# SeedGNN: Graph Neural Network for Supervised Seeded Graph Matching

**Liren Yu** [1] **Jiaming Xu** [2] **Xiaojun Lin** [1]

## Abstract

There is a growing interest in designing Graph Neural Networks (GNNs) for seeded graph matching, which aims to match two unlabeled graphs using only topological information and a small set of seed nodes. However, most previous GNNs for this task use a semi-supervised approach, which requires a large number of seeds and cannot learn knowledge that is transferable to unseen graphs. In contrast, this paper proposes a new supervised approach that can learn from a training set how to match unseen graphs with only a few seeds. Our SeedGNN architecture incorporates several novel designs, inspired by theoretical studies of seeded graph matching: 1) it can learn to compute and use witness-like information from different hops, in a way that can be generalized to graphs of different sizes; 2) it can use easily-matched node-pairs as new seeds to improve the matching in subsequent layers. We evaluate SeedGNN on synthetic and real-world graphs and demonstrate significant performance improvements over both non-learning and learning algorithms in the existing literature. Furthermore, our experiments confirm that the knowledge learned by SeedGNN from training graphs can be generalized to test graphs of different sizes and categories.

## 1. Introduction

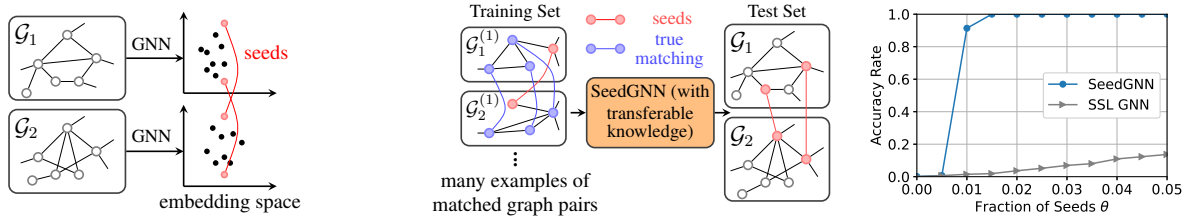
Graph matching, also known as network alignment, aims to find the node correspondence between two graphs that maximally aligns their edge sets. As a ubiquitous but challenging problem, graph matching has numerous applications, including social network analysis (Narayanan & Shmatikov, 2008; 2009; Zafarani et al., 2015; Zhang et al., 2015; Zhang & Philip, 2015; Chiasserini et al., 2016), computer vision (Conte et al., 2004; Schellewald & Schnörr, 2005; Vento & Foggia, 2013), natural language processing (Haghighi et al., 2005), and computational biology (Singh et al., 2008; Kazemi et al., 2016; Kriege et al., 2019). This paper focuses on seeded graph matching, where a small portion of the node correspondence between the two graphs is revealed as seeds, and we seek to complete the correspondence using the few seeded node-pairs. Seeded graph matching is motivated by the fact that, in many real applications, the correspondence between a small portion of the two node sets is naturally available. For example, in social network de-anonymization, some users who explicitly link their accounts across different social networks could become seeds (Narayanan & Shmatikov, 2008; 2009). Knowledge of even a few seeds has been shown to significantly improve the matching results for many real-world graphs (Kazemi et al., 2015; Fishkind et al., 2019).

Recently, the Graph Neural Network (GNN) approach for graph matching has attracted much research attention. Although such a machine-learning-based approach usually does not possess provable theoretical guarantees, it has the potential to learn valuable features from a large set of training data. Unfortunately, to date GNN has not been successfully applied to seeded graph matching. Most previous GNNs for seeded graph matching are limited to a *semi-supervised* learning (SSL) paradigm, which only operates on a *single* pair of graphs and treats the seed set as the labelled training data (Zhang et al., 2019; Li et al., 2019a;b;c; Zhou et al., 2019; Chen et al., 2020; Derr et al., 2021). The goal is to learn from the seed set useful features that can be used to compute node embeddings for all nodes (see Figure 1(a)). However, this semi-supervised learning suffers from two major limitations. First, in order to obtain high matching accuracy, the set of seeds needs to be large, which is often unrealistic in practice. Second, as this semi-supervised setting only learns *within* a given pair of graphs, there is no effort in *transferring knowledge* from one pair of graphs to other pairs of unseen graphs, which severely limits GNNs' potential in distilling the common knowledge from a large set of training graphs. A natural but fundamental question is that

*Can we learn to match two graphs with only a few seeds while generalizing to unseen graphs?*

[1]Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, USA [2]The Fuqua School of Business, Duke University, Durham, North Carolina, USA. Correspondence to: Liren Yu <yu827@purdue.edu>, Jiaming XU <jx77@duke.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).




(a) In semi-supervised learning, only one pair of graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is provided. The GNN is applied to each graph separately. Taking the topological and non-topological features as input, this GNN maps each node to a node embedding in some space and matches nodes across two graphs based on the closeness in node embeddings. Using seeds as the training set, this GNN is trained so that the seed pairs have close node embeddings. However, when the number of seeds is small (while the GNN complexity is high), other true pairs may still have very different embeddings, leading to low matching accuracy.

(b) Our SeedGNN is trained with many examples of matched graph pairs (on the left). Each training graph pair comes with the true matching (in blue), as well as the seeds (in red). With training, SeedGNN can distill transferable knowledge on how to effectively use a small number of seeds to best match other nodes. This SeedGNN is then applied to an unseen graph pair (on the right) to achieve high matching accuracy with only a limited number of seeds.

(c) Performance comparison on correlated Erdős-Rényi graphs shows that, with a very small fraction of seeds, SeedGNN can already achieve high matching accuracy, while SSL GNN almost completely fails. See detailed set-up in Section 6.1.

Figure 1: Illustration of the advantages of supervised learning over semi-supervised learning for seeded graph matching.

This paper provides an affirmative answer to this question. Specifically, we design a novel GNN architecture through a supervised approach, namely SeedGNN, that can learn from many examples of matched graphs, distill the knowledge into the trained model automatically, and then apply such knowledge to match unseen graph pairs with only a small number of seeds (see an illustration in Figure 1(b)). The performance comparison on correlated Erdős-Rényi graphs (Pedarsani & Grossglauser, 2011) shown in Figure 1(c) demonstrates that our SeedGNN requires significantly fewer seeds to achieve higher matching accuracy than the SSL GNN.

Our supervised SeedGNN incorporates two valuable insights from theoretical studies of seeded graph matching. The first insight is the notion of "witnesses" (Korula & Lattanzi, 2014; Mossel & Xu, 2019). For node $v$ in graph $\mathcal{G}_1$ and node $w$ in graph $\mathcal{G}_2$, we say a seed pair $(u, u')$ is a *$\ell$-hop witness* for $(v, w)$ if $u$ is an $\ell$-hop neighbor of $v$ in $G_1$ and $u'$ is an $\ell$-hop neighbor of $w$ in $G_2$. For example, in Figure 2 the seed pair $(2, 2')$ is a 1-hop witness for $(1, 1')$. For many graph matching problems, a true pair tends to have more witnesses than fake pairs (Korula & Lattanzi, 2014; Mossel & Xu, 2019). Thus, the number of witnesses can be used to measure the similarity of node-pairs. The second insight is the notion of "percolation." In particular, note that some true pairs are easier to match than others. Some theoretical algorithms use these easily-matched true pairs as new seeds to match other nodes (Korula & Lattanzi, 2014; Kazemi et al., 2015). For example, in Figure 2, since the node-pairs $(1, 1')$ and $(4, 4')$ have a 1-hop witness, these two node-pairs can be matched first. Then, used as new seeds, they become the witnesses for other node-pairs, such as $(3, 3')$ and $(6, 6')$. When done properly, a percolation process can be triggered to match a large number of nodes. For some graphs, these two insights have been crucial for theoretical algorithms to successfully match graphs of $n$ nodes with only $\Theta(\log n)$ seeds (Korula & Lattanzi, 2014; Kazemi et al., 2015; Yu et al., 2021b).

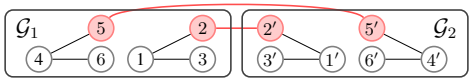



Figure 2: The node-pairs $(i, i')$ are true matches and the red node-pairs are seeds.

Compared to the existing GNNs, our SeedGNN architecture is carefully designed to effectively and explicitly incorporate the above two insights. First, most existing GNNs are "node-based," i.e., they are applied to each node to compute a node embedding. In contrast, SeedGNN is "pair-wise," i.e., it is applied to each pair of nodes across the two graphs. As we elaborate in Section 4.2, this pair-wise architecture is much more effective in learning how to compute and use witnesses in a way that can be generalized to graphs with different sizes. Second, we carefully design a percolation module to filter out node-pairs with low similarities. As a result, only the "cleaner" new seeds are used to trigger the percolation process.

Note that a similar pair-wise architecture has appeared before in NGM (Wang et al., 2021) for seedless graph matching. However, the NGM architecture was not designed for seeded graph matching, and there are crucial differences that prevent the effective use of the witnesses and percolation ideas. We analytically show in Section 5 that, for seeded graph matching, such differences can significantly affect the matching performance when the number of seeds is small. Our experiments in Section 6.2 further verify that NGM does not generalize well when the test graphs have much larger sizes and node-degrees than the training graphs.

Our numerical experiments (in Section 6) on both synthetic and real-world graphs show that SeedGNN significantly outperforms the state-of-the-art algorithms, including both

non-learning and learning-based ones, in terms of seed-size requirement and matching accuracy. Moreover, our SeedGNN can generalize to match unseen graphs of sizes and types different from the training set.

## 2. Further Related Work

In this section, we discuss further related work. More discussion on additional related work is deferred to Appendix A.

**Theoretical Algorithms** Various seeded matching algorithms have been proposed based on hand-designed similarity metrics computed from local topological structures (Pedarsani & Grossglauser, 2011; Yartseva & Grossglauser, 2013; Korula & Lattanzi, 2014; Shirani et al., 2017; Mossel & Xu, 2019; Yu et al., 2021b). The theoretical analysis on these algorithms explains why a particular set of features (e.g., witnesses (Korula & Lattanzi, 2014) and percolation (Yartseva & Grossglauser, 2013)) are valuable for graph matching. However, these theoretical algorithms require carefully hand-tuned parameters and may not synthesize different features most effectively (see detailed discussion in Appendix A). In contrast, SeedGNN can potentially learn (from the training data) what combinations of features are most useful, and thus outperform theoretical algorithms, as shown in our experiments Section 6.2.

**GNN for Seedless Graph Matching** As aforementioned, most existing GNNs for seeded graph matching take a semi-supervised learning approach. In contrast, our SeedGNN falls into a supervised learning approach, which aims to transfer knowledge from training graphs to unseen graphs. In the literature, such a supervised learning approach has been applied to *seedless* versions of the graph matching problems in (Zanfir & Sminchisescu, 2018; Wang et al., 2019; 2021; 2020a; 2021; Jiang et al., 2022; Wang et al., 2020b; Fey et al., 2020; Rolínek et al., 2020; Gao et al., 2021; Yu et al., 2021c). For such seedless matching problems, non-topological node features are often assumed to be available and informative. Thus, a node-based GNN is effective in learning how to extract useful node representations from high-quality non-topological node features. However, for seeded matching problems, it is difficult to design a node-based GNN to effectively utilize seed information (see further discussions in Section 4.2). In contrast, our pair-wise SeedGNN architecture is much more effective in learning how to use seed information.

## 3. Problem Definition

We represent a graph of $n$ nodes by $\mathcal{G} = (V, \mathbf{A})$, where $V = \{1, 2, ..., n\}$ denotes the node set, and $\mathbf{A} \in \{0, 1\}^{n \times n}$ denotes the adjacent matrix, such that $\mathbf{A}(i, j) = 1$ if and only if nodes $i$ and $j$ are connected. For seeded graph matching, we are given two graphs $\mathcal{G}_1 = (V_1, \mathbf{A}_1)$ of $n_1$ nodes and $\mathcal{G}_2 = (V_2, \mathbf{A}_2)$ of $n_2$ nodes. Without loss of generality, we assume $n_1 \leq n_2$. There is an *unknown* injective mapping $\pi : V_1 \to V_2$ between $\mathcal{G}_1$ and $\mathcal{G}_2$. When $\pi(i) = j$, we say that $i \in V_1$ corresponds to $j \in V_2$. Throughout the paper, we denote a node-pair by $(i, j)$, where $i \in V_1$ and $j \in V_2$. For each node-pair $(i, j)$, if $j = \pi(i)$, then $(i, j)$ is a *true pair*; if $j \neq \pi(i)$, then $(i, j)$ is a *fake pair*. Then, a seed set $\mathcal{S}$ containing a fraction of true pairs is given. The goal of seeded graph matching is to recover the ground-truth mapping $\pi$ based on the observation of $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{S}$.

