Datasets, Tasks, and Training Methods for Large-Scale Hypergraph Learning

Sunwoo Kim [1,*], Dongjin Lee [2,*], Yul Kim [3], Jungho Park [3], Taeho Hwang [3], Kijung Shin [1,2]

[1] Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea,
[2] School of Electrical Engineering, Daejeon, South Korea,
[3] Samsung Research, Seoul, South Korea

kswoo97@kaist.ac.kr, dongjin.lee@kaist.ac.kr, yul5.kim@samsung.com,
j0106.park@samsung.com, taeho.hwang@samsung.com, kijungs@kaist.ac.kr

Abstract

Relations among multiple entities are prevalent in many fields, and hypergraphs are widely used to represent such group relations. Hence, machine learning on hypergraphs has received considerable attention, and especially much effort has been made in neural network architectures for hypergraphs (a.k.a., hypergraph neural networks). However, existing studies mostly focused on small datasets for a few single-entity-level downstream tasks and overlooked scalability issues, although most real-world group relations are large-scale. In this work, we propose new tasks, datasets, and scalable training methods for addressing these limitations. First, we introduce two pair-level hypergraph-learning tasks to formulate a wide range of real-world problems. Then, we build and publicly release two large-scale hypergraph datasets with tens of millions of nodes, rich features, and labels. After that, we propose PCL, a scalable learning method for hypergraph neural networks. To tackle scalability issues, PCL splits a given hypergraph into partitions and trains a neural network via contrastive learning. Our extensive experiments demonstrate that hypergraph neural networks can be trained for large-scale hypergraphs by PCL while outperforming 16 baseline models. Specifically, the performance is comparable, or surprisingly even better than that achieved by training hypergraph neural networks on the entire hypergraphs without partitioning.

Keywords: Large-Scale Hypergraph Datasets, Scalable Hypergraph Learning, Hypergraph Neural Networks, Contrastive Learning, Partitioning

1 Introduction

Beyond pairwise relations among entities, understanding and modeling higher-order relations have received considerable attention [1, 2, 3, 4, 5, 6, 7, 8]. A hypergraph, which generalizes a graph, is a data structure that is commonly used to model such higher-order relations [9, 10, 11]. While an edge in a graph joins only two entities, a hyperedge joins an arbitrary number of entities, which makes a hypergraph, which is a pair of a node set and a hyperedge set, inherently capable of capturing high-order relations.

[*] Equal contribution.

Due to the omnipresence of hypergraphs, a number of machine learning tasks, such as node classification [12, 13, 14, 15] and hyperedge prediction [16, 17, 18], have been considered for hypergraph-structured data. One possible approach to tackle such tasks is to transform hypergraphs into ordinary graphs and apply existing graph representation models (e.g., graph neural networks) [19, 20, 21]. On the other hand, previous studies have shown that this transition can result in the loss of significant higher-order information, causing performance degradation in machine learning tasks [22, 23, 16, 18]. These studies highlight the need for specialized representation models specifically designed for hypergraphs.

Numerous *hypergraph neural network models* (e.g., HGNN [23], NHP [16], UniGCNII [12], and AllSet [14]) have been developed in recent years. Despite these advancements, the evaluation of these models has been limited to small datasets, usually consisting of tens of thousands of nodes and a few single-entity level downstream tasks. Moreover, scalability for large-scale datasets has been overlooked in prior research, with the majority of studies focusing on enhancing the expressive power of these networks.

However, many real-world applications require predicting properties beyond the single-entity level, such as pairs or groups of entities. Examples include detecting collusion among users in peer review [24], recommending products to users [25], identifying the same users on online social networks [26], and predicting pairwise differences for chemical discovery [27], to name a few. In addition, a massive number of group relations can be found across various domains, including over 200 million co-authorship relations in research papers [28], co-appearances of hashtags in over 500 million posts on social media [29], and over 10 million group discussions on an online Q&A platform [1].

To bridge the apparent gap between previous studies and practical applications, we present new tasks, datasets, and scalable training methods for large-scale hypergraph learning. Our contributions toward these goals are summarized as follows:

- **Novel Pair-level Tasks:** We present two new pair-level prediction tasks, hyperedge disambiguation and local clustering, and demonstrate that they can be used to address various real-world problems.
- **Large-scale Datasets:** We construct and publicly release two large-scale hypergraph-structured datasets: AMiner and MAG, which contain 10 million nodes and 20 million nodes respectively. These datasets are equipped with rich features and labels.
- **Scalable Training Schemes:** We propose PCL (**P**artitioning-based **C**ontrastive **L**earning), a scalable learning method for hypergraph neural networks (HNNs). In a nutshell, PCL partitions the input hypergraph and trains HNNs via contrastive learning while loading only one partition into memory at a time. PCL is also equipped with additional techniques to reduce information loss due to partitioning, and as a result, HNNs trained by PCL show surprisingly good performance in our experiments. Specifically, the performance is comparable to and often even better than that achieved by training HNNs on entire hypergraphs without partitioning, which is not scalable to large-scale datasets.

