} = \mathbf{1}, P^\top \mathbf{1} = \mathbf{1}, P \geq 0, \tag{12b}$$

where $\alpha, \beta > 0$ are positive regularization parameters,

$$\operatorname{tr}(P^\top D) = \sum_{\substack{i \in V_A \\ j \in V_B}} P_{ij} D_{ij} \tag{13}$$

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:

$$D(W_A, W_B) = [D_{ij}]_{i \in V_A, j \in V_B} \in \mathbb{R}^{n \times n}, \text{ where} \tag{14a}$$

$$D_{ij} = \|z_i^A - z_j^B\|_2, \quad \forall i \in V_A, j \in V_B, \tag{14b}$$

$$z_i^A = f(x_i, W_A), \quad \forall i \in V_A, \text{ and} \tag{14c}$$

$$z_j^B = f(x_j, W_B), \quad \forall j \in V_B. \tag{14d}$$

Intuitively, Prob. (12) 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:

$$\mathcal{L}_P(P, W_A, W_B) \equiv \alpha \|AP - PB\|_2^2 + \beta \operatorname{tr}(P^\top D). \tag{15}$$

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 (12b), relaxes this to probabilistic mappings. The second term enforces nodes that map to each other to have similar embeddings. Indeed, if $P_{ij} \in [0, 1]$ is high for some $i \in V_A$, $j \in V_B$, minimizing the penalty in Eq. (15) 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 [59]. The coupling penalty (15) 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. (12), discussed next, exploits these properties.

5 An alternating minimization algorithm for solving the graph transfer learning problem

We solve Prob. (12) via alternating minimization. Denote the combined weights of 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 (12) as

$$\min_{W \in \mathbb{R}^{2m+m'}, P \in \mathbb{B}} \mathcal{L}(W, P), \tag{16}$$

where

$$\mathcal{L} : \mathbb{R}^{2m+m'} \times \mathbb{R}^{n \times n} \to \mathbb{R} \tag{17}$$

is the aggregate loss (12a), and $\mathbb{B} \subseteq \mathbb{R}^{n \times n}$ is the set of doubly stochastic matrices (a.k.a. the *Birkhoff polytope*):

$$\mathbb{B} \triangleq \{P \in [0, 1]^{n \times n} : P\mathbf{1} = \mathbf{1}, P^\top \mathbf{1} = \mathbf{1}\}. \tag{18}$$

We solve Prob. (12) via alternating minimization as follows: at each iteration $k \in \mathbb{N}$, we update weights W and matrix P via

$$W^{(k+1)} = \arg\min_{W \in \mathbb{R}^{2m+m'}} \mathcal{L}(W, P^{(k)}), \tag{19a}$$

$$P^{(k+1)} = \arg\min_{P \in \mathbb{B}} \mathcal{L}(W^{(k+1)}, P). \tag{19b}$$

We describe these two alternating steps in detail below. In short, Eq. (19a) can be solved via standard SGD. Equation (19b) is a convex optimization problem, and admits fast implementations via, e.g., the Frank–Wolfe (FW) algorithm [60] and the alternating directions method of multipliers (ADMM) [61].

5.1 Updating combined weights *W*

Given P, minimizing $\mathcal{L}$ w.r.t. W amounts to the following problem:

$$\min_{\substack{W_A, W_B \\ W'}} \mathcal{L}_S(W_A; G_A) + \mathcal{L}_C(W_A, W'; y_S, G_A) + \mathcal{L}_S(W_B; G_B) + \beta \operatorname{tr}(P^\top D). \tag{20}$$

This is almost identical to the naïve problem formulation (9) save for the linear trace term $\operatorname{tr}(P^\top D)$ that indeed depends on the embeddings via (14b). We thus minimize this objective via stochastic gradient descent (SGD) w.r.t. the weights W. We note that in practice, we only run one epoch of SGD per iteration of (19a) before switching to optimizing P, rather than executing SGD until full convergence.

5.2 Updating graph mapping *P*

5.2.1 An exact solution: constrained optimization over the Birkhoff polytope

Given W and, thereby, embeddings z_i^A, $i \in V_A$, and z_j^B, $j \in V_B$, (19b) amounts to the following problem

$$\min_P \quad \mathcal{L}_P(P) = \alpha \|AP - PB\| + \beta \operatorname{tr}(P^\top D), \tag{21a}$$

$$\text{s.t.} \quad P \in \mathbb{B} \tag{21b}$$

where $D = D(W_A, W_B)$ is fully determined by the (fixed) embeddings and $\mathbb{B}$ is the Birkhoff polytope, given by (18). This is a convex optimization problem and can thus be solved via standard optimization toolboxes, such as CVX OPT [62]. Nevertheless, we can design efficient algorithms tailored to (21) and (19b) precisely because (21) is constrained over the Birkhoff polytope. In particular, Problem (21) can be solved efficiently via the Frank–Wolfe (FW) algorithm [60] and the alternating directions method of multipliers (ADMM) [61]. We describe both in detail below.

Frank–Wolfe (FW) Algorithm. The FW algorithm [60] is an iterative algorithm that solves (21) through a sequence of linear programs (LPs). The algorithm starts from a feasible $P^0 \in \mathbb{B}$, e.g., the identity matrix I, and proceeds with the following iterations $k \in \mathbb{N}$:

$$S^k = \arg\min_{S\in\mathbb{B}} \operatorname{tr}\left(S^\top \cdot \nabla\mathcal{L}_P(P^k)\right), \tag{22a}$$

$$P^{k+1} = (1-\gamma_k)P^k + \gamma_k S^k, \tag{22b}$$

where $\mathcal{L}_P$ is the objective in (21a), and γ_k is a step size set to be, e.g., $\gamma_k = 1/(k+2)$, or determined via line search [63] as follows:

$$\gamma_k = \arg\min_{\gamma_k\in[0,1]} \mathcal{L}_P\left((1-\gamma_k)P^{(k)} + \gamma_k S^{(k)}\right). \tag{23}$$

As L_P is convex, this is guaranteed to converge to an optimal solution under mild conditions [64]. Crucially, as $\mathbb{B}$ is a polytope, (22a) is a linear program; thus, it has an optimal solution that is a vertex of $\mathbb{B}$, by the fundamental theorem of linear programming. By the Birkhoff–von Neuman theorem, the vertices of $\mathbb{B}$ are in fact the permutation matrices. As a result, a solution to (22a) can be found by solving:

$$\arg\min_{P\in\mathbb{P}} \operatorname{tr}\left(S^\top \cdot \nabla\mathcal{L}_P(P^k)\right), \tag{24}$$

instead, where $\mathbb{P}$ is the set of permutation matrices. This is precisely the so-called assignment problem [65], and can be solved in strongly polynomial time via the so-called Hungarian algorithm [65]. As we show in Sect. 6.2.2, using the Frank–Wolfe (22) combined with the Hungarian algorithm to solve assignment problem (22a) has significant computational advantages compared to both generic convex optimization methods for solving (19b), as well as generic LP solvers for solving (22a).

