Semi-Supervised Hierarchical Graph Classification

Jia Li, Yongfeng Huang, Heng Chang and Yu Rong

Abstract—Node classification and graph classification are two graph learning problems that predict the class label of a node and the class label of a graph respectively. A node of a graph usually represents a real-world entity, e.g., a user in a social network, or a document in a document citation network. In this work, we consider a more challenging but practically useful setting, in which a node itself is a graph instance. This leads to a hierarchical graph perspective which arises in many domains such as social network, biological network and document collection. We study the node classification problem in the hierarchical graph where a "node" is a graph instance. As labels are usually limited, we design two novel semi-supervised solutions named SEAL-C/AI. SEAL-C/AI adopt an iterative framework that takes turns to update two modules, one working at the graph instance level and the other at the hierarchical graph level. To enforce a consistency among different levels of hierarchical graph, we propose the Hierarchical Graph Mutual Information (HGMI) and further present a way to compute HGMI with theoretical guarantee. We demonstrate the effectiveness of this hierarchical graph modeling and the proposed SEAL-C/AI methods on text and social network data.

Index Terms—hierarchical graph, graph representation, graph mutual information, active learning.

1 Introduction

Graph has been widely used to model real-world entities and the relationship among them. Two graph learning problems have received a lot of attention recently, i.e., node classification and graph classification. Node classification is to predict the class label of nodes in a graph, for which many studies in the literature make use of the connections between nodes to boost the classification performance. For example, [1] enhances the recommendation precision in LinkedIn by taking advantage of the interaction network, and [2] improves the performance of document classification by exploiting the citation network. Graph classification, on the other hand, is to predict the class label of graphs, for which various graph kernels [3]–[6] and deep learning approaches [7][8] have been designed. In this work, we consider a more challenging but practically useful setting, in which a node itself is a graph instance. This leads to *a hierarchical graph in which a set of graph instances are interconnected via edges*. This is a very expressive data representation, as it considers the relationship between graph instances, rather than treating them independently. The hierarchical graph model applies to many real-world data, for example, a social network can be modeled as a hierarchical graph, in which a user group is represented by a graph instance and treated as a node in the hierarchical graph, and then a number of user groups are interconnected via interactions or common members. As another example, a document collection can be modeled as a hierarchical graph, in which a document is regarded as a graph-of-words [9], and then a set of documents are interconnected via the citation relationship. In this paper, we study *hierarchical graph classification, which predicts the class label of graph instances in a hierarchical graph*.

To represent a hierarchical graph, there is a natural and important question:"given multiple levels of inputs and representations of a hierarchical graph, how can we enforce a consistency among different levels of the graph?" In this work, we propose to use *mutual information* (MI) to enforce this consistency. We are motivated by recent developments of graph MI maximization methods [10][11], and generalize the MI computation to hierarchical graphs, which is named Hierarchical Graph Mutual Information (HGMI). Our theoretical derivations show that HGMI can be decomposed into a linear combination of node-level MI [10] and graph-level MI [11]. In this regard, we can use non-hierarchical graph MI computational methods to compute HGMI. More specifically, for graph instances, we compute MI between nodes and instance representations via graph-level MI computational methods (e.g., INFOGRAPH [11]); for connections between graph instances, we compute MI between instances and hierarchical graph representations via node-level MI computational methods (e.g., GMI [10]).

Another challenge is that the amount of available class labels is usually very small in real-world data, which limits the classification performance. To address this challenge, we take a semi-supervised learning approach to solving the graph classification problem. We design an iterative algorithm framework which takes turns to update two modules: Instance Classifier (IC) and Hierarchical Classifier (HC). We start with the limited labeled training set and build IC, which produces the embedding vectors of graph instances. HC takes the embedding vectors as input and produces predictions. We cautiously select a subset of predicted labels by HC with high confidence to enlarge the training set. The enlarged training set is then fed into IC in the next iteration to update its parameters in the hope of generating more accurate embedding vectors and predictions.

- *Jia Li and Yongfeng Huang are co-first authors and with The Hong Kong University of Science and Technology. E-mail: jialee@ust.hk*
- *Heng Chang is with Tsinghua University.*
- *Yu Rong is with Tencent AI Lab.*

Manuscript received April 19, 2005; revised August 26, 2015.

HC further takes the new embedding vectors for model update and class prediction. This is our proposed solution, called SEmi-supervised grAph cLassification via Cautious Iteration (SEAL-CI), to the graph classification problem.

We also extend this iterative algorithm to the active learning framework, in which we iteratively select the most informative instances for manual annotation, and then update the classifiers with the newly labeled instances in a similar process as described above. This method is called SEAL-AI in short.

Our contributions are summarized as follows.

- We study semi-supervised hierarchical graph classification, which is scarcely studied in the literature. Our proposed solutions SEAL-CI/AI achieve superior classification performance to the state-of-the-other graph kernel and deep learning methods, even when given very few labeled training instances.
- We generalize the MI estimation to the hierarchical graph domain and propose a new concept named Hierarchical Graph Mutual Information (HGMI). We show HGMI can be decomposed by a sum of mutual information in non-hierarchical graph domain. Upon this, HGMI maximization can be achieved with the help of node/graph-level MI maximization.
- We present the HIERARCHICAL GRAPH BENCHMARK [1] with both text data and social network data, with the goal of facilitating reproducible hierarchical graph research. From the social networking platform Tencent QQ, we collect 37,836 QQ groups with 18,422,331 unique anonymized users, in which the hierarchical graph is constructed with common memberships (hierarchy-level) and friendships (instance-level). From the arXiv papers, we collect 4,666 papers, in which the hierarchical graph is constructed with citations (hierarchy-level) and semantics (instance-level)

The remainder of this paper is organized as follows. Section 2 gives the problem definition and Section 3 describes the design of SEAL-C/AI. We report the experimental results in Section 4 and discuss related work in Section 5. Finally, Section 6 concludes the paper.

2 Problem Definition

We denote a set of objects as $O = \{o_1, o_2, \cdots, o_N\}$ which represent real-world entities. We use d attributes to describe properties of objects, e.g., age, gender.

We use a graph instance to model the relationship between objects in O, which is denoted as $g = (V, A, X)$, $V \subseteq O$ is the node set and $|V| = n$, A is an $n \times n$ adjacency matrix representing the connectivity in g, and $X \in \mathbb{R}^{n \times d}$ is a matrix recording the attribute values of all nodes in g.