In this work, we consider the problem of seeded graph matching in the supervised setting. The training set consists of several pairs of graphs, their initial seeds, and ground-truth mappings. Specifically, we use $\mathcal{T} = \{(P^{(1)}, \pi^{(1)}), (P^{(2)}, \pi^{(2)}), ..., (P^{(N)}, \pi^{(N)})\}$ to denote the training set, where $P^{(i)} = (\mathcal{G}_1^{(i)}, \mathcal{G}_2^{(i)}, \mathcal{S}^{(i)})$ denotes the $i$-th training example and $\pi^{(i)}$ is the ground-truth mapping for the $i$-th training example. For different training examples, the sizes of graphs and seed sets could be different. Our goal is to design a GNN architecture that can learn from training examples to predict the ground-truth mappings for unseen test graphs.

## 4. The Proposed Method

In this section, we present in detail our SeedGNN for seeded graph matching. See Figure 3 for a high-level illustration.

**Notation** we use $\mathsf{flatten}(\cdot)$ to denote the matrix reshape operation that converts a $n_1 \times n_2 \times d$ matrix to a matrix of $n_1 n_2 \times d$, where the $(i, j, :)$-th entry of the input matrix is the $((i-1)n_2 + j, :)$-th entry of the output matrix. Then, we use $\mathsf{unflatten}(\cdot)$ to denote the inverse operation of $\mathsf{flatten}(\cdot)$.

### 4.1. Generalizable Encoding Method for Seeds

We encode the seeded relationship as inputs for our SeedGNN. More precisely, let $s_1 \in \{0, 1\}^{n_1 n_2 \times 1}$ be the indicator vector for seeds among $n_1 n_2$ node-pairs. If the node-pair $(i, j)$ is a seed, we let the $((i-1)n_2 + j)$-th entry of $s_1$ be 1, and 0 otherwise.

We contrast our way of encoding seeds with an alternate one-hot encoding method. One-hot encoding assigns the $i$-th seeded node with a binary vector, whose $i$-th element being 1, and all other elements are 0. The benefit of our encoding method is that the dimension of the encoding vector is fixed at 1 for each node-pair, which does not depend on the graph size or the number of seeds. Thus, SeedGNN with our encoding method can be applied to unseen graphs with arbitrary graph sizes and numbers of seeds. In contrast, one-hot encoding method needs to pre-specify the maximum number of seeds, and thus GNNs with one-hot encoding can not generalize to new graphs with even more seeds.

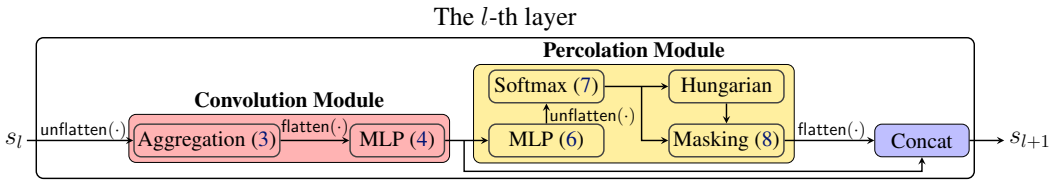



Figure 3: An overview of the $l$-th layer of our SeedGNN architecture. There are $L$ layers in total and each layer consists of two main modules. With the node-pair representations $s_l$ as input, the convolution module is a *local* processing step that aggregates the neighborhood information of each node-pair and updates the representation of its similarity through a neural network. The percolation module is a *global* processing step that compares the updated similarities of all node-pairs and finds the high-confidence ones. Then, we combine the local and global information from the two modules and propagate the new representations $s_{l+1}$ to the next layer.

### 4.2. Convolution Module

With the seed information encoded as 0/1 for each node-pair, we still need to carefully design a GNN architecture that can count witnesses. Note that most existing GNN approaches for graph matching are "node-based" (Zhang et al., 2019; Chen et al., 2020; Wang et al., 2019; 2020a; Fey et al., 2020; Rolínek et al., 2020). They apply a common GNN separately to each of the two graphs in order to learn a node embedding *for each node*. They then match nodes in the two graphs based on the similarity of the corresponding node embeddings. However, it is difficult for these approaches to utilize our newly-encoded seed information effectively. As shown in Figure 2, our encoding of seed information can also be viewed as "cross-links" (highlighted in red color) across the two graphs. With these "cross-links", we can then combine the two graphs together and apply the node-based GNN on this union graph. However, the topological structure of this union graph only informs the GNN that there is a seed at a particular location in the neighborhood, but not the seed identity. For example, in Figure 2, even though node $1'$ and node $4'$ have different seeds in their neighborhoods, their local neighborhood topologies (and the seed positions) look exactly the same. Thus, node-based GNN will have a hard time to come up with node embeddings such that node 1 has a close embedding to node $1'$ but not close to node $4'$. Instead, our SeedGNN is "pair-wise", i.e., it is applied on node-pairs instead of nodes. Intuitively, when we apply such a pair-wise GNN to the node-pairs $(1, 1')$ and $(1, 4')$ in Figure 2, it can easily tell that $(1, 1')$ has a witness, while $(1, 4')$ does not. As a result, this pair-wise GNN will count and utilize witnesses easily.

Specifically, taking the seed encoding vector $s_1$ as input, the counting of 1-hop witnesses can be written as

$$h_1 = (\mathbf{A}_1 \otimes \mathbf{A}_2)s_1 \tag{1}$$

where $\otimes$ denotes the Kronecker product. Applying (1) to Figure 2, we can get that the node-pairs $(1, 1')$ and $(4, 4')$ have a 1-hop witness, respectively. Likewise, we may further compute the $l$-hop witness-like information $h_l$ in the $l$-th layer of our SeedGNN as

$$h_l = (\mathbf{A}_1 \otimes \mathbf{A}_2)s_l, \tag{2}$$

where $s_l \in \mathbb{R}^{n_1 n_2 \times d_l}$ is specified later in Section 4.3, which contains the witness-like information within $(l-1)$-hops. Note that (2) can be expanded as, for node-pair $(i, j)$,

$$h_l[(i-1)n_2+j,:] = \sum_{\substack{(u,v):\\ \mathbf{A}_1(u,i)=\mathbf{A}_2(v,j)=1}} s_l[(u-1)n_2+v,:],$$

which is similar to the aggregation step of the standard GNN in (Hamilton et al., 2017). The only difference is that we aggregate over a node-pair's neighborhood. A direct implementation of (2) takes $O(n_1^2 n_2^2)$ computation, but we can reduce the complexity by letting $H_l = \mathsf{unflatten}(h_l)$ and $S_l = \mathsf{unflatten}(s_l)$, and rewriting (2) as

$$\begin{aligned} H_l[:,:,t] =& \mathsf{unflatten}((\mathbf{A}_1 \otimes \mathbf{A}_2)s_l[:,t]) \\ =& \mathbf{A}_1 S_l[:,:,t]\mathbf{A}_2, \qquad t = 1, 2, ..., d_l. \end{aligned} \tag{3}$$

Assume that the mean of the node degrees of $\mathcal{G}_1$ and $\mathcal{G}_2$ is at most $d_{\text{mean}}$. When we represent $A_1$ and $A_2$ with sparse matrices, each of them only contain $n_1 d_{\text{mean}}$ and $n_2 d_{\text{mean}}$ elements. Thus, by sparse matrix multiplication, the time complexity of Equation (3) is $O(n_1 n_2 d_{\text{mean}})$.

As we will see later in Section 4.3, $s_l$ will also contain outputs from the percolation layer. In order to learn how to best synthesize these two features, we apply a neural network on $h_l$ after (2):

$$m_l = \phi_l(h_l), \tag{4}$$

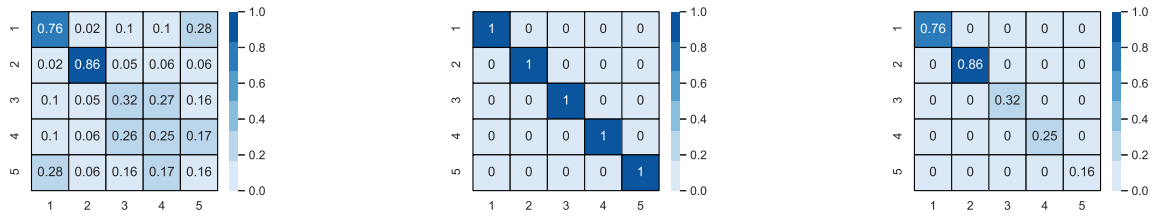
where the update function $\phi_l$ is implemented as a $K$-layer neural network (we use $K = 2$ in our experiment). Let $\phi_l^{[0]}(h_l) = h_l$. The $k$-th layer of $\phi_l$ can be formulated as

$$\phi_l^{[k]}(h_l) = \sigma\left(\phi_l^{[k-1]}(h_l)\boldsymbol{W}^{[k-1]} + \boldsymbol{b}^{[k-1]}\right), \tag{5}$$

where $\boldsymbol{W}^{[k-1]}$ and $\boldsymbol{b}^{[k-1]}$ are learnable weights, initialized as Gaussian random variables; $\sigma$ is an activation function (we use ReLU). The updated representations $m_l \in \mathbb{R}^{n_1 n_2 \times (d_l - 1)}$ will be sent to the next module of SeedGNN.

### 4.3. Percolation Module

The percolation module is designed to match high-confidence nodes at one layer and to propagate the matched




(a) Similarity/confidence matrix $Y_l$. (b) Output $R_l$ of the Hungarian algorithm. (c) $Z_l$ produced by masking.

Figure 4: Illustration of percolation module. The diagonal (resp. off-diagonal) entries correspond to true (resp. fake) pairs.

nodes as new seeds to the subsequent layers. Formally, we first obtain a similarity matrix in the $l$-th layer by mapping the node-pair representations $m_l$ to a 1-dimension vector, which is used to assess the similarity of each node-pair:

$$x_l = \rho_l(m_l). \tag{6}$$

We implement $\rho_l$ as a multi-layer neural network that is defined similarly as $\phi_l$ in (5). The output $x_l$ is in $\mathbb{R}^{n_1 n_2 \times 1}$. Then, we transform $x_l$ to $X_l = \mathsf{unflatten}(x_l) \in \mathbb{R}^{n_1 \times n_2}$, and apply row-wise softmax to normalize $X_l$ and obtain the similarity (confidence) matrix $Y_l$ for node-pairs:

$$Y_l = \left(\mathrm{softmax}(X_l) + \mathrm{softmax}(X_l^\top)^\top\right)/2, \tag{7}$$

where for each row $\boldsymbol{v} = (v_1, ..., v_n) \in \mathbb{R}^n$ of input matrix, the softmax function is defined as

$$\mathrm{softmax}(\boldsymbol{v})_i = \frac{\exp(v_i)}{\sum_{j=1}^n \exp(v_j)}, \quad \text{for } i = 1, 2, .., n.$$

The similarity matrix $Y_l$ needs "cleaning" because it contains a lot of "noisy" information. For example, many fake pairs may possess comparable similarity with true pairs (see Figure 4(a) for example). Further, there are far more fake pairs than true pairs. As a result, directly utilizing such misleading information may lead to even more matching errors. Inspired by the percolation idea from theoretical algorithms, which passes only new seeds with high confidence levels to the next stage (Yartseva & Grossglauser, 2013), we leverage an approach called "masking" to remove the noisy information and retain the cleaner information in $Y_l$. Specifically, we utilize the Hungarian matching algorithm (Edmonds & Karp, 1972) to solve a linear assignment problem on $Y_l$ to find an injective mapping between $\mathcal{G}_1$ and $\mathcal{G}_2$, such that the total similarity of the matched node-pairs is maximized (see Figure 4(b) for example). The matching result is denoted by $R_l \in \{0,1\}^{n_1 \times n_2}$, where $R_l(i,j) = 1$ if the node-pair $(i,j)$ is matched by the Hungarian algorithm, and $R_l(i,j) = 0$ otherwise. Then, we filter out the noisy information in $Y_l$ by "masking":

$$z_l = \mathsf{flatten}(Y_l \circ R_l), \tag{8}$$

where $\circ$ denotes element-wise multiplication (see Figure 4(c) for example). The matching information $z_l$ is sent to the next layer. As a result, many noisy node-pairs are discarded. We note that both the idea of using similarity matrix to refine higher-layer matching and the idea of masking have appeared in seedless matching (Wang et al., 2019; Fey et al., 2020; Yu et al., 2019). However, (Wang et al., 2019; Fey et al., 2020) do not clean up the "noisy" information as we carefully did, and (Yu et al., 2019) only applies the Hungarian algorithm in their loss function (but not the intermediate layers). Readers can refer to the numerical results in Appendix E.1, which demonstrate the importance of carefully cleaning up "noisy" information in each layer. Further, unlike previous percolation algorithms (Yartseva & Grossglauser, 2013), our design of the percolation module can correct matching errors from earlier layers. We discuss these differences further in Appendix A.

**The combination of the two features** With the convolution module and the percolation module, our SeedGNN can identify witnesses-like information at different hops and generate new seeds for percolation. However, these capabilities alone are insufficient. For example, when graphs are very sparse, even true node-pairs may not have enough witnesses if the number of hops $l$ is small. When graphs are very dense, a fake pair may also have many witnesses if $l$ is large. Thus, SeedGNN needs to learn how to adaptively utilize various types of witnesses in different types of graphs. Similarly, even with the above "cleaning" procedure, the output of the percolation module may still have low-confidence seeds. Directly using them for percolation could lead to cascading errors. Thus, SeedGNN also needs to learn how to use new seeds with different levels of confidence.