Alternating directions method of multipliers (ADMM). To employ ADMM [61], we first incorporate constraints into the objective of Prob. (21) using the indicator function $\chi_{\mathcal{D}} : \mathbb{R}^{n\times n} \to \{0, \infty\}$, where $\mathcal{D}$ is a convex set and $\chi_{\mathcal{D}}(P) = 0$ if and only if $x \in \mathcal{P}$. Now, we can reformulate Prob. (21) as follows:

$$\begin{aligned} \min \quad & \mathcal{L}_P(P) + \chi_{\mathcal{R}}(P_1) + \chi_{\mathcal{C}}(P_2) && \text{(25a)}\\ \text{s.t.} \quad & P = P_1 && \text{(25b)}\\ & P = P_2, && \text{(25c)} \end{aligned}$$

where

$$\mathcal{R} = \{P \in [0,1]^{n\times n} : P\mathbf{1} = \mathbf{1}\} \tag{26}$$

and

$$\mathcal{C} = \{P \in [0,1]^{n\times n} : P^\top\mathbf{1} = \mathbf{1}\}. \tag{27}$$

Note that $P \in \mathbb{B}$ if and only if $P \in \mathcal{R}$ and $P \in \mathcal{C}$. Problem (25) is well-suited for the ADMM algorithm [61], as it involves linear constraints and two sets of variables, i.e., P and (P_1, P_2).

With regard to the given problem, the steps of ADMM amount to the following:

$$P^{(k+1)} = \arg\min_P \mathcal{L}_P(P) + \frac{\rho}{2}(\|P - P_1^{(k)} + U_1^{(k)}\|^2 + \|P - P_2^{(k)} + U_2^{(k)}\|^2) \tag{28a}$$

$$P_1^{(k+1)} = \arg\min_{P_1 \in \mathcal{R}} \|P^{(k+1)} - P_1 + U_1^{(k)}\|^2 \tag{28b}$$

$$P_2^{(k+1)} = \arg\min_{P_2 \in \mathcal{C}} \|P^{(k+1)} - P_2 + U_2^{(k)}\|^2 \tag{28c}$$

$$U_1^{(k+1)} = U_1^{(k)} + \left(P^{(k+1)} - P_1^{(k+1)}\right) \tag{28d}$$

$$U_2^{(k+1)} = U_2^{(k)} + \left(P^{(k+1)} - P_2^{(k+1)}\right), \tag{28e}$$

where $\rho > 0$ is an ADMM parameter controlling convergence and $U_1, U_2 \in \mathbb{R}^{n\times n}$ are (scaled) dual variables corresponding to (25b) and (25c), respectively. All the steps in (28) can be executed efficiently; (28a) is an unconstrained strongly convex problem which can be solved via, e.g., gradient methods. Problems (28b) and (28c) are projections over the simplex, which can be solved efficiently via strongly polynomial algorithms (see, e.g., Michelot [66]). Finally, dual variable adaptations (28d) and (28e) can be executed in linear time.

5.2.2 An inexact solution: projected gradient descent

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 (20), may hinder the convergence of alternating minimization (19).

For this reason, we consider the following alternative for updating P in (19b). Rather than solving (21) exactly, we execute one step of projected gradient descent, instead:

$$P^{(k+1)} = \Pi_{\mathbb{B}}(P^k - \gamma \nabla_P \mathcal{L}_P(P^{(k)})), \tag{29}$$

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

5.3 Overall algorithm and initialization

A summary of our overall framework for solving Prob. (12) is shown in Fig. 2. The embeddings f for both graphs and the predictor network g are neural networks, and their corresponding parameters can be trained via SGD. The optimization w.r.t. P, appearing in penalty $\mathcal{L}_P$, requires customized solvers like Frank–Wolfe or ADMM, or convex optimization solvers like CVX OPT. 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. (7) 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. (19a) w.r.t. W_B alone). The remaining alternating minimization proceeds as in Eq. (19), with each step as described above.

5.4 Extensions

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

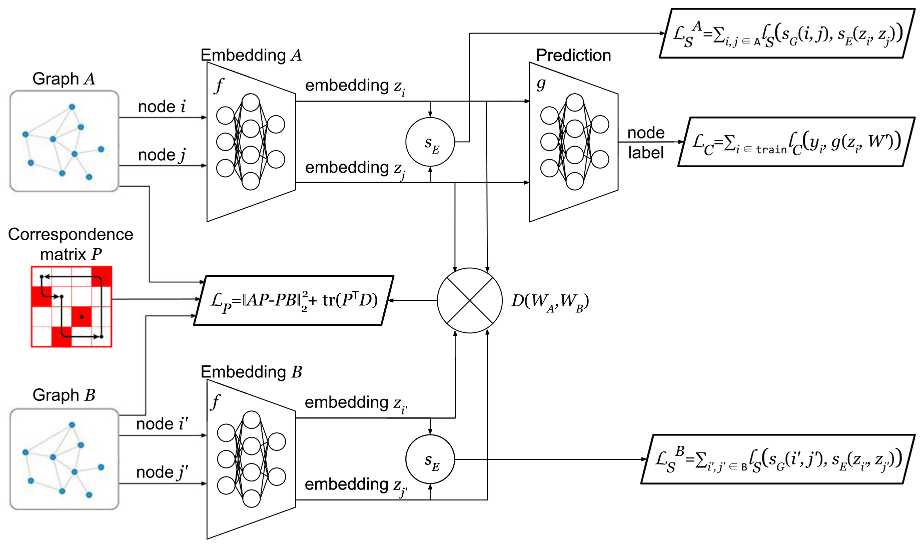

Graph A
Embedding A
f
node i
node j
embedding z_i
embedding z_j
s_E
Prediction
g
node label
$\mathcal{L}_S^A = \sum_{i,j \in A} \ell_S(s_G(i,j), s_E(z_i, z_j))$
$\mathcal{L}_C = \sum_{i \in \text{train}} \ell_C(y_i, g(z_i, W'))$
Correspondence matrix P
$\mathcal{L}_P = \|AP - PB\|_2^2 + \text{tr}(P^\top D)$
$D(W_A, W_B)$
Graph B
Embedding B
f
node i'
node j'
embedding $z_{i'}$
embedding $z_{j'}$
s_E
$\mathcal{L}_S^B = \sum_{i',j' \in B} \ell_S(s_G(i',j'), s_E(z_{i'}, z_{j'}))$