A set of graph instances $G = \{g_1, g_2, \cdots\}$ can be interconnected, and the connectivity between the graph instances is represented by an adjacency matrix A. The graph instances and their connections are modeled as a hierarchical graph $G = (G, A)$.

A graph instance $g \in G$ is a labeled graph if it has a class label, represented by a vector $y \in \{0, 1\}^c$, where c is the number of classes. A graph instance is unlabeled if its class label is unknown. Then G can be divided into two subsets: labeled graphs G_l and unlabeled graphs G_u, where $G = G_l \cup G_u$, $|G_l| = L$ and $|G_u| = U$. In this paper, we study the problem of graph classification, which determines the class label of the unlabeled graph instances in G_u from the available class labels in G_l and the hierarchical graph topological structure. As the amount of available class labels is usually very limited in real-world data, we take a semi-supervised learning approach to solving this problem.

Figure 1 depicts a hierarchical graph in the context of a social network. A, B, C, D denote four user groups. Group A has the class label of "game", B has the label of "non-game", while the class labels of C and D are unknown. These four groups are connected via some kind of relationships, e.g., interactions or common members. The internal structure of each user group shows the connections between individual users. From this hierarchical graph, we want to determine the class labels of groups C and D.

Fig. 1: A hierarchical graph with four graph instances A, B, C, D, each of which corresponds to a user group in a social network.

3 Methodology

3.1 Problem Formulation

In our problem setting, we have two kinds of information: graph instances and connections between the graph instances, which provide us with two perspectives to tackle the graph classification problem. Accordingly, we build two classifiers: a classifier IC constructed for graph instances and a classifier HC constructed for the hierarchical graph.

For both classifiers, one goal is to minimize the supervised loss, which measures the distance between the predicted class probabilities and the true labels. Another goal is to maximize a hierarchical graph mutual information, which measures the distance among the input hierarchical graph, IC representations and HC representations. The purpose of this hierarchical graph mutual information is to enforce a consistency among different levels of hierarchical graph.

Formally, we formulate the graph classification problem as an optimization problem:

1. data and code are available at https://hiergraph.github.io/

Fig. 2: Overview of the proposed hierarchical graph mutual information computation. The orange part shows the mutual information computation between the input G and graph instance representations E; the green part shows the mutual information computation between graph instance representations E and final hierarchical representations Φ.

$$\arg\min \ \ (G_l) \quad (G); \tag{1}$$

where (G_l) is the supervised loss for the labeled graph instances, and (G) is the hierarchical graph mutual information for all graph instances.

Specifically, (G_l) includes two parts:

$$(G_l) = \sum_{g_i \in G_l} (L(y_i; \ _i) + L(y_i; \ _i)); \tag{2}$$

where $_i$ is a vector of predicted class probabilities by IC, and $_i$ is a vector of predicted class probabilities by HC. $L(\ ;\ )$ is the cross-entropy loss function.

The hierarchical graph mutual information (HGMI) $(\ )$ is defined as:

$$(G) = I(G; E; \ ); \tag{3}$$

where $I(\ ;\ ;\ )$ denotes the mutual information between a set of three variables, E denotes the representations derived by IC and $$ denotes the representations derived by HC. In the following subsections, we first analyse HGMI and describe the way to compute HGMI; we then give our design of classifiers IC and HC, and our detailed training algorithms.

3.2 Analysis of HGMI

We first show HGMI coincides with graph mutual information between G and $$ when the hierarchical graph forms a Markov chain.

Theorem 1. *Consider the hierarchical graph forms a Markov chain, i.e., $G \to E \to $, then*

$$I(G; E; \ ) = \ I(G; \ ) \tag{4}$$

here $I(G; \ )$ is the mutual information between the input graph instances and hierarchical graph representations.

The proof is trivial, since the three variables form a Markov chain $G \to E \to $, then $I(G; \ ; E) = 0$, we have

$$I(G; E; \ ) = \ I(G; \ ) \quad I(G; \ ; E) = I(G; \ ) \tag{5}$$

It was worth mentioning that the Markov property is quite reasonable here. For example on text data, upon given document representation, it is reasonable to assume graph-of-words inputs (within documents) are irrelevant when predicting document citations (outside of documents).

We have the following hierarchical graph mutual information decomposition theorem to compute HGMI.

Theorem 2. *Consider the hierarchical graph forms a Markov chain, the hierarchical graph mutual information can be decomposed by a sum of non-hierarchical graph mutual information, namely,*

$$I(G; E; \ ) = \ (I(G; E) + I(E; \ )) \tag{6}$$

here $\in [0; \frac{1}{2}]$, $I(G; E)$ is the mutual information between graph instance input and graph representation on instance level, $I(E; \ )$ is the mutual information between instance representation and hierarchical graph representation on hierarchical level.

To prove Theorem 2, we first introduce two lemmas.

Lemma 1. *For any random variable Z_1, Z_2, Z_3, we have*

$$I(Z_1; Z_2; Z_3) \quad \frac{1}{2}(I(Z_1; Z_2) + I(Z_1; Z_3)) \tag{7}$$

here $I(Z_1; Z_2; Z_3)$ is the mutual information between variable Z_1 and the joint distribution of Z_2 and Z_3.

To prove Lemma 1, we make use of the chain rule for mutual information.

$$\begin{aligned} I(Z_1; Z_2; Z_3) = \ & I(Z_1; Z_3) + I(Z_1; Z_2 | Z_3) \\ & I(Z_1; Z_3) \end{aligned} \tag{8}$$

The last inequality holds as mutual information is non-negative. Accordingly, we have

$$I(Z_1; Z_2; Z_3) \quad I(Z_1; Z_2) \tag{9}$$

Based on Eq. 8 and Eq. 9, we complete the proof of Lemma 1.