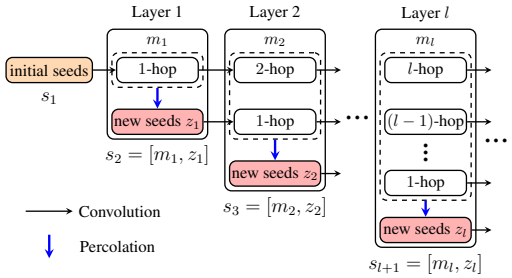



Figure 5: The witness-like information and new seeds computed by each layer.

The neural module in (4) is precisely designed to enable such learning. Specifically, instead of directly using the output $z_l$ from the percolation module as new seeds, we concatenate it with the output of the convolution module, i.e.,

$s_{l+1} = [m_l, z_l] \in \mathbb{R}^{n_1 n_2 \times d_{l+1}}$, as the input to the next layer. Then, after passing $s_{l+1}$ through (2), we apply the neural module (4). The joint effect of this design is that SeedGNN can utilize the confidence levels of $z_l$ to decide how much it should rely on various types of witnesses. Intuitively, at a higher layer $l \geq 2$, after passing $s_l$ by (2) and (4), $m_l$ may contain $l$-hop witness-like information from the initial seeds $s_1$, $(l-1)$-hop witness-like information from new seeds $z_1$, ... and 1-hop witnesses information from new seeds $z_{l-1}$ (see Figure 5). However, unlike the initial seeds $s_1$ that are either 0 or 1, the new seeds $z_1$, $z_2$, ..., $z_{l-1}$ also come with confidence levels. Thus, thanks to the non-linearity in $\phi_l$ at each layer, the strength of the various types of witness-like information (from either the initial seeds or the new seeds) will vary depending on the confidence levels of the new seeds, which then allows SeedGNN to learn how to best utilize them adaptively. For example, for sparse graphs, the confidence levels of the new seeds in the first several layers are low. As a result, SeedGNN can utilize witnesses based on the initial seeds but at a larger number of hops. In contrast, for dense graphs, if the confidence levels of the new seeds in the first several layers are already high, SeedGNN can then utilize the new witnesses computed from those new seeds. This capability is experimentally validated in Appendix E.3 by studying the layer-wise matching process of SeedGNN for different types of graphs. Further, SeedGNN can even combine different types of witness-like information together and extract more valuable features.

### 4.4. Loss Function

Finally, we utilize the ground-truth node correspondence as the supervised training information for end-to-end training. More precisely, for any training example $(P, \pi) \in \mathcal{T}$, we adopt the cross-entropy loss to measure the difference between our prediction and the ground-truth mapping $\pi$. Then, we add up the cross-entropy loss of every layer:

$$\mathcal{L}_P(\vartheta) = -\sum_{l=1}^{L} \left( \sum_{(i,j):\ j=\pi(i)} \log\left(Y_l(i,j) + \epsilon\right) + \sum_{(i,j),\ j\neq\pi(i)} \log\left(1 - Y_l(i,j) + \epsilon\right) \right),$$

where $Y_l$ is given in (7), $\vartheta$ denotes all the learnable weights in the networks $\phi_l$ and $\rho_l$, and $\epsilon$ is a small positive value (e.g. $\epsilon = 10^{-9}$) to avoid a logarithm of zero. The total loss function is $\mathcal{L}(\vartheta) = \sum_{P\in\mathcal{T}} \mathcal{L}_P(\vartheta)$. We find that the use of the losses from all layers in training helps to speed up the training process. This is somewhat inspired by hierarchical learning methods in (Bengio, 2009; Schmidhuber, 1992; Simonyan & Zisserman, 2015). It allows the lower layers to be trained first, making it easier to train the next layers. In testing, we will apply the trained SeedGNN model on the test graphs and only use the matching result of the final layer, $R_L$, as the predicted mapping since the final layer already synthesizes all the features learned at the lower layers.

The total time complexity of SeedGNN is $O(n_1 n_2^2)$, and the space complexity is $O(n_1 n_2)$. The detailed discussion on the complexity and scalability of our SeedGNN is deferred to Appendix C.

## 5. Theoretical Comparison Study

We note that NGM (a supervised seedless GNN method) in (Wang et al., 2021) bears some similarity with SeedGNN, because NGM also applies a pair-wise GNN, which uses an aggregation step similar to (2). However, a crucial difference is that after (2), NGM divides each node-pair representation by the product of the degrees of the corresponding two nodes. This type of normalization is quite common in GNNs to transform the non-topological features to a similar scale. However, this division can lead to very poor performance for seeded graph matching. The reason is that, if the number of seeds is small (e.g., $O(\log n)$) and the node degree increases proportionally to $n$, after the normalization step in NGM, we expect that the resulting output value ($O(\frac{\log n}{n^2})$) will decrease to zero as $n$ increases. Hence, we expect that it would be difficult for NGM to distinguish the true pairs from the fake pairs in test graphs with larger sizes and node degrees than the training graphs.

To formally study this effect, we conduct a theoretical study on a widely-adopted graph matching model, the correlated Erdős-Rényi graph model $\mathcal{G}(n, p, s; \theta)$ (Pedarsani & Grossglauser, 2011). We first generate the parent graph $\mathcal{G}_0$ from the Erdős-Rényi model $\mathcal{G}(n, p)$ with $n$ nodes and edge probability $p$. Then, we obtain a subgraph $\mathcal{G}_1$ by sampling each edge of $\mathcal{G}_0$ independently with probability $s$. Repeat the same sampling process independently to obtain another subgraph $\mathcal{G}_2$. Then, each true pair among $\mathcal{G}_1$ and $\mathcal{G}_2$ is independently added into the seed set $\mathcal{S}$ with probability $\theta$. We assume that the training set and test set have the same parameters $p, s, \theta$, and the only difference is the graph size, denoted by $n_{\text{train}}$ and $n_{\text{test}}$, respectively.

For ease of analysis, we focus on our SeedGNN model and an NGM-like model. For SeedGNN, we fix the number of layers $L = 2$. The first layer is the same as we described in Section 4. The second layer only uses the output of the percolation module as input (i.e., use $s_2 = z_1$ instead of $s_2 = [m_1, z_1]$). The NGM-like model is the same as SeedGNN except that the NGM-like model adds normalization after the aggregation step (2). Since the node degrees of Erdős-Rényi graphs are highly concentrated around the average degree, we let the NGM-like model divide all the node-pair representations by the square of the average de-

gree. We assume that, after training, the Lipschitz constants of the neural networks $\rho_l$ (in (6)) in NGM are all $K_L$.

We then present the sufficient conditions for the trained SeedGNN and NGM-like model to successfully match all nodes in test graphs. Note that, in the first layer, both SeedGNN and the NGM-like model count the 1-hop witnesses in the same way, except that the NGM-like model further divides the output by the square of the average degree. Thus, the results are proportional to each other, and applying the Hungarian algorithm to these results yields the same matching results $R_1$. Let $\beta$ denote the fraction of correct matches in $R_1$ (which typically increases with the fraction of seeds $\theta$). However, the confidence levels $Z_1$, computed by these two models are quite different because NGM does a normalization step. The difference in confidence levels will influence the performance in the second layer. More precisely, we have the following theorem, with the proofs deferred to Appendix B.

**Theorem 5.1.** *Suppose* $(n_{\text{test}}ps)^2 \geq c_0 K_L \log n_{\text{test}}$. *We have*

- *if* $\beta \geq c_1 \frac{\log n_{\text{test}}}{n_{\text{test}}ps^2}$, *SeedGNN correctly matches all nodes with high probability;*
- *if* $\beta \geq c_2 \max\left\{\frac{\log n_{\text{test}}}{n_{\text{test}}ps^2}, \sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}}s^2}}\right\}$, *the NGM-like model correctly matches all nodes with high probability,*

*where* $c_0, c_1, c_2 > 1$ *are some absolute constants.*

Comparing the two sufficient conditions in Theorem 5.1, we see that, when the graph is dense, i.e., $p \geq \sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}}s^2}}$, we have $\sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}}s^2}} \geq \frac{\log n_{\text{test}}}{n_{\text{test}}ps^2}$ so that $\sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}}s^2}}$ dominates the sufficient condition for the NGM-like model. Therefore, when the test graphs have large sizes and node degrees, the condition of SeedGNN require a much smaller $\beta$ (and thus fewer seeds) than that of NGM. Note that a smaller requirement of $\beta$ translates to a smaller number of initial seeds needed. Thus, given a small set of seeds, SeedGNN may already successfully match all nodes, while NGM still has a low matching accuracy (see Section 6.2).

# 6. Experiments

## 6.1. Experimental Set-up

In our experiment, the number of SeedGNN layers is fixed to 6. We implement the operators $\phi_l$ and $\rho_l$ as two-layer neural networks with $d_l = 16$. For all experiments, optimization is done via ADAM (Kingma & Ba, 2015) with a fixed learning rate of $10^{-2}$. Our model is implemented using PyTorch (Paszke et al., 2019) and trained on an Intel Core i7-8750H CPU. The performance is evaluated using the matching accuracy rate, i.e., the fraction of nodes that are correctly matched. Our code is publicly available at https://github.com/Leron33/SeedGNN.

**Datasets.** We use the correlated Erdős-Rényi graph model (Pedarsani & Grossglauser, 2011), Facebook networks in (Traud et al., 2012), SHREC'16 computer vision dataset in (Lähner et al., 2016), and Willow Object dataset (Cho et al., 2013) in our experiments. We described the correlated Erdős-Rényi graph model earlier in Section 5, and the details of the three real datasets are deferred to Appendix D.1.

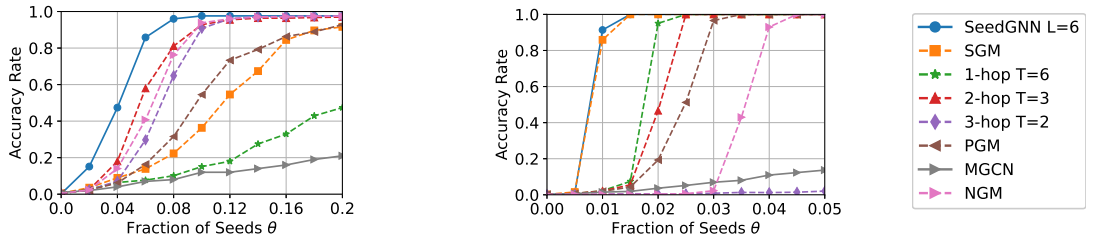
**Training set.** We construct the training set $\mathcal{T}$ in the following way. First, we generate 100 random pairs of correlated Erdős-Rényi graphs with $n = 100$, $p \in \{0.1, 0.3, 0.5\}$, $s \in \{0.6, 0.8, 1\}$, and $\theta = 0.1$. Second, we add 10 pairs of Facebook networks as discussed above with $\theta = 0.1$ into the training set. Third, we do not include any SHREC'16 dataset or Willow Object dataset in the training set. Our SeedGNN trained on the above training set already performs well for these two datasets (see Section 6.2), which verifies the generalization power of our SeedGNN. The training batch size is 64. The overall training for 500 epochs takes about 12 hours and requires 2.68 GB memory.

**Baselines.** We compare the performance of our proposed SeedGNN with several state-of-the-art algorithms: $\boldsymbol{D}$-hop (Mossel & Xu, 2019), PGM (Kazemi et al., 2015), and PLD (Yu et al., 2021b) are theoretical algorithms; SGM (Fishkind et al., 2019) is a convex relaxation algorithm; MGCN (Chen et al., 2020) is a representative semi-supervised learning-based GNN approach; NGM (Wang et al., 2021) is a supervised GNN method for seedless graph matching. We adapt the NGM approach to seeded graph matching by replacing the affinity matrix in NGM with the Kronecker product of the two adjacent matrices and inputting the seed information in the same way as SeedGNN. The details of these baselines are deferred to Appendix D.2.

## 6.2. Results

**SeedGNN requires fewer seeds than existing algorithms to successfully match graphs.** In Figure 6, we show the performance of the algorithms on the correlated Erdős-Rényi graph model. For test graphs, we vary $\theta$ while fixing $n = 500$, $p \in \{0.01, 0.2\}$, $s = 0.8$. We can observe that, among the state-of-the-art methods, the iterative 2-hop algorithm has the best performance for sparse graphs ($p = 0.01$), and the SGM algorithm performs the best for dense graphs ($p = 0.2$). In comparison, our SeedGNN has overall the best performance among all algorithms. Existing theoretical studies such as (Mossel & Xu, 2019) must use witnesses at different numbers of hops, depending on whether the graphs are sparse ($p = 0.01$) or dense ($p = 0.2$). In contrast, our SeedGNN is capable of choosing the right features automatically to match different types of graphs.

We then compare SeedGNN with the state-of-the-art algo-




(a) $p = 0.01$ (b) $p = 0.2$

Figure 6: Performance comparison on correlated Erdős-Rényi graphs. Fix $n = 500$ and $s = 0.8$.

rithms on Facebook networks, which are real-world graphs with an approximate power-law degree distribution. In Figure 7, we can see that SeedGNN is comparable to SGM and significantly outperforms other algorithms. Note that the matching accuracy is saturated at around 80%, because there are about 15% nodes that do not have any common neighbour in $\mathcal{G}_1$ and $\mathcal{G}_2$, and thus can not be correctly matched.