Fig. 2 Schematic portrayal of the proposed framework. Here, $\mathcal{L}_S^A$ and $\mathcal{L}_S^B$ are node embedding penalties (Eq. (4b)) for graphs G_A, G_B, respectively, L_C is the prediction model loss (Eq. (8)), and L_P is the matching penalty (Eq. (15))

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, 67]). 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 (15) 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. (20)) remain unaltered.
Weighted Graphs. The coupling penalty (Eq. (15)) 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 2 can be appropriately adjusted to do so.

6 Experiments

To validate our proposed methodology, we perform a series of experiments on real and synthetic datasets. We provide in detail our experimental setup in this section, followed by description of results in Sect. 6.2.

Table 3 Dataset summary

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

6.1 Experimental setup

6.1.1 Datasets

In our experiments, we use three real-world datasets, which are *Zachary Karate Club* (ZKC) [68], *Email* [69], and *Infectious Disease Transmission Dataset* (IDTD) [70], and three synthetic graphs with $C = 2$, 4 and 6 equal-sized clusters. *Zachary's Karate Club* is a popular dataset for the classification task [14, 71, 72] because it contains two distinct clusters. *Email* is a popular graph [53, 73, 74] with large number of nodes and clusters. *IDTD* is a graph representing close proximity interactions at an American high school, representing 655 students, 73 teachers, 55 staff, and 5 other persons. Nodes represent individuals and edges represent contacts. This dataset is tailored to assessing the performance of our framework in an epidemic spread task. We also constructed synthetic datasets with distinguishable clusters; thus, they are good means to assess the performance and limitations of our proposed framework on the classification task. Details for all six datasets are summarized in Table 3.

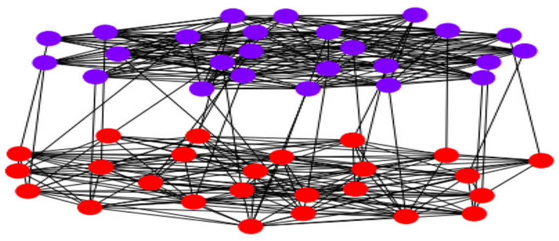
The synthetic graph with two clusters, *BP-2*, contains one cluster generated via Erdős-Rényi model [75] $G(25, 0.5)$, while the second cluster is a complete bipartite graph $K_{12,13}$; these two clusters are connected via a bipartite graph with a one-to-one correspondence between nodes from the two 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 [76]. 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 Fig. 3b and Fig. 3c, which also provide the inter- and intra-connection probabilities for both stochastic block models.

6.1.2 Labels

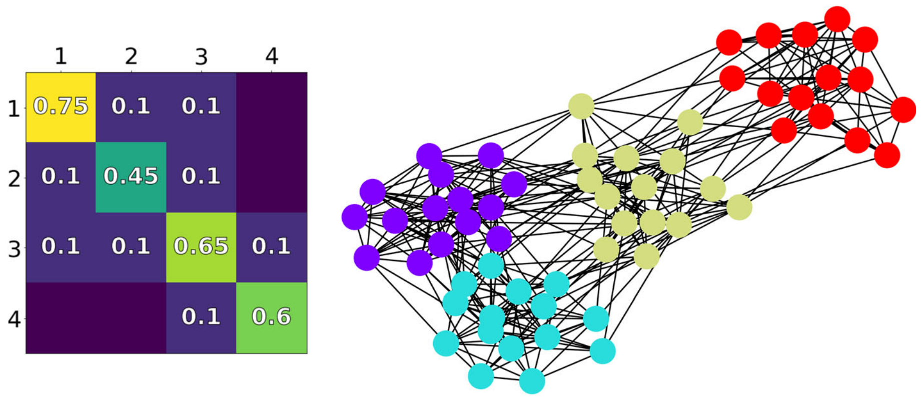
We predict two types of labels in our experiments: *clustering* labels and *epidemic spread/influence* labels. Both are structural (i.e., depend on the position of a node in the graph), can be inferred from latent embeddings, and, as we show below, are transferable across graphs.

Clustering labels are standard: each node is assigned with a single integer-valued label representing a single cluster it belongs to. We use ground truth cluster labels for *ZKC* as provided by Zachary in the original paper [68]. 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.

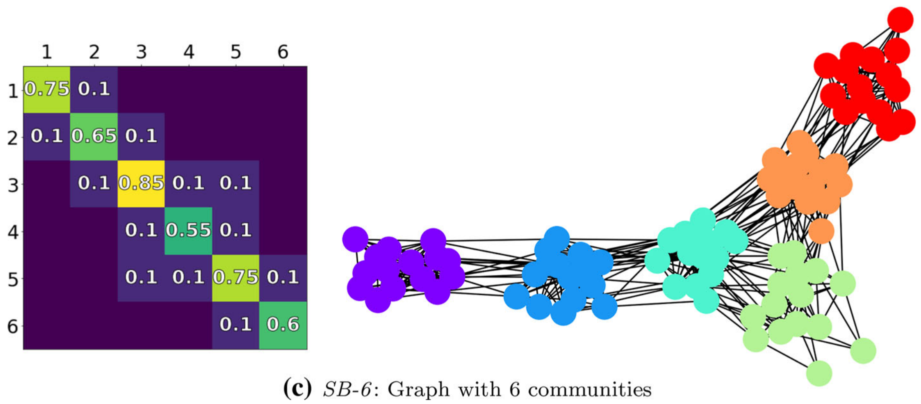
Influence/epidemic spread labels are generated with the Independent Cascade (IC) model [51] from the `Network Diffusion Library (NDlib)` [77] as follows. We first always select a center node, i.e., a node with eccentricity equal to the radius of a graph,



(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 (left) depict probabilities of intra- and inter-cluster connections; shallow inter-clusters connections produce asymmetric structure 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 independent cascades 1000 times to obtain labels. 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 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, and we utilize our proposed framework to regress them for both synthetic and real-world datasets.