Lemma 2. *For a Markov chain $G \to E \to $, we have*

$$I(E; G; \ ) \quad I(E; G) + I(E; \ ) \tag{10}$$

Proof According to [12], if the conditional distribution $P(Z_3 | Z_1; Z_2)$ is multiplicative, i.e., $\exists$ two functions s_1 and s_2, s.t., $P(Z_3 | Z_1; Z_2) = s_1(Z_3; Z_1)s_2(Z_3; Z_2)$, then $I(Z_1; Z_2) \quad I(Z_1; Z_2 | Z_3)$. Since $G \to E \to $ is a Markov chain, we have $P(\ | G; E) = P(\ | E)$, which means $P(\ | G; E)$ is multiplicative. Thus, $I(G; E) \quad I(G; E | \ )$ holds in our case.

$$\begin{aligned} I(E; G; \ ) = \ & I(E; \ ) + I(E; G | \ ) \\ & I(E; G) + I(E; \ ) \end{aligned} \tag{11}$$

Thus, we complete the proof of Lemma 2. We then prove Theorem 2,

$$\begin{aligned} I(G; E; \ ) = \ & I(G; E) \quad I(G; E | \ ) \\ = \ & I(G; E) \quad (I(E; G; \ ) \quad I(E; \ )) \\ = \ & I(G; E) + I(E; \ ) \quad I(E; G; \ ) \\ = \ & \ (I(G; E) + I(E; \ )) \end{aligned} \tag{12}$$

The last equality holds as we have $\frac{1}{2}(I(G; E) + I(E; \ ))$ $I(E; G; \ ) \quad I(G; E) + I(E; \ )$, based on Lemma 1 and Lemma 2. Thus, Theorem 2 is proved.

Fig. 3: Schematic diagram of the learning framework SEAL-CI. There are two subroutines: graph instance representation (in the orange box) and hierarchical graph representation (in the green box).

We use Figure 2 to illustrate the idea to compute HGMI. One naive method to compute HGMI is according to the de nition, which inevitably involves with the computation of joint distribution $P(G; E;)$ and is not tractable in general [13]. With Theorem 2, we decouple the computation of HGMI into a sum of the computation of graph mutual information without hierarchies, which is tractable as used in many previous works [14][10]. Next we present the design of IC and HC classi er, and then give the detailed way to maximize HGMI in Section 3.4.

3.3 Design of Classi ers

Classi er IC takes a graph instance as input. As different graph instances have different numbers of nodes, IC is expected to handle graph instances of arbitrary size. Classi er HC takes the hierarchical graph as input, in which individual graph instances are the "nodes". We propose to embed a graph instance $g_i \in G$ into a xed-length vector e_i via IC rst. Then HC can take as input the embedding vectors of graph instances and the adjacency matrix A. In particular, IC takes as input the adjacency matrix A_i and attribute matrix X_i of an arbitrary-sized graph instance g_i, and outputs an embedding vector e_i and a vector of predicted class probabilities $_i$, i.e., $(e_i; _i) = IC(A_i; X_i)$. HC takes the embedding vectors $E = \{e_i\}_{i=1}^{L+U}$ and A, and outputs the predicted class probabilities $= \{ _i\}_{i=1}^{L+U}$, i.e., $= HC(E; A)$.

3.3.1 Graph instance representation

The task of graph instance representation is to produce a xed-length embedding vector of a graph instance, for which, however, we identify two challenges:

> Size invariance How to design the network structure to exibly take an arbitrary-sized graph instance and produce a xed-length embedding vector?
> Permutation invariance How to derive the representation regardless of the permutation of nodes?

Recently graph pooling has emerged as a research topic that can be used to tackle the above challenges. Some common practices include DIFFPOOL [15], Attention-based Pool [16][17][18], MinCut-Pool [19]. In particular, [18] shows Attention-based Pool is permutation-invariant by connecting with Weisfeiler-Lehman test. To this end, in IC, we rst utilize a multi-layer Graph Neural Network (GNN) [20] to smooth each node's features over the graph's topology. Then we adopt a graph pooling method and transform a variable number of smoothed nodes into a xed-length embedding vector. Finally, the embedding vector is cascaded with a fully connected layer and a softmax function, in which the label information can be leveraged to discriminatively transform the embedding vector e into $$.

Formally, we are given the adjacency matrix $A \in R^{n \times n}$ and the attribute matrix $X \in R^{n \times d}$ of a graph instance g as inputs. We apply a multi-layer GNN network:

$$H = GNN(A; X); \tag{13}$$

here we get a set of representation $H \in R^{n \times v}$ for nodes in g. Note that the representation H is size variant, i.e., its size is still determined by the number of nodes n. So next we utilize the graph pooling mechanism:

$$e = Pooling(H): \tag{14}$$

here consider Attention-based Pool is adopted, as the attention weight is used to multiply with H and atten the representation, $e \in R^{m}$ is size invariant and does not depend on the number of nodes n any more.

To summarize, we use GNN and graph pooling to construct the instance-level classi er IC. It produces not only the estimated class probability $= \{ _i\}_{i=1}^{L+U}$, but also instance representations $E = \{e_i\}_{i=1}^{L+U}$, which is the input for classi er HC in the next.

3.3.2 Hierarchical graph representation

Given E and the adjacency matrix $A \in R^{(L+U) \times (L+U)}$, our next task is to infer the parameters of classi er HC and derive the predicted probabilities $= \{ _i\}_{i=1}^{L+U}$. This problem falls into the setting of non-hierarchical graph setting where E can be treated as the set of node features. In this context, we consider again a multi-layer GNN. Thus the model becomes:

$$= HC(E; A) = softmax(GNN(E; A)); \tag{15}$$

where $\in R^{(L+U) \times c}$ is the derived hierarchical graph representation. With and E, next we introduce the way to maximize HGMI.

3.4 Maximization of HGMI

With Theorem 2, the computation of HGMI is reduced into MI between graph input and graph-level representation ($I(G;E)$) and MI between graph input and node-level representation ($I(E;\ )$), in which we have many choices such as INFOGRAPH [11] and GMI [10].

To compute $I(G;E)$, we use the method of INFOGRAPH [11], where MI is computed between the graph input G and graph-level representation E. More specifically, it is shown in [11] that maximizing the global $I(G;E)$ can be estimated by MI between node representations and graph-level representations,

$$I(G;E) = \sum_{i}^{L+U} \frac{1}{L+U} \sum_{j}^{n} \frac{1}{n} I(h_{ij}\ ;e_i); \tag{16}$$

where h_{ij} is a node representation sampled from H_i. The core thus becomes how we compute $I(h_{ij}\ ;e_i)$. In this work, we resort to Jensen-Shannnon MI estimator (JSD),

$$I(h_{ij}\ ;e_i) = \quad sp(\ \ D_I(h_{ij}\ ;e_i) + E_{\hat{h}_{ij}}\ sp(D_I(\hat{h}_{ij}\ ;e_i)); \tag{17}$$

where $D_I : R^v \quad R^m \ ! \quad R$ is a discriminator constructed by a neural network with parameters D_I. $\hat{h}_{ij}$ is a negative example. $sp(x) = \log(1+\exp(\ x))$ is the soft-plus function. For a detailed graph-level negative sampling process, please see INFOGRAPH [11].