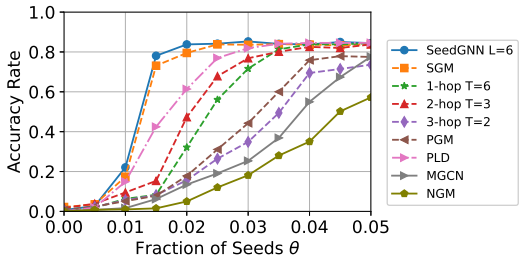



Figure 7: Performance comparison on the Facebook networks with different $\theta$.

**SeedGNN generalizes well to unseen graphs with sizes and types different from the training graphs.** When we test on the correlated Erdős-Rényi graphs in Figure 6, the different graph sizes between the training set ($n = 100$) and the test set ($n = 500$) already demonstrates the generalization power of SeedGNN. To further validate that our SeedGNN can adapt to different graphs, we evaluate SeedGNN for deformable shape matching using the SHREC'16 dataset. Note that the sizes and types of graphs in this dataset are quite different from the Erdős-Rényi and Facebook graphs in the training set. The performance improvement shown in Figure 8 verifies the generalization power of our SeedGNN.

**SeedGNN can be much more effective than semi-supervised learning GNN when non-topological features are not informative.** Existing semi-supervised GNNs rely heavily on high-quality non-topological features (e.g., DeepLink (Zhou et al., 2018), CrossMNA (Chu et al., 2019), MGCN (Chen et al., 2020)). However, in the SHREC'16 dataset, the non-topological node features correspond to 3D coordinates, which do not provide much useful information

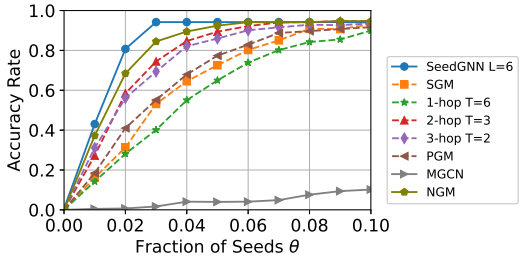



Figure 8: Performance comparison on the SHREC'16 dataset with different $\theta$.

for correlating two 3D shapes with different poses. As a result, in Table 1, we observe that, provided with only a very small fraction of seeds ($\theta = 0.01$), our SeedGNN can significantly outperform the semi-supervised methods.

**Supervised node-based GNNs for seedless graph matching underperform for seeded graph matching.** We compare our SeedGNN with other supervised learning GNN methods on the SHREC'16 dataset. We fix the fraction of seeds $\theta$ at 0.01, use the random encoding to represent seed information, and provide them as input for supervised GNNs (except NGM, for which we described the changes in Section 6.1). In Table 1, we can observe that our SeedGNN significantly outperforms the supervised GNNs, even when the latter are augmented with seed information. This suggests that our method is more effective in using seed information than most supervised seedless GNN methods.

**NGM does not generalize well when the test graph has much larger size and node-degree than the training graphs.** As we discussed in Section 4.2, most existing supervised seedless GNN approaches are node-based and have difficulty learning how to use seed information in a generalizable way. The only exception is NGM in (Wang et al., 2021), which has a similar pair-wise architecture as SeedGNN. Thus, we transfer NGM to the seeded matching version. However, we can observe from Figure 6 that, although NGM performs relatively well (it still underperforms our SeedGNN) in larger sparse graphs ($p = 0.01$), it

Table 1: Comparison of GNN methods on SHREC'16 dataset. The best results are marked as bold.

| Method | Semi-Supervised GNN | | | Supervised GNN | | | | |
|---|---|---|---|---|---|---|---|---|
| | DeepLink (Zhou et al., 2018) | CrossMNA (Chu et al., 2019) | MGCN (Chen et al., 2020) | DGMC (Fey et al., 2020) | BB-GM (Rolínek et al., 2020) | DGM (Gao et al., 2021) | NGM (Wang et al., 2021) | SeedGNN ours |
| Accuracy (%) | $3.3 \pm 0.8$ | $4.2 \pm 1.7$ | $3.8 \pm 1.1$ | $23.2 \pm 6.8$ | $21.1 \pm 4.4$ | $19.31 \pm 10.6$ | $37.9 \pm 5.7$ | $\mathbf{43.1 \pm 8.5}$ |

performs quite poorly in larger dense graphs ($p = 0.2$). This observation confirms our theoretical analysis in Section 5. Note that, for the experiment on the SHREC'16 dataset (Table 1), NGM has similar performance as our SeedGNN. This is because in this experiment, we train NGM also with the SHREC '16 dataset. Thus, the test graphs and training graphs are with similar node degrees. As a result, the issue caused by the normalization operation in NGM is not as evident for the SHREC'16 dataset.

**SeedGNN can also be used in an overall pipeline to refine the outputs of other seedless graph matching algorithms, taking advantage of informative non-topological features.** Although SeedGNN is designed for seeded graph matching and only uses topological information, it can also be integrated into an overall pipeline to utilize non-topological node features that are sufficiently informative. For example, we can first use a seedless graph matching algorithm to generate an initial matching based on non-topological node features. By taking this initial matching as partially-correct seeds, we can then apply SeedGNN to correct these seeds.

To demonstrate this capability, we conduct experiments on the Willow Object dataset, whose node features are informative enough for correlating nodes. We compare the SeedGNN pipeline with several state-of-the-art seedless GNN methods (see Table 2). The performance values of these existing seedless GNN methods are directly retrieved from their respective papers. For the SeedGNN pipeline, we still directly use the model trained in Section 6.1. The input to SeedGNN is generated in two ways. The first way is to apply a neural network only on non-topological features (without graph information) and use the dot product of its outputs on each pair of nodes to generate the similarity for each node-pair (we denote this neural network as **MLP**). Then, **MLP+SeedGNN-1** directly uses the similarity values of all node-pairs as input, while **MLP+SeedGNN-2** further filters out noisy information by applying the Hungarian algorithm on the similarity matrix. The second way (**DGMC+SeedGNN**) is to use the matching result of DGMC, one of the GNN methods for seedless graph matching, as input.

From the matching accuracy presented in Table 2, we make the following observations. First, our results show that SeedGNN can effectively refine the output of other seedless graph matching algorithms. Specifically, both MLP+SeedGNN-1 and MLP-SeedGNN-2 outperform MLP, while DGMC+SeedGNN outperforms DGMC. Second, it is crucial to clean up the output of seedless matching algorithms in order to achieve good performance in the SeedGNN pipeline. This is evident as MLP+SeedGNN-2 consistently outperforms MLP+SeedGNN-1. Indeed, the similarity matrix contains significantly more fake pairs than true pairs. Thus, by using the Hungarian algorithm on the similarity matrix, MLP+SeedGNN-2 sends only high-confidence seeds to SeedGNN, and is more effective in suppressing the misleading information. Third, our experiments show that the effectiveness of the SeedGNN pipeline still depends on the choice of seedless algorithm. Specifically, we observe that DGMC+SeedGNN achieves the best performance, outperforming MLP+SeedGNN and other seedless GNN methods.

Table 2: Comparison of matching accuracy (%) on Willow Object dataset. The best results are marked as bold.

| Method | face | mbike | car | duck | wbottle | Mean |
|---|---|---|---|---|---|---|
| GMN (Zanfir & Sminchisescu, 2018) | 98.1 | 65.0 | 72.9 | 74.3 | 70.5 | 76.2 |
| PCA-GM (Wang et al., 2019) | **100.0** | 76.7 | 84.0 | 93.5 | 96.9 | 90.2 |
| NGM (Wang et al., 2021) | 99.2 | 82.1 | 84.1 | 77.4 | 93.5 | 87.2 |
| IPCA-GM (Wang et al., 2020a) | **100.0** | 77.7 | 90.2 | 84.9 | 95.2 | 89.6 |
| CIE (Yu et al., 2019) | **100.0** | 90.0 | 82.2 | 81.2 | 97.6 | 90.2 |
| DGMC (Fey et al., 2020) | **100.0** | 92.1 | 90.3 | 89.0 | 97.1 | 93.7 |
| BB-GM (Rolínek et al., 2020) | **100.0** | 98.9 | 95.7 | 93.1 | 99.1 | 97.4 |
| DGM (Gao et al., 2021) | **100.0** | 98.8 | **98.0** | 92.8 | 99.0 | 97.7 |
| DLGM (Yu et al., 2021c) | **100.0** | 99.3 | 96.5 | 93.7 | **99.3** | 97.8 |
| MLP | 98.1 | 48.3 | 65.3 | 66.0 | 77.7 | 71.1 |
| MLP+SeedGNN-1 | 99.4 | 77.8 | 84.1 | 77.4 | 89.5 | 85.6 |
| MLP+SeedGNN-2 | **100.0** | 98.9 | 96.8 | 93.1 | 98.7 | 97.5 |
| DGMC+ SeedGNN | **100.0** | **99.6** | 97.4 | **98.7** | 99.0 | **98.9** |

**Additional experiments to study the complexity and inner working of our SeedGNN.** In Appendix C.2, we show that the runtime of our SeedGNN is comparable to other GNN-based algorithms. In Appendix E, we investigate the inner working of our SeedGNN. First, we verify the effectiveness of our design choices for SeedGNN by comparing the performance of different architectural designs. Then, we investigate which sets of samples need to be included in our training set to obtain an effective trained model. Finally, we study the matching process of SeedGNN for different types of graphs. The results verify that SeedGNN chooses the appropriate features for different graphs based on the confidence levels of new seeds as illustrated in Figure 5.

In summary, all of the above experiments demonstrate that our SeedGNN significantly outperforms these baselines across various types of graphs while requiring fewer seeds. Moreover, the knowledge learned by SeedGNN from training graphs can be effectively generalized to test graphs of different sizes and categories.

Limitations and societal impacts are discussed in the Appendix F and G, respectively.

## Acknowledgements

L. Yu and X. Lin are supported in part by the NSF Grants CNS-2113893 and CNS-2225950. J. Xu is supported in part by the NSF Grant CCF-1856424 and an NSF CAREER award CCF-2144593. We would like to thank Prof. Qiang Qiu at Purdue University and the anonymous reviewers for their valuable comments and suggestions on our paper.

## References


Avis, D. A survey of heuristics for the weighted matching problem. *Networks*, 13(4):475–493, 1983.

Barabási, A.-L. et al. *Network Science*. Cambridge University Press, 2016.

Bengio, Y. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

Chen, H., Yin, H., Sun, X., Chen, T., Gabrys, B., and Musial, K. Multi-level graph convolutional networks for cross-platform anchor link prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1503–1511, 2020.

Chiasserini, C.-F., Garetto, M., and Leonardi, E. Social network de-anonymization under scale-free user relations. *IEEE/ACM Transactions on Networking*, 24(6): 3756–3769, 2016.

Cho, M., Alahari, K., and Ponce, J. Learning graphs to match. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 25–32, 2013.

Chu, X., Fan, X., Yao, D., Zhu, Z., Huang, J., and Bi, J. Cross-network embedding for multi-network alignment. In *The world wide web conference*, pp. 273–284, 2019.

Conte, D., Foggia, P., Sansone, C., and Vento, M. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

Crouse, D. F. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

Derr, T., Karimi, H., Liu, X., Xu, J., and Tang, J. Deep adversarial network alignment. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 352–361, 2021.

Dubhashi, D. P. and Panconesi, A. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

Edmonds, J. and Karp, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

Fey, M., Lenssen, J. E., Morris, C., Masci, J., and Kriege, N. M. Deep graph matching consensus. In *International Conference on Learning Representations*, 2020.

Fishkind, D. E., Adali, S., Patsolic, H. G., Meng, L., Singh, D., Lyzinski, V., and Priebe, C. E. Seeded graph matching. *Pattern recognition*, 87:203–215, 2019.

Gao, Q., Wang, F., Xue, N., Yu, J.-G., and Xia, G.-S. Deep graph matching under quadratic constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5069–5078, 2021.

Haghighi, A. D., Ng, A. Y., and Manning, C. D. Robust textual inference via graph matching. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 387–394. Association for Computational Linguistics, 2005.

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

Jiang, B., Sun, P., and Luo, B. Glmnet: Graph learning-matching convolutional networks for feature matching. *Pattern Recognition*, 121:108167, 2022.

Jiang, M., Cui, P., Yuan, N. J., Xie, X., and Yang, S. Little is much: Bridging cross-platform behaviors through overlapped crowds. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 13–19. AAAI Press, 2016.

Kazemi, E., Hassani, S. H., and Grossglauser, M. Growing a graph matching from a handful of seeds. *Proceedings of the VLDB Endowment*, 8(10):1010–1021, 2015.

Kazemi, E., Hassani, H., Grossglauser, M., and Modarres, H. P. Proper: global protein interaction network alignment through percolation matching. *BMC bioinformatics*, 17(1):527, 2016.