6.1.3 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 as follows:

$$B = P^\top A P, \tag{30}$$

where A, B are adjacency matrices of graphs G_A, G_B, respectively, and P is a random permutation matrix, i.e., $P \in \mathbb{P}$ (Eq. (10)).

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 Sect. 6.2.5, 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. (8)) using a subset $S \subset V_A$ containing only 80% of the nodes G_A, selected so that cluster class ratios are preserved. When reporting, the results for this subset are denoted as *tr*. The rest 20% of G_A's nodes are used as a test set (*tA*). *All* of G_B nodes are used as a separate test set (*tB*), 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.

6.1.4 Metrics

To assess performance in experiments on clustering labels, we use *accuracy*, i.e., the fraction of correct predictions $\hat{y}_i$ across all classes in the test set, which is a common metric for classification problems on balanced or slightly imbalanced datasets, given by

$$\texttt{ACC} = \frac{\sum_{i \in \texttt{test}} \mathbb{1}_{y_i = \hat{y}_i}}{|\texttt{test}|} \in [0, 1]. \tag{31}$$

For influence/epidemic spread labels, we use *root-mean-square error*

$$\texttt{RMSE} = \sqrt{\frac{\sum_{i \in \texttt{test}} (y_i - \hat{y}_i)^2}{|\texttt{test}|}} \in [0, \infty) \tag{32}$$

that is one of the most popular general purpose metrics for regression models evaluation. Additionally, we measure the *coefficient of determination* [78, 79], which can be more intuitively informative than RMSE due to its scale invariance.

$$\mathrm{R}^2 = 1 - \frac{\sum_{i \in \mathtt{test}} (y_i - \hat{y}_i)^2}{\sum_{i \in \mathtt{test}} (\overline{y} - y_i)^2} \in (-\infty, 1], \tag{33}$$

where

$$\overline{y} = \frac{1}{|S|} \sum_{i \in S} y_i \tag{34}$$

is the average label in the training set.

6.1.5 Architectures

We implement *Laplacian Eigenmaps* [2], *Graph Factorization* [1], and *node2vec* [3] embedding methods, whose loss and similarity functions are given in Table 2 and briefly discussed in Sect. 3.1. For the *Laplacian Eigenmaps* and *Graph Factorization* algorithms, default parameters proposed by authors were used in all conducted experiments. 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 $q = 4$. Furthermore, negative sampling with $n = 5$ was used in the experiments involving *node2vec* to reduce computation burden. We additionally employ *GraphSAGE* [13], a graph neural network node-classification framework, using an open-source implementation distributed by algorithm's authors.

In order to ensure the adequate minimization of the label prediction loss (Eq. (8)), we design the prediction branch of the framework consisting of seven 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 training a framework on synthetic datasets *BP-2*, *SB-4* and *SB-6*. Each hidden fully connected layer contains ten neurons with hyperbolic tangent activation function applied.

6.1.6 Solvers

We perform update Eq. (19b) 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 Frank–Wolfe. We compare these in efficiency and use the best-performing solver for the rest of our experiments: Frank–Wolfe for *optP* and ADMM for *iterP*, respectively.

We solve the *graph transfer learning* optimization problem (12) 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. (15)), 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.

6.1.7 Graph transfer algorithms and implementation

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. (19b), respectively) to the following baselines. First, we implement the naïve algorithm (9) that ignores the coupling penalty; we refer to this algorithm as *noP*. We also solve Prob. (12) 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 (12); 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. (12) 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.

We implement our proposed framework on `Python 3.6`, using `Keras 2.2` neural network interface with `TensorFlow 1.10` backend. We make our code publicly available.[1]

6.2 Experimental results

6.2.1 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. (7) on the *SB-4* and *SB-6* datasets. Table 4 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 Sect. 6.1.2. 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, *node2vec* algorithm has superior prediction performance for graph G_A, both in train (*tr*) and test (*tA*) subsets. Thus, in all further experiments, we focus on transfer learning using this embedding method.

6.2.2 Solver comparison

Figure 4 illustrates the convergence of four solvers on randomly generated graphs with $n = 100$ nodes for both (a) constrained optimization (21) and (b) orthogonal projection (29). The solvers are ADMM, CVX OPT, Frank–Wolfe with line-search step sizes, and Frank–Wolfe with fixed step sizes.

All methods eventually converge to the same loss. For a straightforward constrained optimization solution (Fig. 4a), Frank–Wolfe algorithm with fixed step size exhibits the fastest convergence; we use it as the default algorithm in the corresponding experiments, *optP*. For the projected gradient descent (Fig. 4b), ADMM converges the fastest; we use it in all upcoming experiments for *iterP*.

6.2.3 Transferring clustering labels

Figure 5 shows the performance of the five graph transfer algorithms, *noP*, *trueP*, *dsP*, *iterP*, *optP*, described in Sect. 6.1.7 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. (9)) *fails to accurately predict*

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

Table 4 Performance of embedding algorithms w.r.t. solving node label prediction optimization problem (7)

Dataset	Label/	LE [2]			GF [1]			N2V [3]			GS [13]		
	Metric	*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

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

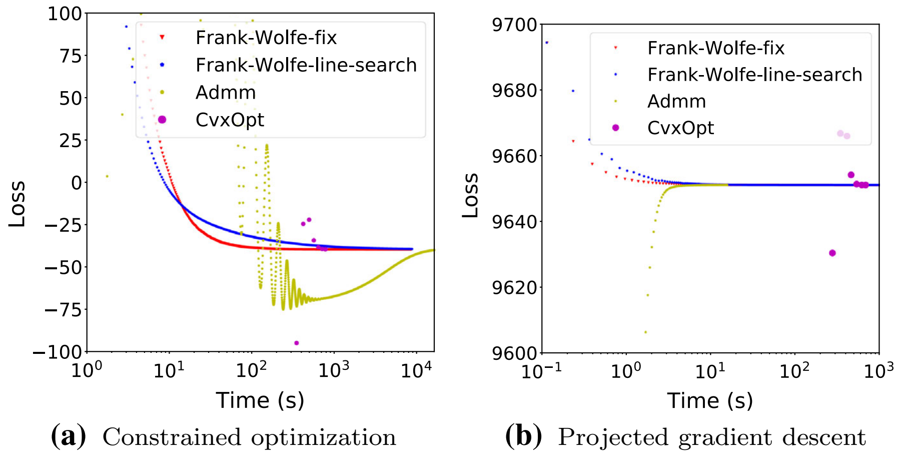


Fig. 4 Convergence rate comparison of solvers used to solve (19b). In the experiment, both the graphs G_A and G_B have $|V_A| = |V_B| = 100$ nodes. Analysis reveals that (**a**) Frank–Wolfe algorithm with fixed step size has the steepest convergence rate finding the solution of straightforward constrained optimization problem (21), and (**b**) projected gradient descent (29) converges faster when implemented via ADMM algorithm