To compute $I(E;\ )$, we adopt the method of GMI [10], where MI is computed between the graph feature input and node-level representation. More specifically, it is shown in [10] that, for a node-level representation $_i$, maximizing the global $I(E;\ _i)$ can be decomposed as a weighted sum of local MIs,

$$I(E;\ _i) = \sum_{j}^{L+U} w_{ij}\ I(e_j;\ _i); \tag{18}$$

here we adopt the mean version of GMI, meaning $w_{ij} = \frac{1}{L+U}$. To compute $I(e_j;\ _i)$, we use JSD again,

$$I(e_j;\ _i) = \quad sp(\ \ D_H(e_j;\ _i) + E_{\hat{e}_j}\ sp(D_H(\hat{e}_j;\ _i)); \tag{19}$$

where $D_H : R^m \quad R^c \ ! \quad R$ is a discriminator constructed by a neural network with parameters D_H. $\hat{e}_j$ is a negative example. For a detailed node-level negative sampling process, please see GMI [10].

3.5 The Proposed SEAL-CI Model

In this subsection, we present our method to minimize the objective function Eq. 1. A naive way would be directly minimizing Eq. 1 in an end-to-end fashion with the limited labels. However, in real-world scenarios, the number of labeled graph instances L can be quite small compared to the number of unlabeled instances U. In this context, neural network based classifiers may suffer from the problem of over fitting.

Following previous works [21][2], we use the idea of iterative algorithm to alternate optimizing the two modules of IC and HC by trusting a subset of predictions of HC. To be more specific, we combine the instance representation in Section 3.3.1 and hierarchical representation in Section 3.3.2 into one iterative algorithm. We build IC to produce instance representation E^t for all graph instances in iteration t, and then feed E^t into HC to get the predicted probabilities ^{t}. We then make use of ^{t} to update the parameters of IC and generate E^{t+1}, which is then used as the input of HC in iteration $t+1$. Figure 3 depicts the overall framework of this iterative process.

Algorithm 1: SEAL-CI

Input: $G = \{G; A\}$.
Output: $^{t+1}$.

1 Initial: $G_{tmp} = \{\ \}, G_l^0 = G_l, t = 0$;
2 while t U do
3 $\quad W^{t+1} \quad \arg\min\ (G_l^t | W^t) \quad (G^t | W^t)$;
4 $\quad ^{t+1}; E^{t+1} \quad IC(A; X | W^{t+1})$;
5 $\quad ^{t+1} \quad HC(E^{t+1}; A | W^{t+1})$;
6 $\quad G_{tmp} \quad T(t; \ ^{t+1}_{G_u})$;
7 $\quad G_l^{t+1} \quad G_l \ [\ G_{tmp}$;
8 $\quad G_{tmp} = \{\ \}$;
9 Return $^{t+1}$;

3.5.1 How to utilize ^{t}?

To update the instance representations, a naive approach is feeding the whole set of ^{t} for the parameter update in IC, which is the idea of the original ICA [2]. However, not all ^{t} are correct in their predictions. The false predictions may lead the update of embedding neural network to the wrong direction. Within the set of unlabeled graphs, different could contribute differently to the update of embedding neural network. To this end, we make use of the idea of [21], a variant of the original ICA, and cautiously exploit a subset of ^{t} to update the parameters of IC in each iteration. Specifically, in iteration t, we choose the t most confident predicted labels while ignoring the less confident predicted labels. This operation continues until all the unlabeled samples have been utilized. This algorithm is called SEmi-supervised grAph cLassification via Cautious Iteration (SEAL-CI) and is presented in Algorithm 1. Note here W is the set of all the parameters of IC and HC. In line 6, the training set for IC has been enlarged by t instances and it is done by "committing" these instances' labels from their maximum probability. In other words, the newly enrolled training instances are found by:

$$T(\ ;\ ) = \quad top(\max_{2}\ ;\ ): \tag{20}$$

Here function $top(\ ;\ )$ is used to select the top instances and function max is used to select the maximum value in the probability vector .

3.6 The Proposed SEAL-AI Model

Our proposed model is easy to extend to the active learning scenario. In case further annotation is available, we can perform active learning and ask for annotations with a budget of B. Denote the set of graph instances being annotated as G_B, then the objective function in the active learning setting is re-written as:

$$\begin{aligned} &\min f(G_B | B; W) \\ &\text{s.t.} \quad |G_B| \quad B; \end{aligned} \tag{21}$$

Algorithm 2: SEAL-AI

Input: $G = \{G, A\}$.
Output: Θ^{t+1}.
1 Initial: $G_{tmp} = \{\}, G_B^0 = \{\}, G_l^0 = G_l, G_u^0 = G_u, t = 0$;
2 while $|G_B^t| \leq B$ do
3 $W^{t+1} \leftarrow \arg\min \Omega(G_l^t|W^t) - \Phi(G_l^t|W^t)$;
4 $\Theta^{t+1}, E^{t+1} \leftarrow \mathrm{IC}(A, X|W^{t+1})$;
5 $\Phi^{t+1} \leftarrow \mathrm{HC}(E^{t+1}, A|W^{t+1})$;
6 $G_{tmp} \leftarrow \arg\max_{|G_{tmp}|=k} \Phi(G_u^t \cap G_{tmp}|W^{t+1})$;
7 $G_B^{t+1} \leftarrow G_B^t \cup G_{tmp}$;
8 $G_l^{t+1} \leftarrow G_l^t \cup G_{tmp}$;
9 $G_u^{t+1} \leftarrow G_u^t \cap G_{tmp}$;
10 $G_{tmp} = \{\}$;
11 Return Θ^{t+1};

where $f(G|B, W) = \Phi(G_l \cup G_B|W) - \Phi(G_u \cap G_B|W)$. This is a mixed combinatorial and continuous optimization problem. It is very hard to infer the model parameters and the active learning set G_B simultaneously. By definition, the active learning set G_B is intractable unless the model parameters are completely inferred. To solve this chicken-and-egg problem, we decompose the objective function into two sub-steps: parameter optimization and candidate generation. Then we optimize $f(G|B, W)$ iteratively. This algorithm is called SEmi-supervised grAph cLassification via Active Iteration (SEAL-AI) and is shown in Algorithm 2.