Kim, V. G., Lipman, Y., and Funkhouser, T. Blended intrinsic maps. *ACM Transactions on Graphics (TOG)*, 30(4): 1–12, 2011.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Korula, N. and Lattanzi, S. An efficient reconciliation algorithm for social networks. *Proceedings of the VLDB Endowment*, 7(5):377–388, 2014.

Kriege, N. M., Humbeck, L., and Koch, O. Chemical similarity and substructure searches. In *Encyclopedia of Bioinformatics and Computational Biology*, pp. 640 – 649. Academic Press, Oxford, 2019.

Lähner, Z., Rodolà, E., Bronstein, M., Cremers, D., Burghard, O., Cosmo, L., Dieckmann, A., Klein, R., and Sahillioglu, Y. SHREC'16: Matching of deformable shapes with topological noise. *Proc. 3DOR*, 2(10.2312), 2016.

Li, C., Wang, S., Wang, H., Liang, Y., Yu, P. S., Li, Z., and Wang, W. Partially shared adversarial learning for semi-supervised multi-platform user identity linkage. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 249–258, 2019a.

Li, C., Wang, S., Wang, Y., Yu, P., Liang, Y., Liu, Y., and Li, Z. Adversarial learning for weakly-supervised social network alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 996–1003, 2019b.

Li, C.-Y. and Lin, S.-D. Matching users and items across domains to improve the recommendation quality. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 801–810, 2014.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pp. 3835–3845. PMLR, 2019c.

Lu, C.-T., Xie, S., Shao, W., He, L., and Yu, P. S. Item recommendation for emerging online businesses. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 3797–3803. AAAI Press, 2016.

Lubars, J. and Srikant, R. Correcting the output of approximate graph matching algorithms. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1745–1753. IEEE, 2018.

Lyzinski, V., Fishkind, D. E., and Priebe, C. E. Seeded graph matching for correlated Erdös-Rényi graphs. *Journal of Machine Learning Research*, 15(1):3513–3540, 2014.

Mossel, E. and Xu, J. Seeded graph matching via large neighborhood statistics. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1005–1014. SIAM, 2019.

Narayanan, A. and Shmatikov, V. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 111–125. IEEE, 2008.

Narayanan, A. and Shmatikov, V. De-anonymizing social networks. In *Security and Privacy, 2009 30th IEEE Symposium on*, pp. 173–187. IEEE, 2009.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pedarsani, P. and Grossglauser, M. On the privacy of anonymized networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1235–1243. ACM, 2011.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. Deep graph matching via blackbox differentiation of combinatorial solvers. In *16th European Conference on Computer Vision*, pp. 407–424. Springer, 2020.

Sahillioğlu, Y. Recent advances in shape correspondence. *The Visual Computer*, 36(8):1705–1721, 2020.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Schellewald, C. and Schnörr, C. Probabilistic subgraph matching based on convex relaxation. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pp. 171–186. Springer, 2005.

Schmidhuber, J. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

Shirani, F., Garg, S., and Erkip, E. Seeded graph matching: Efficient algorithms and theoretical guarantees. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 253–257. IEEE, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Singh, R., Xu, J., and Berger, B. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008.

Traud, A. L., Mucha, P. J., and Porter, M. A. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

Van Kaick, O., Zhang, H., Hamarneh, G., and Cohen-Or, D. A survey on shape correspondence. In *Computer Graphics Forum*, volume 30, pp. 1681–1707. Wiley Online Library, 2011.

Vento, M. and Foggia, P. Graph matching techniques for computer vision. In *Image Processing: Concepts, Methodologies, Tools, and Applications*, pp. 381–421. IGI Global, 2013.

Vestner, M., Lähner, Z., Boyarski, A., Litany, O., Slossberg, R., Remez, T., Rodola, E., Bronstein, A., Bronstein, M., and Kimmel, R. Efficient deformable shape correspondence via kernel matching. In *2017 International Conference on 3D Vision (3DV)*, pp. 517–526. IEEE, 2017a.

Vestner, M., Litman, R., Rodolà, E., Bronstein, A., and Cremers, D. Product manifold filter: Non-rigid shape correspondence via kernel density estimation in the product space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3327–3336, 2017b.

Wang, R., Yan, J., and Yang, X. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3056–3065, 2019.

Wang, R., Yan, J., and Yang, X. Combinatorial learning of robust deep graph matching: an embedding based approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020a.

Wang, R., Yan, J., and Yang, X. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Wang, T., Liu, H., Li, Y., Jin, Y., Hou, X., and Ling, H. Learning combinatorial solver for graph matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7568–7577, 2020b.

Wen, Z., Fang, Y., and Liu, Z. Meta-inductive node classification across graphs. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1219–1228, 2021.

Yartseva, L. and Grossglauser, M. On the performance of percolation graph matching. In *Proceedings of the first ACM conference on Online social networks*, pp. 119–130. ACM, 2013.

Yu, L., Xu, J., and Lin, X. Graph matching with partially-correct seeds. *Journal of Machine Learning Research*, 22 (280):1–54, 2021a.

Yu, L., Xu, J., and Lin, X. The power of d-hops in matching power-law graphs. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(2), 2021b.

Yu, T., Wang, R., Yan, J., and Li, B. Learning deep graph matching with channel-independent embedding and hungarian attention. In *International conference on learning representations*, 2019.

Yu, T., Wang, R., Yan, J., and Li, B. Deep latent graph matching. In *International Conference on Machine Learning*, pp. 12187–12197. PMLR, 2021c.

Zafarani, R., Tang, L., and Liu, H. User identification across social media. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(2):1–30, 2015.

Zanfir, A. and Sminchisescu, C. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2684–2693, 2018.

Zhang, J. and Philip, S. Y. Multiple anonymized social networks alignment. In *2015 IEEE International Conference on Data Mining*, pp. 599–608. IEEE, 2015.

Zhang, W., Shu, K., Liu, H., and Wang, Y. Graph neural networks for user identity linkage. *arXiv preprint arXiv:1903.02174*, 2019.

Zhang, Y., Tang, J., Yang, Z., Pei, J., and Yu, P. S. Cosnet: Connecting heterogeneous social networks with local and global consistency. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1485–1494, 2015.

Zhou, F., Liu, L., Zhang, K., Trajcevski, G., Wu, J., and Zhong, T. Deeplink: A deep learning approach for user identity linkage. In *IEEE INFOCOM 2018-IEEE conference on computer communications*, pp. 1313–1321. IEEE, 2018.

Zhou, F., Wen, Z., Trajcevski, G., Zhang, K., Zhong, T., and Liu, F. Disentangled network alignment with matching explainability. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1360–1368. IEEE, 2019.

## A. Additional Discussion on Related Work

**Theoretical Algorithms** Existing theoretical algorithms suffer from several limitations. First, graphs with different characteristics may require different features and carefully tuned parameters. For instance, the Percolation algorithm in (Kazemi et al., 2015) needs to carefully choose a threshold parameter to achieve good performance, and the PLD algorithm in (Yu et al., 2021b) is designed for graphs with power-law degree distributions. It is cumbersome to design a new algorithm and/or tune the parameters every time when a new type of graph is processed. In contrast, by learning from the training graphs, our SeedGNN can automatically choose the effective features. Second, these theoretical algorithms may not synthesize different features most effectively. For instance, the $l$-hop algorithm in (Mossel & Xu, 2019) only utilizes the witnesses at the $l$-th hop, but does not combine witnesses at different hops.

**Inductive Semi-supervised Learning on Graphs** Our goal of using supervised learning for seeded graph matching shares some similarities with the work in (Wen et al., 2021), which also aims to both perform inductive learning (i.e., learn transferrable knowledge from training graphs) and utilize a small amount of labeled data on the test graph. However, (Wen et al., 2021) focuses on a node classification problem, which is quite different from seeded graph matching. In particular, (Wen et al., 2021) uses node-based GNNs, which (as we discussed in Section 4.1) have more difficulty in effectively utilizing seed information than our proposed pair-wise GNN. Further, in order to transfer knowledge from the trained GNN to test graphs, (Wen et al., 2021) scales all GNN weights by a common factor. It is unclear how this scaling will effectively transfer knowledge for seeded graph matching, e.g., how to best use different hops of witnesses. In contrast, our design of SeedGNN exploits the inherent structure of the seeded graph matching problem, and can be shown to generalize well to unseen graphs of sizes and types very different from the training set. For future work, it would be of interest to explore whether our SeedGNN can be further improved with a meta-learning component (Santoro et al., 2016).

**Convex Relaxation Algorithms** In addition to the theoretical algorithms and the GNN approaches, there is another class of algorithms based on convex relaxations of the quadratic assignment problem, which maximizes the total number of matched edges between two graphs subject to the seed constraint (Lyzinski et al., 2014; Fishkind et al., 2019). In (Fishkind et al., 2019), the authors describe a gradient ascent approach to solve this relaxed problem, which is called SGM. Compared to SeedGNN, SGM also has flavors of using witnesses and percolation ideas. Specifically, the gradient of the SGM algorithm is similar to a matrix counting 1-hop witnesses. However, using only 1-hop witnesses is known to be ineffective in sparse graphs (as there are very few 1-hop witnesses even for true pairs). Indeed, our experiments in Section 6 find that our SeedGNN often outperforms SGM, especially in sparse graphs.

**Differences between Our Percolation Module and Previous Percolation Algorithms** Unlike previous percolation algorithms (Yartseva & Grossglauser, 2013), we allow SeedGNN to correct errors from earlier layers by re-matching nodes at each layer. Note that in many percolation algorithms, once a new pair of seeds is identified, it will be used as the correct match until the end. This approach can be problematic if an incorrect pair is identified as a seed, whose impact will be lasting for many iterations down the road. In contrast, since our SeedGNN rematches nodes at each layer, even if some of the newly-identified seeds in the previous layer are incorrect, we can correct these errors in the next layer, as long as the fraction of incorrect seeds is small. In other words, our design of SeedGNN takes advantage of the power of partially correct (i.e., noisy) seeds (as theoretically verified in (Lubars & Srikant, 2018; Yu et al., 2021a)).

## B. Proof of Theorem 5.1

**Notation** We use $\dot{\sim}$ to denote "approximately distributed".

Note that the first layers of SeedGNN and the NGM-like model output the same set of new seeds ($\beta$ fraction of which are correct). However, their confidence levels are very different. Specifically, after softmax in (7), correctly matched pairs will have confidence levels close to 1, while incorrectly matched pairs will only have confidence levels close to $\frac{1}{n_{\text{test}}}$. In contrast, since the NGM-like model divides the node-pair representations by the square of the average degree, the resulting value ($\sim \frac{1}{{n_{\text{test}}}^2}$) will decrease close to zero as $n_{\text{test}}$ increases. After we apply softmax, the confidence levels of correct matching and incorrect matching will both become close to $\frac{1}{n_{\text{test}}}$. More precisely, we have the following lemma, with the proof deferred to Appendix B.1.

**Lemma B.1.** *Assume that $(n_{\text{test}}ps)^2 \geq c_0 K_L \log n_{\text{test}}$ for a sufficiently large constant $c_0$. In the first layer of the NGM-like model, for any $(i, j)$, we have $\frac{1-\delta}{n_{\text{test}}} \leq Y_1(i, j) \leq \frac{1+\delta}{n_{\text{test}}}$ with high probability, where $\delta$ is some sufficiently small constant.*

Since the NGM-like model has lower confidence levels for the correctly-matched new seeds than SeedGNN, it is more difficult for the NGM-like model to match all nodes correctly in the second layer. We present the sufficient conditions for the second layer of SeedGNN and NGM-like model to correctly match all nodes. The proofs of these theorems are deferred to Appendix B.1.