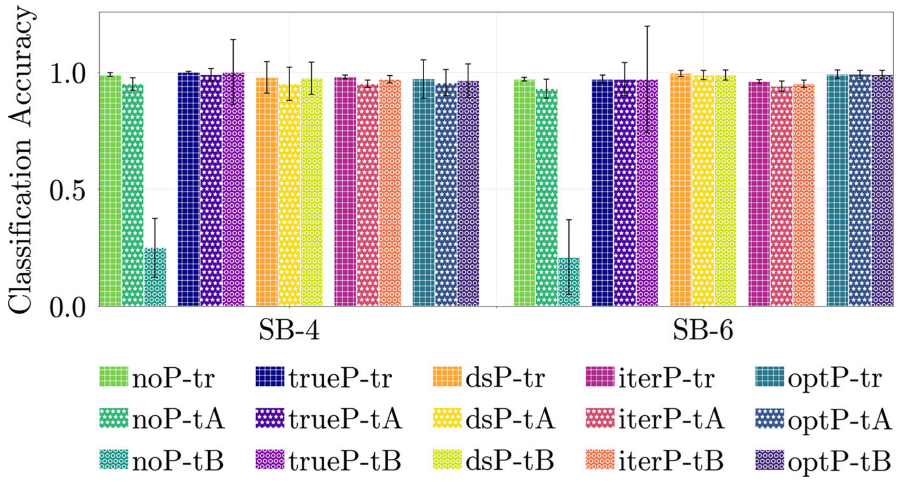


Fig. 5 Classification accuracy, ACC, w.r.t. clustering labels of different transfer learning algorithms (*noP*, *trueP*, *dsP*, *iterP*, *optP*) on two synthetic datasets (*SB-4* and *SB-6*). Each group of 3 ACC values is for training (*tr*) and testing (*tA*) subsets of graph G_A, and testing subset 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

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

Table 5 Classification label accuracy, ACC, on *BP-2, SB-4, SB-6, ZKC,* and *Email* datasets for the *noP* and *iterP* transfer algorithms

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

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 5 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.

6.2.4 Transferring epidemic spread labels

Figure 6 illustrates the performance of the five algorithms, viz. *noP*, *trueP*, *dsP*, *iterP*, *optP*, employed to transfer influence/epidemic spread labels from graph G_A to graph G_B of two synthetic datasets, *SB-4* and *SB-6*. In concordance with previously discussed experiments on clustering labels, we compare algorithms' performance on test subset of graph G_B (*tB*) w.r.t. root-mean-square error RMSE and coefficient of determination R^2.

Table 6 shows the predicted 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 ($\mathrm{R}^2 < 0$), while prediction on G_B under *iterP* is almost as good as prediction on G_A (tA), sometimes even better (e.g., for *SB-6* and *Email*).

6.2.5 Impact of graph perturbation

Figure 7 illustrates performance with respect to prediction accuracy and RMSE on the *SB-4* and *SB-6* datasets obtained for different perturbation factors (see Sect. 6.1.3). 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