At the beginning of this iterative process, we optimize the supervised loss $\Omega(G_l|W)$ and HGMI based on current labeled graphs in G_l (line 3 in Algorithm 2). In active learning, the choice of candidate generator is a key component. We exploit the idea of [22][23] and choose the candidate graph instances G_{tmp} by maximizing the current HGMI based on the new parameter obtained in the first step (line 6 in Algorithm 2). At last we label G_{tmp} and update G_B, G_l and G_u respectively (line 7-9 in Algorithm 2).

Formally, we follow the "most uncertain" candidate selection criteria in active learning to reduce the model uncertainty[23]–[25], and select the instances whose annotations will result in the maximum mutual information among the unlabeled graph instances:

$$\arg\max \Phi(G_u \cap G_B|W) \tag{22}$$

Here HGMI is adopted and can be decomposed into linear combinations of individual MIs ($I(h_{ij}; e_i)$ and $I(e_j; \theta)$), by Theorem 2, Eq. 16 and Eq. 18. Thus, we can choose the candidates by:

$$z_i = \sum_j^n \frac{1}{n} I(h_{ij}; e_i) + \sum_j^{L+U} \frac{1}{L+U} I(e_j; \theta). \tag{23}$$

Then we choose k instances with the smallest values. For the consideration of efficiency, $I(e_j; \theta)$ can be computed within one hop of instance g_i.

3.7 Complexity Analysis

We analyze the computational complexity of our proposed methods. Here we only focus on Algorithm 1, since Algorithm 2 is almost the same except the step of selecting candidate graph instances to the training set. In Algorithm 1, the intensive parts in each iteration contain the updates of IC and HC as well as the selection of candidate instances. We discuss each part in details below.

Regarding IC, the core is to compute the activation matrix H in Eq. (13) where the matrix-vector multiplications are up to $O(E_1 d)$ ops for one input graph instance; here E_1 denotes the number of edges in the graph instance and d is the input feature dimension. Thus, it leads to the complexity of $O(E_1(L+U)d)$ by going through all $L+U$ graph instances.

Next, the computation by HC in Eq. (15) requires $O(E_2 m)$ ops in total, where E_2 denotes the number of links between graph instances. Then in candidate selection, performing comparisons between all unlabeled graph instances has a complexity of $O(L+U)$ given the outputs of two classifiers IC and HC.

Overall, the complexity of our method is $O(E_1(L+U)d + E_2 m)$ which scales linearly in terms of the number of edges in each graph instance (i.e., E_1), the number of links between graph instances (i.e., E_2) and the number of graph instances (i.e., $(L+U)$).

4 EXPERIMENTS

We evaluate SEAL-C/AI methods on synthetic, text and social network data sets.

4.1 Synthetic Data

We evaluate the performance of SEAL-C/AI on synthetic data. We first give a description of the synthetic generator, then visualize the learned embeddings. Finally we compare our methods with baselines in terms of classification accuracy.

4.1.1 Synthetic Data Generation

The benchmark data set Cora [26] contains 2708 papers which are connected by the citation relationship. We borrow the topological structure of Cora to provide the skeleton (i.e., edges) of our synthetic hierarchical graph. Then we generate a set of graph instances, which serve as the nodes of this hierarchical graph. Since there are 7 classes in Cora, we adopt 7 different graph generation algorithms, that is, Watts-Strogatz [27], Tree graph, Erdos-Rényi [28], Barbell [29], Bipartite graph, Barabasi-Albert graph [30] and Path graph, to generate 7 different types of graph instances, and connect them in the hierarchical graph.

Specifically, to generate a graph instance g, we randomly sample a number from $[100, 200]$ as its size n. Then we generate its structure and assign the class label according to the graph generation algorithm. In this step, the parameter p in Watts-Strogatz, Erdos-Rényi, Bipartite graph and Barabasi-Albert graph is randomly sampled from $[0.1, 0.5]$, the branching factor for Tree graph is randomly sampled from $[1, 3]$. At last, to make this problem more challenging, we randomly remove 1% to 20% edges in the generated graph g. The statistics of the generated graph instances are listed in Table 1.

TABLE 1: Statistics of generated graph instances

Type	Number	Nodes	Edges	Density
Watts-Strogatz	351	173	347	2.3%
Tree	217	127	120	1.5%
Erdos-Rényi	418	174	3045	20%
Barbell	818	169	2379	16.3%
Bipartite	426	144	1102	10.6%
Barabasi-Albert	298	173	509	3.4%
Path	180	175	170	1.1%

Fig. 4: Two-dimensional visualization of graph embeddings generated from the synthesized graph instances using SAGE. The nodes are colored according to their graph types.

4.1.2 Visualization

To have a better understanding of the synthesized graph instances, we split all 2708 graph instances into two parts. 1708 instances are used for training and the remaining 1000 instances are used for testing. We apply SAGE [16] on the training set and derive the embeddings of the 1000 testing instances. We then project these learned embeddings into a two-dimensional space by t-SNE [31], as depicted in Figure 4. Each color in Figure 4 represents a graph type. As we can see from this two-dimensional space, the geometric distance between the graph instances can reflect their graph similarity properly.

4.1.3 Baselines and Metrics

We use 6 approaches as our baselines:

GK-SVM [5], which calculates the graphlet count kernel (GK) matrix, then GK-SVM feeds the kernel matrix into SVM [32].
WL-SVM [6], which is similar as above but using the Weisfeiler-Lehman subtree kernel (WL).
graph2vec-GCN [8], which embeds the graph instances by graph2vec and then feeds the embeddings to GCN.
SAGE [16], which ignores the connections between graph instances and treats them independently.
MIRACLE [33], which uses the multi-view contrast learning method to exploit relation between the structure of graph instance level and the one at the hierarchical graph level.