**Theorem B.2.** *If $\beta \geq c_1 \frac{\log n_{\text{test}}}{n_{\text{test}} ps^2}$, SeedGNN correctly matches all nodes with high probability.*

**Theorem B.3.** *If $(n_{\text{test}} ps)^2 \geq c_0 K_L \log n_{\text{test}}$ and $\beta \geq c_2 \max\left\{\frac{\log n_{\text{test}}}{n_{\text{test}} ps^2}, \sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}} s^2}}\right\}$, the NGM-like model correctly matches all nodes with high probability.*

### B.1. Postponed Proofs

*Proofs of Lemma B.1.* Recall that the first layer of the NGM-like model counts the 1-hop witnesses and divides the value by the square of average degree. Thus, in testing, we have, for any node-pair $(i, j)$,

$$H_1(i,j) = \frac{1}{(n_{\text{test}} ps)^2} \cdot \sum_{(u,v)\in\mathcal{S}} A_1(i,u) A_2(v,j).$$

For any initial seed $(u, v)$, $A_1(i, u)A_2(v, j)$ is equal to 1 with probability $ps^2$ if $j = \pi(i)$, and $A_1(i, u)A_2(v, j)$ is equal to 1 with probability $p^2 s^2$ if $j \neq \pi(i)$. Since there are $\theta$ fraction of seeds, $H_1(i, j)$ follows the distribution given by

$$H_1(i,j) \dot{\sim} \begin{cases} \frac{1}{(n_{\text{test}} ps)^2} \text{Binom}\left(n_{\text{test}}\theta, ps^2\right) & \text{if } j = \pi(i), \\ \frac{1}{(n_{\text{test}} ps)^2} \text{Binom}(n_{\text{test}}\theta, p^2 s^2) & \text{if } j \neq \pi(i). \end{cases}$$

By Bernstein's Inequality (Dubhashi & Panconesi, 2009), the upper bound of $H_1(i, j)$ is

$$\begin{aligned} &\mathbb{P}\left\{H_1(i,j) > \frac{n_{\text{test}}\theta ps^2 + \sqrt{6 n_{\text{test}}\theta ps^2 \log n_{\text{test}}} + 2\log n_{\text{test}}}{(n_{\text{test}} ps)^2}\right\} \\ \leq &\mathbb{P}\left\{\text{Binom}\left(n_{\text{test}}\theta, ps^2\right) > n_{\text{test}}\theta ps^2 + \sqrt{6 n_{\text{test}}\theta ps^2 \log n_{\text{test}}} + 2\log n_{\text{test}}\right\} \\ \leq &\exp\left(-3\log n_{\text{test}}\right) < {n_{\text{test}}}^{-3}. \end{aligned}$$

Recall that we apply a neural network $\rho_1$ on each element of $H_1$ to get $X_1 = \rho_1(H_1)$, and the Lipschitz constant of $\rho_1$ is $K_L$. Since $H_1(i, j) \geq 0$ for any $(i, j)$, we have

$$\max_{(i,j),(u,v)} |X_1(i,j) - X_1(u,v)| \leq K_L \max_{(i,j),(u,v)} |H_1(i,j) - H_1(u,v)| \leq K_L \max_{(i,j)} H_1(i,j).$$

Thus, we let $\frac{\delta}{2} = K_L \frac{n_{\text{test}}\theta ps^2 + \sqrt{6 n_{\text{test}}\theta ps^2 \log n_{\text{test}}} + 2\log n_{\text{test}}}{(n_{\text{test}} ps)^2}$ and have

$$\begin{aligned} \mathbb{P}\left\{\max_{(i,j),(u,v)} |X_1(i,j) - X_1(u,v)| > \frac{\delta}{2}\right\} \leq &\mathbb{P}\left\{K_L \max_{(i,j)} H_1(i,j) > \frac{\delta}{2}\right\} \\ \leq &\mathbb{P}\left\{\bigcup_{(i,j)} \{K_L H_1(i,j) > \frac{\delta}{2}\}\right\} \\ \leq &{n_{\text{test}}}^2 \mathbb{P}\left\{H_1(i,j) > \frac{n_{\text{test}}\theta ps^2 + \sqrt{6 n_{\text{test}}\theta ps^2 \log n_{\text{test}}} + 2\log n_{\text{test}}}{(n_{\text{test}} ps)^2}\right\} \\ \leq &{n_{\text{test}}}^{-1}. \end{aligned} \tag{10}$$

Since $n_{\text{test}} p \geq \theta$ and $(n_{\text{test}} ps)^2 \geq c_0 K_L \log n_{\text{test}}$, $\frac{\delta}{2}$ can be made to be sufficiently small and $1 \leq \exp(\frac{\delta}{2}) < 1 + \delta$. It follows taht

$$\frac{\exp(\frac{\delta}{2})}{\exp(\frac{\delta}{2}) + (n_{\text{test}} - 1)\exp(0)} \leq \frac{1+\delta}{n_{\text{test}}} \quad \text{and} \quad \frac{\exp(0)}{\exp(\frac{\delta}{2}) + (n_{\text{test}} - 1)\exp(0)} \geq \frac{1}{n_{\text{test}} + \delta} \geq \frac{1-\delta}{n_{\text{test}}}.$$

We then apply row-wise softmax on $X_1$ to get the confidence level $Y_1$. We can bounded $Y_1$ by the difference between elements in $X_1$:

$$\begin{aligned}\max_{(i,j)} Y_1(i,j) \leq & \frac{\exp(\max_{(i,j)} X_1(i,j))}{\exp(\max_{(i,j)} X_1(i,j)) + (n_{\text{test}}-1)\exp(\min_{(i,j)} X_1(i,j))} \\ = & \frac{\exp(\max_{(i,j)} X_1(i,j) - \min_{(i,j)} X_1(i,j))}{\exp(\max_{(i,j)} X_1(i,j) - \min_{(i,j)} X_1(i,j)) + (n_{\text{test}}-1)\exp(0)},\end{aligned}$$

and similarly

$$\min_{(i,j)} Y_1(i,j) \geq \frac{\exp(0)}{\exp(\max_{(i,j)} X_1(i,j) - \min_{(i,j)} X_1(i,j)) + (n_{\text{test}}-1)\exp(0)}.$$

Since the difference in $X_1$ is upper bounded in (10), we have

$$\begin{aligned}\mathbb{P}\left\{\max_{(i,j)} Y_1(i,j) \leq \frac{1+\delta}{n_{\text{test}}}\right\} \geq & \mathbb{P}\left\{\max_{(i,j)} Y_1(i,j) \leq \frac{\exp(\frac{\delta}{2})}{\exp(\frac{\delta}{2}) + (n_{\text{test}}-1)\exp(0)}\right\} \\ \geq & 1 - \mathbb{P}\left\{\max_{(i,j),(u,v)} |X_1(i,j) - X_1(u,v)| > \frac{\delta}{2}\right\} \\ \geq & 1 - {n_{\text{test}}}^{-1},\end{aligned}$$

$$\begin{aligned}\mathbb{P}\left\{\min_{(i,j)} Y_1(i,j) \geq \frac{1-\delta}{n_{\text{test}}}\right\} \geq & \mathbb{P}\left\{\min_{(i,j)} Y_1(i,j) \geq \frac{\exp(0)}{\exp(\frac{\delta}{2}) + (n_{\text{test}}-1)\exp(0)}\right\} \\ \geq & 1 - \mathbb{P}\left\{\max_{(i,j),(u,v)} |X_1(i,j) - X_1(u,v)| > \frac{\delta}{2}\right\} \\ \geq & 1 - {n_{\text{test}}}^{-1}.\end{aligned}$$

□

*Proof of Theorem B.2.* Recall that the second layer of SeedGNN aggregates over node-pair's neighborhoods in (3). Then, we have, for any node-pair $(i,j)$,

$$H_2(i,j) = \sum_{(u,v):u,v\in[n_{\text{test}}]} A_1(i,u)A_2(v,j)S_2(u,v).$$

In our analysis, we use the output of percolation module in the first layer as input (i.e. $S_2 = Z_1$). Note that $Z_1$ is the "cleaned" confidence levels. There are only $n_1$ non-zeros elements in $Z_1$ representing the confidence levels of new seeds. Among $n_1$ new seeds, $\beta$ fraction of them are correctly matched seeds, and the rest are incorrectly matched seeds. Since correct seeds have much higher confidence levels than incorrect seeds, after softmax normalization in (7), correct seeds have 1 confidence level and incorrect seeds have $\frac{1}{n_{\text{test}}}$ confidence level. Then, We have

$$H_2(i,j) = \sum_{\substack{(u,v):Z_1(u,v)>0,\\ v=\pi(u)}} A_1(i,u)A_2(v,j) + \frac{1}{n_{\text{test}}} \cdot \sum_{\substack{(u,v):Z_1(u,v)>0,\\ v\neq\pi(u)}} A_1(i,u)A_2(v,j).$$

For any correct seed, $A_1(i,u)A_2(v,j)$ is equal to 1 with probability $ps^2$ if $j=\pi(i)$, and $A_1(i,u)A_2(v,j)$ is equal to 1 with probability $p^2s^2$ if $j\neq\pi(i)$. In contrast, for any incorrect seed, $A_1(i,u)A_2(v,j)$ is equal to 1 with probability $p^2s^2$ for any node-pair $(i,j)$. Thus, it follows that

$$H_2(i,j) \dot{\sim} \begin{cases} \text{Binom}\left(n_{\text{test}}\beta, ps^2\right) + \frac{1}{n_{\text{test}}}\text{Binom}\left(n_{\text{test}}(1-\beta), p^2s^2\right) & \text{if } j=\pi(i), \\ \text{Binom}(n_{\text{test}}\beta, p^2s^2) + \frac{1}{n_{\text{test}}}\text{Binom}\left(n_{\text{test}}(1-\beta), p^2s^2\right) & \text{if } j\neq\pi(i). \end{cases}$$

Since the second terms are both no greater than 1, $H_2(i,j)$ is dominated by the first terms in the right-hand-side (which are contributed by the correct seeds). If $H_2(i,j)$ of any true pair is greater than $H_2(i,j)$ of any fake pair, all the true pairs can be distinguished from fake pairs. By Theorem 1 in (Korula & Lattanzi, 2014), if $\beta \geq c_1 \frac{\log n_{\text{test}}}{n_{\text{test}} p s^2}$ with a sufficiently large constant $c_1$, all nodes can be correctly matched with high probability.

□

*Proof of Theorem B.3.* In the NGM-like model, the second layer aggregates over node-pair's neighborhoods and divides the results by the square of average degree. Thus, we have, for any node-pair $(i,j)$,

$$H_2(i,j) = \frac{1}{(n_{\text{test}} p s)^2} \sum_{(u,v): u \in [n_1], v \in [n_2]} A_1(i,u) A_2(v,j) S_2(u,v).$$

We use the "cleaned" matching as input (i.e., $S_2 = Z_1$), and there are only $n_1$ non-zeros elements in $Z_1$ representing the confidence levels of new seeds. Among the $n_1$ new seeds, $\beta$ fraction of them are correctly matched seeds, and the rest are incorrectly matched seeds. By Lemma B.1, the confidence levels of all new seeds are close to $\frac{1}{n_{\text{test}}}$ with high probability. Then, We have

$$H_2(i,j) = \frac{1}{n_{\text{test}} (n_{\text{test}} p s)^2} \cdot \left( \sum_{\substack{(u,v): Z_1(u,v) > 0, \\ v = \pi(u)}} A_1(i,u) A_2(v,j) + \sum_{\substack{(u,v): Z_1(u,v) > 0, \\ v \neq \pi(u)}} A_1(i,u) A_2(v,j) \right).$$

Analogous to the analysis in the proof of Theorem B.2, $H_2(i,j)$ follows the distribution given by

$$H_2(i,j) \dot{\sim} \begin{cases} \frac{1}{n_{\text{test}} (n_{\text{test}} p s)^2} \left( \text{Binom}\left(n_{\text{test}} \beta, p s^2\right) + \text{Binom}\left(n_{\text{test}} (1-\beta), p^2 s^2\right) \right) & \text{if } j = \pi(i), \\ \frac{1}{n_{\text{test}} (n_{\text{test}} p s)^2} \left( \text{Binom}(n_{\text{test}} \beta, p^2 s^2) + \text{Binom}\left(n_{\text{test}} (1-\beta), p^2 s^2\right) \right) & \text{if } j \neq \pi(i). \end{cases}$$

If $H_2(i,j)$ of any true pair is greater than $H_2(i,j)$ of any fake pair, all the true pairs can be distinguished from fake pairs by the NGM-like model. Note that we need to consider the influence of incorrect seeds. By Theorem 1 in (Yu et al., 2021a), if $\beta \geq c_2 \max\left\{ \frac{\log n_{\text{test}}}{n_{\text{test}} p s^2}, \sqrt{\frac{\log n_{\text{test}}}{n_{\text{test}} s^2}} \right\}$ with a sufficiently large constant $c_2$, all nodes can be correctly matched with high probability. □

## C. Complexity and Scalability

### C.1. Time and Space Complexity

First, we analyze the computational complexity of our SeedGNN. In each layer, $(A_1 \otimes A_2) s_l$ in (2) can be converted into $A_1 S_l A_2$ as shown in (3). When we represent $A_1$ and $A_2$ with sparse matrices, each of them only contain $n_1 d_{\text{mean}}$ and $n_2 d_{\text{mean}}$ elements, where $d_{\text{mean}}$ is the mean of the node degrees of $\mathcal{G}_1$ and $\mathcal{G}_2$. Thus, by sparse matrix multiplication, the time complexity of Equation (3) is $O(n_1 n_2 d_{\text{mean}})$. The neural networks (4) and (6) take $O(n_1 n_2)$ time. The Hungarian algorithm takes $O(n_1 n_2^2)$ times (Crouse, 2016). Thus, the total time complexity is $O(n_1 n_2^2)$.

The space complexity of our SeedGNN is $O(n_1 n_2)$, since $A_1$ and $A_2$ are sparse matrices, and $S_l$ has $n_1 n_2$ elements.