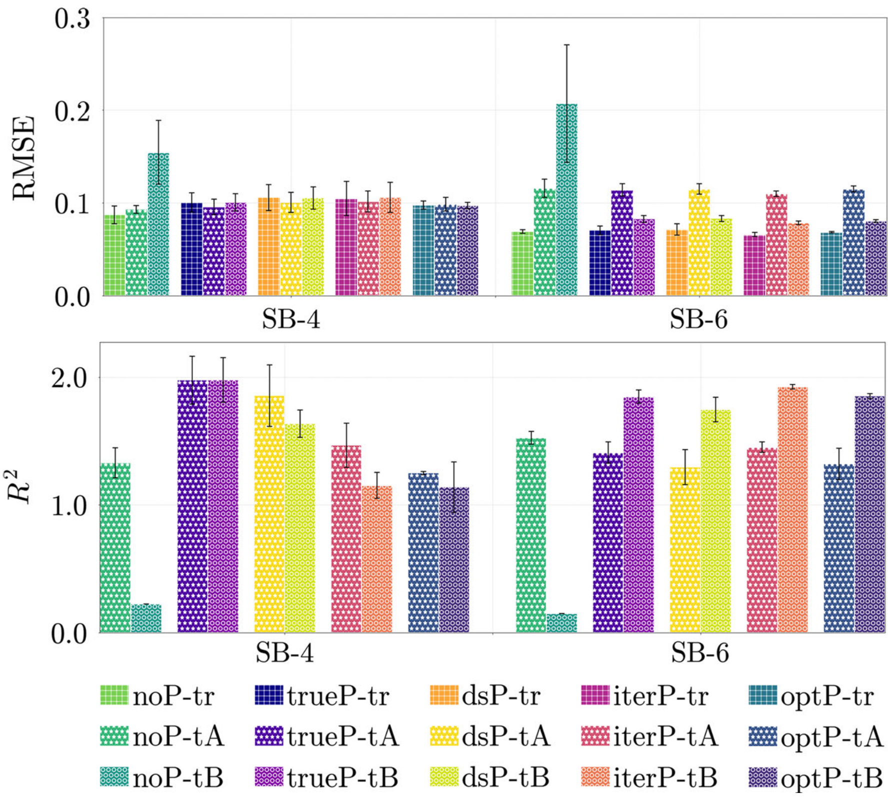


Fig. 6 Performance of different transfer learning algorithms (*noP*, *trueP*, *dsP*, *iterP*, *optP*) on epidemic spread/influence labels of two synthetic datasets (*SB-4* and *SB-6*), w.r.t both coefficient of determination $\mathtt{R}^2$ and `RMSE`. We report 3 `RMSE` values, for training (*tr*) and testing (*tA*) subsets of graph G_A, and testing subset of graph G_B (*tB*), while we report only two $\mathtt{R}^2$ values (for test subsets, *tA* and *tB*, respectively), for each algorithm. For `RMSE`, the smaller values the better; for $\mathtt{R}^2 \in (-\infty, 1]$, the higher the better. For the sake of visualization, we use exponential scale for the latter: $\exp(\mathtt{R}^2) \in (0, e]$. We observe that (**a**) naïve scenario (*noP*) demonstrates much worse performance on *tB* than on *tr* and *tA* for both `RMSE` and $\mathtt{R}^2$ metrics; (**b**) when P is learned (either using projected gradient descent (*iterP*) or constrained optimization (*optP*) methods), `RMSE` on *tB* is on par with, and $\mathtt{R}^2$ is on par or slightly worse than for *trueP* and *dsP* scenarios, when true permutation and doubly stochastic matrices are used, respectively; and (**c**) the performance under *iterP* and *optP* scenarios is on par with train accuracy, and sometimes even surpasses test accuracy on G_A (*tr*, *tA*)

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 remains 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* 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).

Table 6 Influence/epidemic spread label prediction performance of *noP* and *iterP* transfer learning algorithms on *SB-4, SB-6, ZKC, Email* and *IDTD* datasets

Dataset	*noP*			*iterP*		
	tr RMSE	*tA* RMSE/R^2	*tB* RMSE/R^2	*tr* RMSE	*tA* RMSE/R^2	*tB* 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

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*)

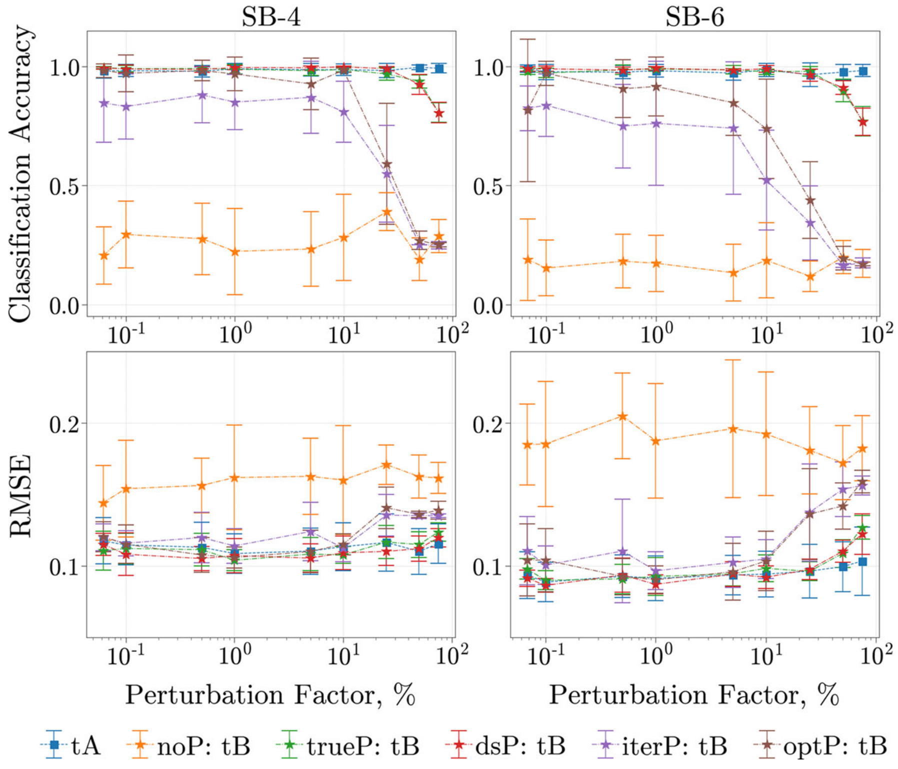


Fig. 7 Effect of edge perturbations between G_A and G_B on the label prediction performance (in ACC and RMSE for clustering and influence labels, respectively) studied on synthetic datasets, *SB-4* and *SB-6*. Transferability is possible even with a 25% perturbation factor, with almost no impact in the $< 10\%$ range

7 Conclusion

We study the graph transfer learning problem using a group of synthetic and real-world datasets. Our proposed method, being generic with respect to the employed embedding methods, offers strong evidence that structural labels can be successfully transferred across graphs. 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.

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

References

1. Ahmed A, Shervashidze N, Narayanamurthy SM, Josifovski V, Smola AJ (2013) Distributed large-scale natural graph factorization. In: Proceedings of the 22nd international world wide web conference, WWW, 2013, pp 37–48
2. Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15(6):1373–1396
3. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD, pp 855–864
4. Pan SJ, Yang Q (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359
5. Bento J, Ioannidis S (2019) A family of tractable graph metrics. Appl Netw Sci 4(1):107-1–107-27
6. Birkhoff G (1946) Tres observaciones sobre el algebra lineal [three observations on linear algebra]. Revista - Universidad Nacional de Tucumán, Serie A 5:147–151
7. Cao S, Lu W, Xu Q (2015) GraRep: learning graph representations with global structural information. In: Proceedings of the 24th ACM international conference on information and knowledge management, CIKM, pp 891–900
8. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. IEEE Data Eng Bull 40(3):52–74
9. Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. Knowl Based Syst 151:78–94
10. Cai H, Zheng VW, Chang KC (2018) A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans Knowl Data Eng 30(9):1616–1637
11. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. IEEE Trans Neural Netw 20(1):61–80
12. Li Y, Tarlow D, Brockschmidt M, Zemel RS (2016) Gated graph sequence neural networks. In: Proceedings of the 4th international conference on learning representations, ICLR,
13. Hamilton W L, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the annual conference on neural information processing systems, NeurIPS, pp 1024–1034
14. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th international conference on learning representations, ICLR
15. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34(4):18–42
16. Kipf TN, Welling M (2016) Variational graph auto-encoders. In: Proceedings of the NeurIPS Bayesian deep learning workshop
17. Pratt L, Jennings B (1996) A survey of connectionist network reuse through transfer. In Learning to Learn, 1996, pp 19–43
18. Do CB, Ng AY(2005) Transfer learning for text classification. Adv Neural Inf Process Syst pp 299–306
19. Wan C, Pan R, Li J (2011) Bi-weighting domain adaptation for cross-language text classification. In: Proceedings of the 21d international joint conference on artificial intelligence, IJCAI
20. Lu Z, Zhu Y, Pan SJ, Xiang EW, Wang Y, Yang Q (2014) Source free transfer learning for text classification. In: Proceedings of the 28th conference on artificial intelligence, AAAI, pp 122–128