TABLE 2: Comparison of different methods on the synthetic data set for semi-supervised graph classification

	Algorithm	Accuracy
*1	GK-SVM	77.8%
	WL-SVM	83.4%
	SAGE	85.7%
*2	graph2vec-GCN	85.2%
	MIRACLE	86.7%
	SEAL	87.8%
*3	SEAL-CI	91.2%
	SEAL-AI	92.4%

SEAL is the base of SEAL-A/CI, which differs with SEAL-A/CI in two ways: 1) it does not consider label enlargement, and 2) it is trained in an end-to-end fashion with the given class labels.

We use 300 graph instances as the training set for all methods except SEAL-AI, for which only 140 graphs are used as labeled graph instances at hand and then $B = 160$ is set for active learning. We use 1000 graph instances as the testing set. We run each method 5 times and report its average accuracy. The number of epochs for graph2vec is 1000 and the learning rate is 0.3. For SEAL-A/CI, we use SAGE [16] as the graph pooling method. We use a two-layer GCN in IC, in which the first GCN layer has 32 output channels and the second GCN layer has 4 output channels. The dense layer has 48 units with a dropout rate of 0.3.

4.1.4 Results

Table 2 shows the experimental results for semi-supervised graph classification. Among all approaches, SEAL-C/AI achieve the best performance. In the following, we analyze the performance of all methods categorized into 3 groups.
Group *1: All the embedding-based methods perform better than these two kernel methods, which proves that embedding vectors are effective representations for graph instances and are suitable input for graph neural networks.
Group *2: graph2vec-GCN achieves 85.2% accuracy, which is comparable to that of SAGE, but lower than that of SEAL-C/AI. One possible explanation is that graph2vec is an unsupervised embedding method, which fails to generate discriminative embeddings for classification. Another possibility is that the 300 training instances do not include very informative ones. These limitations of graph2vec motivate us to use supervised graph representation modules such as SAGE and the label enlargement framework in SEAL-C/AI. MIRACLE and SEAL generate the graph instance representations in a supervised way, and they outperform graph2vec-GCN by more than 1.5%.
Group *3: Both SEAL-CI and SEAL-AI outperform MIRACLE and SEAL, which proves the effectiveness of our hierarchical graph perspective and the label enlargement algorithm for semi-supervised graph classification. SEAL-AI outperforms SEAL-CI slightly by 1.2%. This shows, although SEAL-CI can make use of more training samples, it is still influenced by the false prediction cases of IC.

4.1.5 Influence of the number of labeled training instances

We examine how the number of labeled training instances affects the performance of our methods. We train SAGE and

Fig. 5: Accuracy with different number of labeled training instances on synthetic data for semi-supervised graph classi cation.

SEAL-CI with a label size of $f140; 180; 220; 260; 300g$. We train SEAL-AI with 140 labeled instances and then set the budget B for active learning at $f0; 40; 80; 120; 160g$. Thus the three methods have the same number of labeled training instances. We set $= 40$ in SEAL-CI and $k = 10$ in SEAL-AI. We run all methods 5 times, and plot their average accuracy in Figure 5. As we can see from Figure 5, when the number of labeled training instances is 140, SEAL-CI performs the best since it can utilize more training samples. As the number of labeled training instances increases, the performance of SEAL-AI improves dramatically. SEAL-AI catches up with SEAL-CI at 260 labeled training instances and outperforms SEAL-CI at 300 labeled training instances. It validates that SEAL-AI can make use of the iterations to nd informative and accurate training samples. Meanwhile SEAL-CI trusts the prediction of IC conditionally on its con dence, which may bring some noise to the learning process. SEAL-C/AI outperform SAGE in all cases, which makes sense because SEAL-C/AI make good use of the hierarchical graph setting and consider the connections between the graph instances for classi cation.

4.2 Text Data

We evaluate SEAL-C/AI on the arXiv paper dataset. Firstly, We introduce the compositions of arXiv paper dataset and present how to construct hierarchical graph from text data. Then, we show the evaluation results and give some insights on how to construct a hierarchical graph for text data.

4.2.1 Data Description

arXiv is an open-access repository of electronic preprints. We collect 4666 Computer Science (CS) arXiv papers indexed by Microsoft Academic Graph (MAG) [34]. These papers belong to ve subject areas including AI, CL, IT, LG and CV. The arXiv papers data forms a citation network which indicates citation relationships. Each paper consists of two parts: title and abstract. The statistics of the arXiv paper dataset is listed in Table 3.

The connections among graph instances (i.e., papers) are provided by citation relations. Each paper is a semantic graph instance constructed from its title and abstract. There are three type of nodes in a semantic graph instance: abstract node, title node and argument nodes. Speci cally, for each sentence in abstract, we parse it into a tuple with Semantic Role Labeling (SRL) toolkit developed by AllenNLP [2]. For each tuple, we regard its elements as the argument nodes of the semantic graph. The node features of the semantic graph are extracted by pretrained Bert model [35]. The connections within the semantic graph are created as the following: 1) same tuple, if two nodes are from the same tuple, we connect them, and 2) vocabulary overlapping, we connect the node pair if the size of overlapped words is larger than half of the minimum size of any text node. We show an example of hierarchical graph of arXiv papers data in the Figure 6.

TABLE 3: Statistics of arXiv paper data

Class	Number	Length of title	Length of abstract
AI	232	8.57	158.04
CL	648	9.25	140.36
IT	909	10.12	170.29
LG	1157	8.69	160.63
CV	1720	9.41	170.54

TABLE 4: Comparison of different methods on arXiv Paper data for semi-supervised graph classi cation

	Algorithm	Accuracy
*1	GK-SVM	38.7%
	WL-SVM	35.1%
	graph2vec-GCN	42.7%
*2	Bert-MLP	63.7%
	Bert-IC	70.1%
	Bert-HC	75.2%
*3	MIRACLE	78.4%
	SEAL	78.8%
*4	SEAL-CI	79.4%
	SEAL-AI	80.7%

4.2.2 Baselines and Metrics

In addition to the set of baselines in Section 4.1.3, we compare with the following Bert-feature baselines:

- Bert-MLP, which classi es arXiv papers by feeding average features of title and abstract into a multi-layer perceptron (MLP).
- Bert-IC, which feeds semantic graph instances into IC and classi es arXiv papers in an independent graph classi cation fashion.
- Bert-HC, which feeds average features of title and abstract into HC and classi es arXiv papers in a transductive node classi cation fashion.