### C.2. Making SeedGNN more Scalable

For very large graphs, the step of the Hungarian algorithm becomes the computational bottleneck. We can use greedy max-weight matching (GMWM) in (Avis, 1983) instead, as the time complexity of GMWM is only $O(n_1 n_2 \log n_2)$. With this improvement, the total time complexity is reduced to $O(n_1 n_2 \log n_2 + n_1 n_2 d_{\text{mean}})$. To the best of our knowledge, the best-known time complexity for GNN-based algorithms is $O(n_1 n_2)$ (Fey et al., 2020). Thus, the computational complexity of our SeedGNN is only moderately larger than the best-known one. In Table 3, we show the average run time of GNN-based algorithm to match a pair of large graphs on SHREC'16 dataset (with 8K-11K nodes). The semi-supervised methods

(DeepLink (Zhou et al., 2018), CrossMNA (Chu et al., 2019), MGCN (Chen et al., 2020)) are provided with a fraction of seeds ($\theta = 0.01$) as the training set, and the run time includes the training and test time. The supervised methods (DGMC (Fey et al., 2020), BB-GM (Rolínek et al., 2020), DGM (Gao et al., 2021), NGM (Wang et al., 2021)) utilize only non-topological features but not seeds, and the run time is only is only for a pair of test graphs. The results demonstrate that the run time of our SeedGNN is comparable to the best-known GNN-based algorithms.

Table 3: Run time Comparison of GNN methods on SHREC'16 dataset.

| Method | Semi-Supervised | | | Supervised | | | | |
|---|---|---|---|---|---|---|---|---|
| | DeepLink Zhou et al. (2018) | CrossMNA Chu et al. (2019) | MGCN Chen et al. (2020) | DGMC Fey et al. (2020) | BB-GM Rolínek et al. (2020) | DGM Gao et al. (2021) | NGM Wang et al. (2021) | SeedGNN ours |
| run time (s) | 1847.0 | 2321.9 | 3573.4 | 80.2 | 130.1 | 211.3 | 879.2 | 141.5 |

## D. Details of Experiments

### D.1. Datasets

We give detailed descriptions of the real datasets used in our experiments.

**Facebook networks** The dataset in (Traud et al., 2012) provides 100 Facebook networks from different institutions. We randomly choose 10 for training and 90 for testing. The sizes of the Facebook networks range from 962 to 32361. To lower the training cost, we down-sample the sizes of the training graphs. Specifically, for each Facebook network for training, we first down-sample nodes with probability $0.25$ to get the parent graph $\mathcal{G}_0$. However, for testing, we do not perform this down-sampling and use the original graphs directly as the parent graph $\mathcal{G}_0$. For both training and testing, we generate $\mathcal{G}_1$ and $\mathcal{G}_2$ from $\mathcal{G}_0$ by independently sub-sampling each edge of $\mathcal{G}_0$ twice with probability $s = 0.8$ and sub-sampling each node of $\mathcal{G}_0$ twice with probability $0.9$. The nodes of $\mathcal{G}_2$ are then relabeled according to a random permutation $\pi$. Then, each true pair is independently added into the seed set $\mathcal{S}$ with probability $\theta$.

**The SHREC'16 dataset** Matching 3D deformable shapes is a central problem in computer vision, and has been extensively studied for decades (see (Van Kaick et al., 2011) and (Sahillioğlu, 2020) for surveys). The SHREC'16 dataset in (Lähner et al., 2016) provides 25 deformable 3D shapes (15 for training and 10 for testing) undergoing different topological changes. Each shape is represented by a triangulated mesh graph consisting of around 8K-11K nodes (with 3D coordinates).

**Willow Object dataset** Willow Object dataset (Cho et al., 2013) contains at least 40 images for each of its five categories. Following the experimental setups in (Fey et al., 2020), we construct graphs via the Delaunay triangulation of keypoints, and each image consists of exactly 10 labeled keypoints. The features of the keypoints are given by the concatenated output of relu4_2 and relu5_1 of a pre-trained VGG16 (Simonyan & Zisserman, 2015).

### D.2. Details of Baselines

**1) $D$-hop** (Mossel & Xu, 2019) finds the node mapping between the two graphs that maximizes the total number of $D$-hop witnesses for a given $D$. For a fair comparison with other algorithms, we iteratively apply the $D$-hop algorithm $T$ times (with $DT = 6$ because SeedGNN is fixed to have 6 layers). In each iteration, we use the matching result of the previous iteration as new seeds and apply the $D$-hop algorithm again. **2) PGM** (Kazemi et al., 2015) iteratively matches node-pairs with at least $r$ witnesses. We choose $r = 2$, which is the same as the simulation setting in (Kazemi et al., 2015). **3) PLD** (Yu et al., 2021b) is the state-of-the-art seeded graph matching algorithm designed for graphs with power-law degree distributions (which is a common feature of real-world social networks (Barabási et al., 2016)). **4) SGM** (Fishkind et al., 2019) uses Frank–Wolfe method to approximately solves a quadratic assignment problem that maximizes the number of matched edges between two graphs, while being consistent with the given seeds. **5) MGCN** (Chen et al., 2020) is a representative semi-supervised learning-based GNN approach, whose performance is comparable with other semi-supervised learning approaches. The parameters are set in the same way as those in (Chen et al., 2020). **6) NGM** (Wang et al., 2021) is a supervised GNN method for seedless graph matching, but it also uses a pair-wise GNN that utilizes an affinity matrix as input. We transfer this approach to seeded graph matching by replacing the affinity matrix in NGM by the Kronecker product of the two adjacent matrices, and inputting the seed information as SeedGNN. We then train the weights of NGM with the same training set as our SeedGNN.

## E. Studying the Inner-working of SeedGNN

In this section, we further investigate how the performance of SeedGNN varies as we change its inner working. First, to verify the effectiveness of our design choices for our SeedGNN method, we compare the performance of different architectural designs. Then, we investigate which sets of samples need to be included in our training set to obtain an effective trained model. Finally, we study the matching process of SeedGNN for different types of graphs. The results suggest that SeedGNN could potentially choose the appropriate features for different graphs based on the confidence levels of new seeds.

### E.1. Study of the Design Choices

To verify the effectiveness of our design choices, we consider four variants of SeedGNN, which are:

1. **SeedGNN-x:** SeedGNN without convolution module. This variant aims to verify the importance of extracting witness-like information at a larger number of hops.
2. **SeedGNN-w:** SeedGNN without percolation module. This variant aims to verify the importance of the percolation module in SeedGNN.
3. **SeedGNN-p:** SeedGNN with percolation module but without the Hungarian matching algorithm (i.e., $z_l = \mathsf{unflatten}(Y_l)$ in each GNN layer). This variant aims to verify the importance of the "cleaning" process in SeedGNN.
4. **SeedGNN-h:** SeedGNN with $z_l = \mathsf{unflatten}(R_l)$ instead of (8) in each layer. This variant aims to verify that among the new seeds, it is still important to distinguish the high-confidence one and low-confidence one.

Finally, we use "SeedGNN" to denote the full design in Fig. 3. We train all these variants with the same training set $\mathcal{T}$ in Section 6.1.

In Figure 9, we show the performance of the above variants of SeedGNN on correlated Erdős-Rényi graph model. For test graphs, we increase $\theta$ from 0 to 0.05 while fixing $n = 500$, $p = 0.04$, $s = 0.8$. As illustrated in Figure 9, our SeedGNN with full design achieves the best performance among all variants, which shows the effectiveness of our design choices for the SeedGNN architecture. Further, among the variants, SeedGNN-w almost fails completely, which highlights the significant importance of using the percolation idea in SeedGNN for seeded graph matching. SeedGNNx does performs poorly, which demonstrates that it is also important to extract witness-like information at a larger number of hops instead of only 1-hop. We can observe that SeedGNN and SeedGNN-h both outperform SeedGNN-p and the improvement of SeedGNN is significantly bigger. This result verifies that it is not enough to only use the soft-correspondence (as in SeedGNN-p), and we need to combine both the matching result $R_l$ of the Hungarian algorithm and the similarity $Y_l$ as in (8) to achieve the best performance.

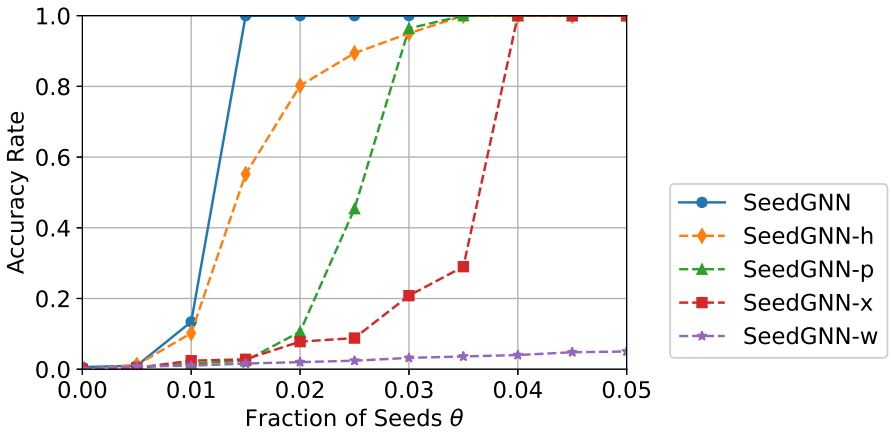


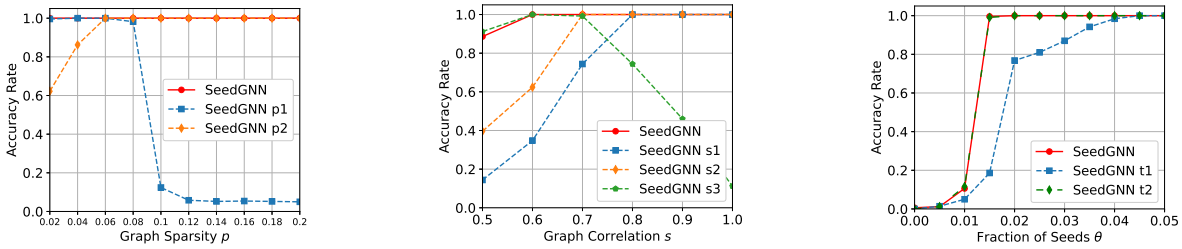

Figure 9: Performance comparison of our SeedGNN and four other variants on correlated Erdős-Rényi graph model with different $\theta$. Fix $n = 500$, $p = 0.04$, $s = 0.8$.

### E.2. Study of the Necessary Training Samples for Generalization

Intuitively, in order to help our SeedGNN successfully learn useful knowledge that can be applied to never-seen graphs, the training set needs to contain graph pairs with different varieties, e.g., graph sparsity, graph correlation, and the size of seed set. However, a larger training set also increases the training time. To show which sets of graph pairs are necessary, we compare SeedGNN trained with different training sets, whose parameters are shown in Table 4. We use $\mathcal{T}$ to denote the training set that only includes the Erdős-Rényi graphs of the training set in Section 6.1. First, to show the necessity of training graph pairs with a wide range of sparsity, we train SeedGNN with $\mathcal{T}$, $\mathcal{T}_{p1}$ and $\mathcal{T}_{p2}$, and compare the performance of the trained models while increasing $p$ from 0.02 to 0.2 and fixing $n = 500$, $s = 0.8$ and $\theta = 0.05$. Figure 10(a) shows that, if SeedGNN is only trained with $p = 0.1$, it performs well on sparse graphs but poorly on dense graphs. In contrast, if SeedGNN is only trained with $p = 0.5$, it performs well on dense graphs but poorly on sparse graphs. Thus, we should include both $p = 0.1$ and $p = 0.5$ in the training set to achieve good performance. Second, to show the necessity of training graph pairs with different correlations, we compare the performance of SeedGNN trained with $\mathcal{T}$, $\mathcal{T}_{s1}$, $\mathcal{T}_{s2}$ and $\mathcal{T}_{s3}$, and compare these models while increasing $s$ from 0.5 to 1 and fixing $n = 500$, $p = 0.08$ and $\theta = 0.05$. Figure 10(b) shows that, if SeedGNN is only trained with $s = 0.6$, it performs well on moderately correlated graphs but poorly on highly correlated graphs. In contrast, if SeedGNN is only trained with $s = 0.8$ or $s = 1$, it performs well on highly correlated graphs but poorly on moderately correlated graphs. Thus, we should include different correlations in the training set to achieve good performance. Third, we compare the performance of SeedGNN trained with $\mathcal{T}$, $\mathcal{T}_{t1}$ and $\mathcal{T}_{t2}$, and compare these models while increasing $\theta$ from 0 to 0.05 and fixing $n = 500$, $p = 0.04$ and $s = 0.8$. Figure 10(c) shows that, if SeedGNN is only trained with $\theta = 0.1$ and $\theta \in \{0.1, 0.3\}$, it performs exactly the same. If SeedGNN is only trained with $\theta = 0.3$, it performs worse than the former two. Thus, we only need to include graph pairs with a relatively small seed set in the training set.

Table 4: Different Training Sets

| Training Sets | $p$ | $s$ | $\theta$ |
|---|---|---|---|
| $\mathcal{T}_{p1}$ | $\{0.1\}$ | $\{0.6, 0.8, 1\}$ | $\{0.05, 0.1\}$ |
| $\mathcal{T}_{p2}$ | $\{0.5\}$ | $\{0.6, 0.8, 1\}$ | $\{0.05, 0.1\}$ |
| $\mathcal{T}_{s1}$ | $\{0.1, 0.5\}$ | $\{1\}$ | $\{0.05, 0.1\}$ |
| $\mathcal{T}_{s2}$ | $\{0.1, 0.5\}$ | $\{0.8\}$ | $\{0.05, 0.1\}$ |
| $\mathcal{T}_{s3}$ | $\{0.1, 0.5\}$ | $\{0.6\}$ | $\{0.05, 0.1\}$ |
| $\mathcal{T}_{t1}$ | $\{0.1, 0.5\}$ | $\{0.6, 0.8, 1\}$ | $\{0.3\}$ |
| $\mathcal{T}_{t2}$ | $\{0.1, 0.5\}$ | $\{0.6, 0.8, 1\}$ | $\{0.1, 0.3\}$ |




(a) $s = 0.8,\ \theta = 0.05$. (b) $p = 0.08,\ \theta = 0.05$. (c) $p = 0.04,\ s = 0.8$.