21. Lee J, Kim H, Lee J, Yoon S (2017) Transfer learning for deep learning on graph-structured data. In: Proceedings of the 31st conference on artificial intelligence, AAAI, pp 2154–2160
22. Gong K, Gao Y, Liang X, Shen X, Wang M, Lin L (2019) Graphonomy: universal human parsing via graph transfer learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, CVP, pp 7450–7459
23. Verma S, Zhang Z-L (2019) Learning universal graph neural network embeddings with aid of transfer learning, arXiv preprint arXiv:1909.10086
24. Banerjee B, Stone P (2007) General game learning using knowledge transfer. In: Proceedings of the 20th international joint conference on artificial intelligence, IJCAI, pp 672–677
25. Kuhlmann G, Stone P (2007) Graph-based domain mapping for transfer learning in general games. In: Proceedings of the 18th European conference on machine learning, ECML, pp 188–200
26. Long M, Wang J, Ding G, Shen D, Yang Q (2013) Transfer learning with graph co-regularization. IEEE Trans Knowl Data Eng 26(7):1805–1818
27. Piao G, Breslin JG (2018) Transfer learning for item recommendations and knowledge graph completion in item related domains via a co-factorization model. In: Proceedings of the 15th extended semantic web conference, ESWC, pp 496–511
28. Conte D, Foggia P, Sansone C, Vento M (2004) Thirty years of graph matching in pattern recognition. Int J Pattern Recognit Artif Intell 18(3):265–298
29. Allen FH (2002) The cambridge structural database: a quarter of a million crystal structures and rising. Acta Crystallogr B 58(3):380–388
30. Kvasnička V, Pospíchal J, Baláž V (1991) Reaction and chemical distances and reaction graphs. Theoret Chem Account Theory Comput Model 79(1):65–79
31. Macindoe O, Richards W (2010) Graph comparison using fine structure analysis. In Proceedings of the IEEE 2nd international conference on social computing, SocialCom, 2010, pp 193–200
32. Faloutsos C, Koutra D, Vogelstein JT(2013) DELTACON: a principled massive-graph similarity function. In: Proceedings of the 13th SIAM international conference on data mining, ICDM, pp 162–170
33. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co,
34. Fischer A, Suen CY, Frinken V, Riesen K, Bunke H (2015) Approximation of graph edit distance based on hausdorff matching. Pattern Recogn 48(2):331–343
35. Bunke H (1997) On a relation between graph edit distance and maximum common subgraph. Pattern Recogn Lett 18(8):689–694
36. Bunke H, Shearer K (1998) A graph distance metric based on the maximal common subgraph. Pattern Recogn Lett 19(3–4):255–259
37. Chartrand G, Kubicki G, Schultz M (1998) Graph similarity and distance in graphs. Aequationes Mathematicae
38. Jain BJ (2016) On the geometry of graph spaces. Discret Appl Math 214:126–144
39. Koca J, Kratochvil M, Kvasnicka V, Matyska L, Pospichal J (2012) Synthon model of organic chemistry and synthesis design. Springer Science & Business Media, vol 51
40. Riesen K, Neuhaus M, Bunke K (2007) Graph embedding in vector spaces by means of prototype selection. Graph-Based Represent Pattern Recogn 4538:383–393
41. Riesen K, Bunke H (2010) Graph Classification and clustering based on vector space embedding. World Scientific
42. Ferrer M, Valveny E, Serratosa F, Riesen K, Bunke H (2010) Generalized median graph computation by means of graph embedding in vector spaces. Pattern Recogn 43(4):1642–1655
43. Zhu P, Wilson RC (2005) A study of graph spectra for comparing graphs. In: Proceedings of the the British machine vision conference, BMVC
44. Wilson RC, Zhu P (2008) A study of graph spectra for comparing graphs and trees. Pattern Recogn 41:2833–2841
45. Elghawalby H, Hancock ER (2008) Measuring graph similarity using spectral geometry. In: Proceedings of the 5th international conference on image analysis and recognition, ICIAR, 2008, pp 517–526
46. Zhang S, Tong H (2016) Final: fast attributed network alignment. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD, pp 1345–1354
47. Riesen K, Bunke H (2009) Approximate graph edit distance computation by means of bipartite graph matching. Image Vis Comput 27(7):950–959
48. Fankhauser S, Riesen K, Bunke H (2011) Speeding up graph edit distance computation through fast bipartite matching. In: Proceedings of the 8th international workshop on graph-based representations in pattern recognition, GbRPR, pp 102–111