400 graph instances are used as labeled training instances for all methods except SEAL-AI, for which only 300 are used as labeled training instances at hand and then B is set to 100 for active learning. We use 2000 instances for testing for all methods. We set the dimension of Bert node feature to 768. We use average pool in IC. We set $= 1$ and updated times $t = 100$ for SEAL-CI, and $k = 1$ for SEAL-AI. We run each method 3 times and report its average accuracy.

2. https://demo.allennlp.org/semantic-role-labeling

Fig. 6: An illustration of the hierarchical graph of arXiv papers data

4.2.3 Results

Table 4 shows the experimental results for semi-supervised graph classi cation on arXiv paper data. SEAL-C/AI obtain the state-of-the-art performances among all baselines. To analyze the results of these baselines, all methods are divided into four group as the following.

Group *1: Both GK-SVM and WL-SVM only utilize the structural information of semantic graph instances. And their classi cation accuracies are lower than 40%, which indicates that structural information of semantic graph instances is not enough to obtain satis ed performance on text data. Graph2vec-GCN tries to exploit both structure of semantic graph instances and the hierarchical graph but still has a weak performance with an accuracy of 42.7%.

Group *2: In this group, we show the performance of several baselines based on Bert. Bert-MLP only uses Bert features extracted from title and abstract, and obtains an accuracy of 63.7%. Bert-IC outperforms Bert-MLP by about 6.4%, by combing both structural and Bert features of text data. Bert-HC, by utilizing the connections between Bert features of title and abstract, achieves the best performance among baselines within Group *2.

Group *3: Both SEAL and MIRACLE aim to minimize mutual information between graph instance representations and hierarchical representations and differ in the measurement of consistency. SEAL and MIRACLE outperform baselines in Group *1 with a margin of 30% and Group *2 with a margin of more than 3%.

Group *4: SEAL-C/AI outperform MIRACLE and SEAL. Moreover, SEAL-AI outperforms SEAL-CI by about 1%, which demonstrates the effectiveness of the active learning algorithm 2.

4.2.4 Discussion

How to construct hierarchical graphs from text data is an open question. In the above experiment, we use semantic parsing techniques to construct graph instances for papers, and citation relationships to connect these papers. We think hierarchical graph structures could be a good paradigm to boost the performance of Natural Language Processing classi cation tasks. Moreover, it is worth mentioning that there are several other ways to construct semantic graphs such as Name Entity Recognition (NER).

TABLE 5: Statistics of collected Tencent QQ groups

Class label	Number	Nodes	Edges	Density
game	1,773	147	395	5.48%
non-game	36,063	365	1586	3.28%

4.3 Social Network Data

In this section, we evaluate SEAL-C/AI on Tencent QQ group data. We describe the characteristics of this data set and then present the experimental results. Finally, we have some open discussions on how to construct a hierarchical graph for social network data.

4.3.1 Data Description

Tencent QQ is a social networking platform in China with nearly 800 million monthly active users [3]. There are around 100 million active online QQ groups. In this experiment, we select 37,836 QQ groups with 18,422,331 unique anonymized users. For each user, we extract seven personal features:

number of days ever since the registration day;
most frequently active area code in the past 90 days;
number of friends;
number of active days in the past 30 days;
number of logging in the past 30 days;
number of messages sent in the past 30 days;
number of messages sent within QQ groups in the past 30 days.

We have 298,837,578 friend relationships among these users. 1,773 groups are labeled as “game” and the remaining groups are labeled as “non-game”.

We construct the hierarchical graph from this Tencent QQ group data as follows. A user is treated as an object, and a QQ group as a graph instance. The users in one group are connected by their friendship. The attribute matrix X is lled with the attribute values of the users. The statistics of the graph instances are listed in Table 5. We build the hierarchical graph from the graph instances via common members across groups, that is, if two groups have more than one common member, we connect them.

4.3.2 Baselines and Metrics

We use the same set of baselines as in Section 4.1.3. 1000 graph instances are used as labeled training instances for all

3. https://www.tencent.com/en-us/articles/17000391523362601.pdf

Fig. 7: The false prediction rate of IC with in SEAL-CI.

TABLE 6: Comparison of different methods on Tencent QQ group data for semi-supervised graph classi cation

	Algorithm	Macro-F1
*1	GK-SVM	48.8%
	WL-SVM	47.8%
	SAGE	54.7%
*2	graph2vec-GCN	48.1%
	MIRACLE	80.3%
	SEAL	84.3%
*3	SEAL-CI	85.5%
	SEAL-AI	87.1%

methods except SEAL-AI, for which only 500 are used as labeled training instances at hand and then B is set to 500 for active learning. We use 10,000 instances for testing for all methods. We run each method 3 times and report its average performance. For IC, we use SAGE and set the parameters as the same in Section 4.1. Since the class distribution is extremely imbalanced in this data set, we report the Macro-F1 instead of accuracy.

4.3.3 Results

Table 6 shows the experimental results. SEAL-C/AI outperform GK, WL and graph2vec by at least 36% in Macro-F1. SEAL beats MIRACLE by about 4%, and SEAL-A/CI outperforms SEAL and MIRACLE with a margin of more than 1.2%. SEAL-AI outperforms SEAL-CI by 1.6%. Next we provide the reason why SEAL-AI outperforms SEAL-CI on this data set. Figure 7 shows the false prediction rate (i.e., the percentage of misclassi ed instances) within the most con dent predictions of IC. As we can see, the false prediction rate increases as increases and it reaches 2:4% when = 2000. In the framework of SEAL-CI, as the iteration goes on, we shall bring in more noise to the parameter update of SAGE, while all the training samples in SEAL-AI are informative and correct. This explains why SEAL-AI outperforms SEAL-CI on this Tencent QQ group data.

4.3.4 Visualization

We provide visualization of a “game” group and its neighborhood in Figure 8. The left part is the ego network of the center “game” group. In the one-hop neighborhood of this “game” group, there are 10 “game” groups and 19 “non-game” groups. “Game” groups are densely interconnected with a density of 34.5%, whereas “non-game” groups are sparsely connected with a density of 8.8%. The much higher density among “game” groups validates that common membership is an effective way to relate them in a hierarchical graph for classi cation. The right part depicts the internal structure of the ego “game” group with 22 users. A bigger node indicates a larger importance obtained by SAGE, and a darker green color implies a larger node degree. These 22 members are loosely connected and there are no triangles. This makes sense because in reality online “game” groups are not acquaintance networks. Regarding the learned node importance, node 1 has the highest importance as it is the second active member and has a large degree in this group. Node 16 is also important since it has the highest degree in this group. The “border” member 5 has a big attention weight since it has the largest number of days ever since the registration day and is quite active in this group.