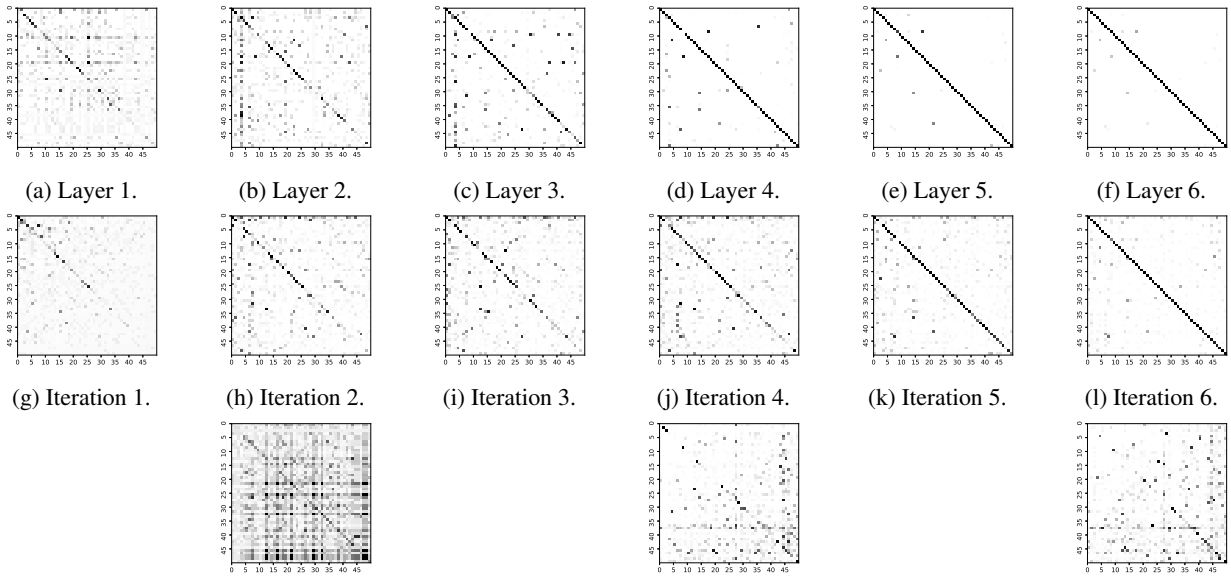
Figure 10: Performance comparison of SeedGNN trained with different training sets. Fix $n = 500$.

### E.3. Layer-wise Study of SeedGNN

Recall from Section 4.3 that our design on the feature combination potentially enables SeedGNN to utilize various types of witness-like information adaptively, based on the confidence levels of new seeds $z_l$. In this section, we verify this capability through numerical results. To directly visualize $z_l$ in the matching process, we present the similarity matrix $Y_l$ of each layer of SeedGNN and compare it with the witness matrix of the iterative 1-hop and 2-hop algorithms at each iteration. We assume that the true mapping $\pi$ is the identity permutation, i.e., $\pi(i) = i$.

First, we study the matching process in *dense* graphs. We fix a pair of correlated Erdős-Rényi graphs with $n = 50$, $p = 0.4$,

$s = 0.8$ and $\theta = 0.1$. Then, we index the nodes from 0 to 49 in the descending order of the node degree in the parent graph $\mathcal{G}_0$. In Figure 11, we show the similarity matrix $Y_l$ in each layer of our SeedGNN, and compare it with the witness matrix in each iteration using either the 1-hop or 2-hop algorithm. We can immediately see that the similarity matrices provided by SeedGNN are more similar to the witness matrices of the iterative 1-hop algorithm than that of the iterative 2-hop algorithm. Specifically, since the graphs are dense, the 1-hop witness from the initial seeds can already generate new seeds with high confidence levels (see Figure 11(a) and 11(g), where there are many dark points on the diagonal (i.e., consistent with the underlying true mapping), while there are few dark points off the diagonal). The iterative 1-hop algorithm is known to use new 1-hop witnesses from these new seeds (see Figure 11(h)) in the next iteration. In contrast, the 2-hop witnesses from the initial seeds are much noisier (see Figure 11(m), where the darkness of the points on the diagonal cannot be differentiated from those off the diagonal). As we illustrated in Figure 5, these two types of witness-like information are both contained in the second layer of SeedGNN. By comparing Figure 11(b) with Figure 11(h) and Figure 11(m), we can observe that the second layer of SeedGNN produces a similarity matrix that is closer to the witness matrix of the 1-hop algorithm than that of the 2-hop algorithm. Thus, we infer that, for these dense graphs in which the new seeds are reliable, the SeedGNN relies more on witnesses computed from these new seeds.



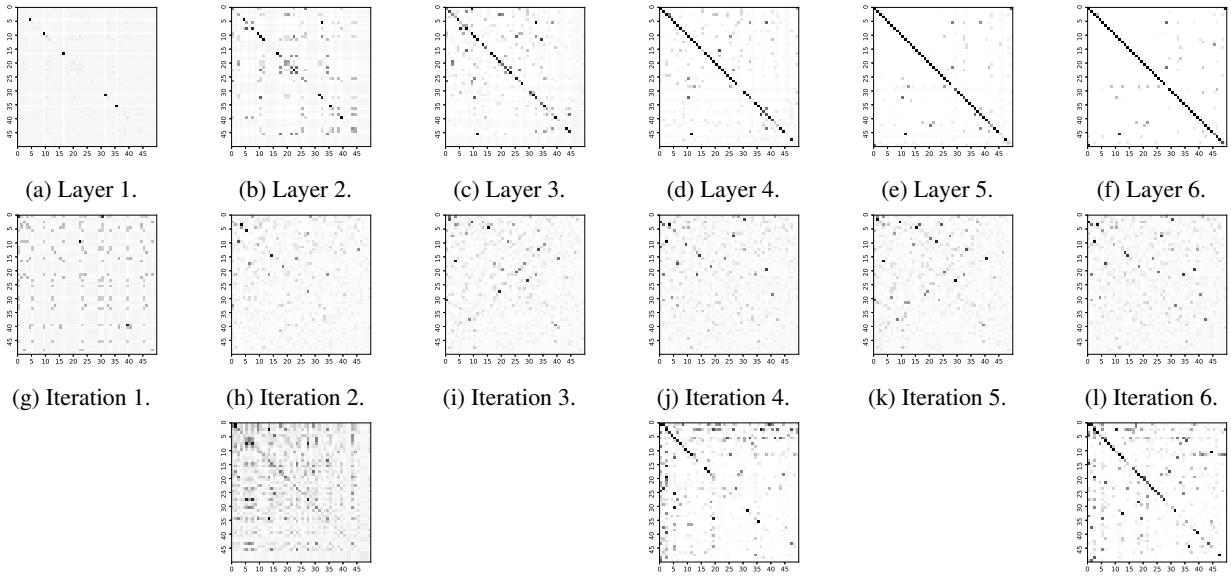
(a) Layer 1. (b) Layer 2. (c) Layer 3. (d) Layer 4. (e) Layer 5. (f) Layer 6.

(g) Iteration 1. (h) Iteration 2. (i) Iteration 3. (j) Iteration 4. (k) Iteration 5. (l) Iteration 6.

(m) Iteration 1. (n) Iteration 2. (o) Iteration 3.

Figure 11: The similarity/witness matrices of the matching process on a fixed pair of dense correlated Erdős-Rényi graphs with $n = 50$, $p = 0.4$, $s = 0.8$ and $\theta = 0.1$. Darker points correspond to higher similarity (in $Y_l$) or a larger number of witnesses. Figure 11(a) — Figure 11(f) are the similarity matrix from each layer of SeedGNN. Figure 11(g) — Figure 11(l) are the witness matrix from each iteration of the iterative 1-hop algorithm. Figure 11(m) — Figure 11(o) are the witness matrix from each iteration of the iterative 2-hop algorithm.

Then, we study the matching process in *sparse* graphs. We fix a pair of correlated Erdős-Rényi graphs with $n = 50$, $p = 0.1$, $s = 0.8$ and $\theta = 0.1$. Then, we also index the nodes from 0 to 49 in the descending order of the node degree in the parent graph $\mathcal{G}_0$. In Figure 12, we show the similarity matrix $Y_l$ in each layer of our SeedGNN, and compare it with the witness matrix in each iteration using either the 1-hop or 2-hop algorithm. In contrast to Figure 11, in this case, we observe that the similarity matrices provided by SeedGNN are more similar to the witness matrices of the iterative 2-hop algorithm than those of the iterative 1-hop algorithm. Specifically, since the graphs are sparse, there are very few 1-hop witnesses even for true pairs. Thus, the 1-hop algorithm almost fails completely (see Figure 12(g) — Figure 12(l)). On the contrary, the 2-hop witnesses from the initial seeds are much more reliable (see Figure 12(m)). As a result, the iterative 2-hop algorithm produces much better results (see Figure 12(m) — Figure 12(o)). By comparing Figure 12(b) with Figure 12(h) and Figure 12(m), we can observe that the second layer of SeedGNN produces a similarity matrix that is closer to the witness matrix of the 2-hop algorithm than that of the 1-hop algorithm. Thus, we can infer that, for these sparse graphs in which the confidence levels of new seeds are low, SeedGNN utilizes 2-hop witness-like information from the initial seeds, and avoids using 1-hop witnesses based on these new seeds.



(a) Layer 1. (b) Layer 2. (c) Layer 3. (d) Layer 4. (e) Layer 5. (f) Layer 6.

(g) Iteration 1. (h) Iteration 2. (i) Iteration 3. (j) Iteration 4. (k) Iteration 5. (l) Iteration 6.

(m) Iteration 1. (n) Iteration 2. (o) Iteration 3.

Figure 12: The similarity/witness matrices of the matching process on a fixed pair of sparse correlated Erdős-Rényi graphs with $n = 50$, $p = 0.1$, $s = 0.8$ and $\theta = 0.1$. Darker points correspond to higher similarity (in $Y_l$) or a larger number of witnesses. Figure 12(a) — Figure 12(f) are the similarity matrix from each layer of SeedGNN. Figure 12(g) — Figure 12(l) are the witness matrix from each iteration of the iterative 1-hop algorithm. Figure 12(m) — Figure 12(o) are the witness matrix from each iteration of the iterative 2-hop algorithm.

In summary, from these two case studies, we conclude that our SeedGNN might be able to choose the appropriate features for different types of graphs according to the confidence level of new seeds. Further, we observe that the matching accuracy of SeedGNN is even higher than that of the 1-hop and 2-hop algorithms, the latter two of which have been theoretically proven to work well for dense graphs and sparse graphs, respectively (Mossel & Xu, 2019). Thus, this result suggests that SeedGNN may extract more valuable features, or learn more effective ways to synthesize witness-like information, than the theoretical algorithms.

## F. Limitations

The limitations of our proposed SeedGNN are three-fold. 1) We only consider using topological structure in our SeedGNN. Although the non-topological features are sometimes hard to obtain or inaccurate due to various constraints in practice, when they are available, we may consider combining them with topological features. 2) We only adopt the insights from theoretical algorithms for seeded graph matching. The seedless graph matching algorithms may yield some additional useful insights on effectively using the topological information. 3) Compared to non-learning methods, SeedGNN requires additional training process and higher computational complexity.

There are many interesting future directions to overcome the limitations, such as simultaneously using topological and non-topological features, and extending our key ideas into seedless graph matching.

## G. Broader Impact

Our work has a positive impact on many practical applications, such as network privacy, computational biology, computer vision, and natural language processing. For example, with the increasing number and scale of social networks, mapping users across online social networks attracts much attention from both academia and industry. Our work can further help information analysis, such as user behavior prediction (Jiang et al., 2016), identity verification and cross-domain recommendation (Lu et al., 2016; Li & Lin, 2014). Another important application of graph matching is protein interaction network alignment (Singh et al., 2008; Kazemi et al., 2016; Kriege et al., 2019). The alignment of protein-protein interaction

(PPI) networks enables us to uncover the relationships between different species, which leads to a deeper understanding of biological systems. In computer vision, our work can be applied in finding similar images (Conte et al., 2004; Schellewald & Schnörr, 2005; Vento & Foggia, 2013) and matching 3D deformable shapes (Kim et al., 2011; Lähner et al., 2016; Vestner et al., 2017a;b). In natural language processing, our work can be used in question answering, machine translation, and information retrieval (Haghighi et al., 2005).

Participants joining different social platforms may have privacy or anonymity considerations. Our work may have some adverse impact on user privacy protection. However, we believe that our proposed design choices may also be useful for guiding the design of privacy protection schemes.