49. Heimann M, Shen H, Safavi T, Koutra D (2018) REGAL: representation learning-based graph alignment. In: Proceedings of the 27th ACM international conference on information and knowledge management, CIKM, pp 117–126
50. Chen X, Heimann M, Vahedian F, Koutra D (2020) CONE-align: consistent network alignment with proximity-preserving node embedding. In: Proceedings of the The 29th ACM international conference on information and knowledge management, CIKM, pp 1985–1988
51. Kempe D, Kleinberg JM, Tardos É (2003) Maximizing the spread of influence through a social network. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining, KDD, 2003, pp. 137–146
52. Myers SA, Leskovec J (2010) On the convexity of latent social network inference. In: Proceedings of the 24th annual conference on neural information processing systems, NeurIPS, pp 1741–1749
53. Gomez-Rodriguez M, Leskovec J, Krause A (2012) Inferring networks of diffusion and influence. ACM Trans Knowl Discov Data 5(4):211–2137
54. Abrahao B D, Chierichetti F, Kleinberg R, Panconesi A (2013) Trace complexity of network inference. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, KDD, pp 491–499
55. Morin F, Bengio Y (2005) Hierarchical probabilistic neural network language model. In: Proceedings of the 10th international workshop on artificial intelligence and statistics, AISTATS
56. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of the 27th annual conference on neural information processing systems, NeurIPS, pp 3111–3119
57. Gold S, Rangarajan A (1996) A graduated assignment algorithm for graph matching. IEEE Trans Pattern Anal Mach Intell 18(4):377–388
58. Wiskott L, Fellous J, Krüger N, von der Malsburg C (1997) Face recognition by elastic bunch graph matching. In: Proceedings of the 7th international conference on computer analysis of images and patterns, CAIP, vol 1296, pp 456–463
59. Babai L (2016) Graph isomorphism in quasipolynomial time [extended abstract]. In Proceedings of the 48th annual ACM SIGACT symposium on theory of computing, STOC, 2016, pp 684–697
60. Frank M, Wolfe P (1956) An algorithm for quadratic programming. Naval Res Logist Quart 3(1–2):95–110
61. Boyd SP, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn 3(1):1–122
62. Andersen M, Dahl J, Liu Z, Vandenberghe L, Sra S, Nowozin S, Wright S (2011)Interior-point methods for large-scale cone programming. Optim Mach Learn 5583
63. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press
64. Bertsekas DP (1999) Nonlinear programming. Athena Scientific Belmont,
65. Kuhn HW (1955) The hungarian method for the assignment problem. Naval Res Logist Quart 2(1–2):83–97
66. Michelot C (1986) A finite algorithm for finding the projection of a point onto the canonical simplex of R^n. J Optim Theory Appl 50(1):195–200
67. Gold S, Rangarajan A (1996) Softmax to Softassign: neural network algorithms for combinatorial optimization. J Art Neural Netw 2(4):381–399
68. Zachary WW (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33(4):452–473
69. Leskovec J, Kleinberg JM, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. ACM Trans Knowl Discov Data 1(1): 2–es
70. Salathé M, Kazandjieva M, Lee JW, Levis P, Feldman MW, Jones JH (2010) A high-resolution human contact network for infectious disease transmission. Proc Natl Acad Sci 107(51):22020–22025
71. Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113
72. Zhu Y, Xu Y, Yu F, Liu Q, Wu S, Wang L (2021) Graph contrastive learning with adaptive augmentation. In: WWW ’21: the web conference 2021, Virtual Event/Ljubljana, Slovenia, April 19–23, 2021. ACM / IW3C2, 2021, pp 2069–2080
73. Wang D, Cui P, Zhu W (2016) Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp 1225–1234
74. Leskovec J, Lang KJ, Dasgupta A, Mahoney MW (2009) Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Int Math 6(1):29–123
75. Erdös P, Rényi A (1959) On Random Graphs I. Publicationes Mathematicae Debrecen 6:290–297
76. Holland PW, Laskey KB, Leinhardt S (1983) Stochastic blockmodels: first steps. Soc Netw 5(2):109–137

77. Rossetti G, Milli L, Rinzivillo, S Sîrbu A, Pedreschi D, Giannotti F (2017) NDlib: studying network diffusion dynamics. In: Proceedings of the IEEE international conference on data science and advanced analytics, DSAA, 2017, pp 155–164
78. Glantz SA, Slinker BK, Neilands TB (1990) Primer of applied regression and analysis of variance, vol 309. McGraw-Hill
79. Draper NR, Smith H (1998) Applied regression analysis, vol 326. Wiley

Andrey Gritsenko is a Research Associate in the Electrical and Computer Engineering Department of Northeastern University, Boston, MA. He received his MSc (2010) in Applied Mathematics and Computer Science from the Stavropol State University, Russia, and his PhD (2017) in Industrial Engineering from the University of Iowa, IA. He is a recipient of the Best Technical Paper award at IEEE DySPAN 2019, and a Google Cloud Certified Professional Machine Learning Engineer. His research interests span the areas of interpretable machine learning, graph mining, and signal processing.

Kimia Shayestehfard is a PhD candidate in the Electrical and Computer Engineering Department of Northeastern University, in Boston, MA. She received her BSc (2014) in Electrical and Computer Engineering from Shiraz University, Shiraz, Iran, and her MSc (2018) in Electrical and Computer Engineering from Northeastern University, in Boston, MA. Her research interests span machine learning, graph mining, and optimization.

Yuan Guo is a senior software engineer in Baidu Research, USA. Before joining Baidu, Yuan acquired PhD degree from the Electrical and Computer Engineering Department of Northeastern University, in Boston, MA. He received his BSc (2011) in Electrical and Computer Engineering from the Huazhong University of Science and Technology, Wuhan, China, and his MSc (2014) in Electrical and Computer Engineering from Tsinghua University, Beijing, China. His current research interests include natural language processing and video processing.

Armin Moharrer is currently an Artificial Intelligence Engineer at Liminal Sciences, in Palo Alto, CA. He received his BSc (2015) degree in Electrical Engineering from Amirkabir University of Technology (Tehran Polytechnic), in Tehran, Iran and his MSc (2018) degree and his PhD (2021) degree from the department of Electrical and Computer Engineering at Northeastern University in Boston, MA.

Jennifer Dy is Professor at the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, where she first joined the faculty in 2002. She received her MS and PhD in 1997 and 2001, respectively, from Purdue University, and her BS degree from the University of the Philippines in 1993. Her research spans both fundamental research in machine learning and their application to biomedical imaging, health, science and engineering, with research contributions in unsupervised learning, dimensionality reduction, feature selection, learning from uncertain experts, active learning, Bayesian models, and deep representations. She received an NSF Career award in 2004. She has served or is serving as Secretary for the International Machine Learning Society, associate editor/editorial board member for the Journal of Machine Learning Research, Machine Learning journal, IEEE Transactions on Pattern Analysis and Machine Intelligence, organizing and or technical program committee member for premier conferences in machine learning and data mining (ICML, NeurIPS, ACM SIGKDD, AAAI, IJCAI, UAI, AISTATS, SIAM SDM), and program co-chair for SIAM SDM 2013 and ICML 2018.

Stratis Ioannidis is an associate professor in the Electrical and Computer Engineering Department of Northeastern University, in Boston, MA, where he also holds a courtesy appointment with the Khoury College of Computer Sciences. He received his BSc (2002) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, and his MSc (2004) and PhD (2009) in Computer Science from the University of Toronto, Canada. Prior to joining Northeastern, he was a research scientist at the Technicolor research centers in Paris, France, and Palo Alto, CA, as well as at Yahoo Labs in Sunnyvale, CA. He is the recipient of an NSF CAREER Award, a Google Faculty Research Award, and a best paper award at ACM ICN 2017 and IEEE DySPAN 2019. His research interests span machine learning, distributed systems, networking, optimization, and privacy.