Fig. 8: The ego network of a “game” group. The left side is the ego network, in which “game” groups are in red and “non-game” groups are in blue. The right side is the internal structure of the ego “game” group, in which a bigger node indicates a larger importance, and a darker color implies a larger node degree.

4.3.5 Discussion

How to construct a hierarchical graph from social network is an open question. In the above experiment, we connect two QQ groups if they have more than one common member (i.e., > 1). When we change the threshold, it directly affects the edge density in the hierarchical graph, and may in uence the classi cation performance. For example, if we connect two QQ groups when they have one common member or more (i.e., 1), the edge density is 2.8% compared with 0.27% in the rst setting. A proper setting of this threshold is data dependent, and can be determined through a validation set.

5 Related Work

This work is related to semi-supervised classi cation of networked data, graph representation, graph mutual information maximization and active learning.

Most work on semi-supervised learning for networked data aims to utilize the network structure to boost the learning performance. The assumption is that network context can provide additional information that is not covered by

node attributes. Ever since the pioneer work of Sen et al. [2], Iterative Classification Algorithm (ICA) has become a paradigm for networked data with limited annotations. In ICA, for each node a local classifier takes the estimated labels of its neighborhood and its own features as input, and outputs a new estimated label. The iteration continues until adjacent estimations stabilize. In ALFNET [22], the authors first cluster the network nodes into several groups, and design a content-only classifier CO and a collective classifier CC. Based on the disagreement score of CO and CC in each iteration, a candidate instance set is generated from different clusters and labeled. Then both CO and CC are re-trained using the labeled set until convergence. One main difference between ICA and ALFNET is that ICA does not require human intervention while ALFNET needs human annotation in case labels of the candidate set are not available. Recent work has focused on using deep learning neural networks to further improve the performance. [36] leverages both network context and node features by jointly training node embedding to predict the class label and the context of the network. Later Kipf and Welling [20] simplify the loss design by only considering the supervised loss while network context is exploited by the GCN operator. Our problem is motivated by these previous works but different from all of the above, as the studied graph is hierarchical.

Representation learning on graphs has been proposed to transform instances in topological space into fixed-size vectors in Euclidean space in which geometric distance reflects their structural similarity. Some pioneer works include node-level representation [37][38]. In terms of graph-level representation, [39] uses CBOW and skip-gram model [40], previously proven to be successful in natural language processing, to learn a new graph kernel. Meanwhile, some other methods focus on generating graph embeddings by integrating node embeddings. [7] proposes a spatial-based graph CNN operator and then concatenates these obtained node representations by imposing a problem-specific node ordering. [41] defines a "graph coarsening" operation by first clustering the node representations and then applying a max-pooling operation.

There has been a surging interest in using MI to derive unsupervised graph representations recently. For node-level representations, DGI [14] maximizes MI between a graph summery representation and node-level representations and shows that maximizing this kind of MI is equivalent to maximizing the one between the node features and node-level representations. Later, GMI [10] directly approaches MI computation by comparing the node input and node-level representations, without the overheads of an extra graph summery representation. For graph-level representations, INFOGRAPH [11] extends the idea of DGI to the field of graph-level representation and contrasts graph-level and subgraph-level representations. Our work advances this research area by proposing hierarchical graph MI and develops an tractable computation method with guarantees.

Active learning has been integrated in many collective classification methods [42][22] to find the most informative samples to be labeled. However, research that generalizes active learning with deep semi-supervised learning is still sparse. The closest work is [43] in which the authors utilize active learning to incrementally fine-tune a CNN network for image classification. Our solution SEAL-AI is different in the sense that the informative samples selected by active learning are used to update the parameters of the graph embedding network, whose output is then fed into HC in an iterative framework.

6 CONCLUSION

In this paper, we study the problem of semi-supervised hierarchical graph classification. To enforce a consistency among different levels of the hierarchical graph, we propose Hierarchical Graph Mutual Information (HGMI) and show HGMI can be computed via non-hierarchical graph mutual information computation methods. We build two classifiers IC and HC at the graph instance level and the hierarchical graph level respectively to fully exploit the available information. Our semi-supervised solutions SEAL-C/AI adopt an iterative framework to update IC and HC alternately with an enlarged training set. We present two hierarchical graph benchmarks and demonstrate that SEAL-C/AI outperform other competitors by a significant margin in accuracy/Macro-F1.

REFERENCES

[1] R. Ramanath, H. Inan, G. Polatkan, B. Hu, Q. Guo, C. Ozcaglar, X. Wu, K. Kenthapadi, and S. C. Geyik, "Towards deep and representation learning for talent search at linkedin," in CIKM, 2018, pp. 2253–2261.

[2] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," AI magazine, vol. 29, no. 3, pp. 93–106, 2008.

[3] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in ICDM, 2005, pp. 74–81.

[4] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in Learning theory and kernel machines, Springer, 2003, pp. 129–143.

[5] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in AISTATS, 2009, pp. 488–495.

[6] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," Journal of Machine Learning Research, vol. 12, no. Sep, pp. 2539–2561, 2011.

[7] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in ICML, 2016, pp. 2014–2023.

[8] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "Graph2vec: Learning distributed representations of graphs," CoRR, vol. abs/1707.05005, 2017. arXiv: 1707.05005.

[9] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem," in ACL-IJCNLP, 2015, pp. 1702–1712.

[10] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, "Graph representation learning via graphical mutual information maximization," in Proceedings of The Web Conference 2020, 2020, pp. 259–270.

[11] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," ICLR, 2020.

[12] R. Renner and U. Maurer, "About the mutual (conditional) information."

[13] L. Paninski, "Estimation of entropy and mutual information," Neural computation, vol. 15, no. 6, pp. 1191–1253, 2003.

[14] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," ICLR, 2019.

[15] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in Advances in neural information processing systems, 2018, pp. 4800–4810.