The structure of this paper is as follows. In Section 2, we provide some preliminaries and related work. In Section 3, we propose two pair-level prediction tasks with their mathematical formulation and real-world applications. In Section 4, we describe the two large-scale hypergraph datasets that we build. In Section 5, we present PCL, our scalable learning method for hypergraph neural networks. Using all the above, we perform experiments and report the results in Section 6. In Section 7, we offer a conclusion of our work.

Code and Data Availability: The source code used in this paper and the large-scale hypergraph datasets that we build are publicly available at `https://github.com/kswoo97/pcl` for reproducibility.

2 Preliminaries and Related Work

2.1 Preliminaries

We first give some preliminaries on hypergraphs and hypergraph neural networks. See Table 1 for a list of the frequently-used symbols.

2.1.1 Hypergraphs and Notations

A hypergraph generalizes a graph by allowing edges to join an arbitrary number of nodes. Consider a *hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ where $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes, $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ is the set of hyperedges, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$ is the node feature matrix. Each hyperedge $e \in \mathcal{E}$ is a non-empty subset of $\mathcal{V}$ (i.e., $e \subseteq \mathcal{V}$ and $e \neq \emptyset$), and we use $\boldsymbol{x}_i = \mathbf{X}[i,:]^T \in \mathbb{R}^F$ to denote the feature vector of the node v_i. The topological information in a hypergraph can also be represented in the form of a matrix called an *incidence matrix*. In the incidence matrix $\mathbf{H} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ of $\mathcal{H}$, each (i,j)-th entry has a value of 1 if a node v_i is incident to a hyperedge e_j (i.e., $h_{ij} = 1$ if $v_i \in e_j$), or has a value of zero (i.e., $h_{ij} = 0$) otherwise. That is, $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ can be denoted equivalently by $\mathcal{H} = (\mathbf{X}, \mathbf{H})$.

2.1.2 Hypergraph Neural Networks (HNNs)

Hypergraph neural networks (HNNs) use the hypergraph structure $\mathbf{H}$, node features $\mathbf{X}$, and (optionally) hyperedge features $\mathbf{Y}$ to learn representations of nodes $\mathbf{P}$ and/or of hyperedges $\mathbf{Q}$. Most of the modern HNNs [23, 13, 30, 22, 31, 14] follow a two-stage message aggregation strategy: node-to-hyperedge and hyperedge-to-node message aggregation. They iteratively update (a) the representation of a hyperedge by aggregating messages from its incident nodes and (b) the representation of a node by aggregating messages from its incident hyperedges. Let $\mathbf{P}^{(k)} \in \mathbb{R}^{|\mathcal{V}| \times F'}$ and $\mathbf{Q}^{(k)} \in \mathbb{R}^{|\mathcal{E}| \times F''}$ be the hidden representations of the nodes and hyperedges at the k-th layer, respectively. Formally, the k-th layer of HNNs uses the following update rules:

$$\begin{aligned} \boldsymbol{q}_j^{(k)} &= f_{\mathcal{V} \to \mathcal{E}}^{(k)}\Big(\boldsymbol{q}_j^{(k-1)}, \{\boldsymbol{p}_i^{(k-1)} : v_i \in e_j\}\Big), \\ \boldsymbol{p}_i^{(k)} &= f_{\mathcal{E} \to \mathcal{V}}^{(k)}\Big(\boldsymbol{p}_i^{(k-1)}, \{\boldsymbol{q}_j^{(k)} : v_i \in e_j\}\Big), \end{aligned} \tag{1}$$

where $f_{\mathcal{V} \to \mathcal{E}}(\cdot)$ and $f_{\mathcal{E} \to \mathcal{V}}(\cdot)$ are the message aggregation functions, and the initial node representation is identical to the node feature vector (i.e., for a node v_i, $\boldsymbol{z}_i^{(0)} = \boldsymbol{x}_i$ holds). Throughout this paper, we use HGNN [23] whose message aggregation strategies are as follows:

$$\boldsymbol{q}_j^{(k)} = \sum_{v_i \in e_j} \frac{\boldsymbol{p}_i^{(k-1)}}{\sqrt{d_i}}, \quad \boldsymbol{p}_i^{(k)} = \sigma\Bigg(\frac{1}{\sqrt{d_i}} \sum_{e_j : v_i \in e_j} \frac{w_j \boldsymbol{q}_j^{(k)} \boldsymbol{\Theta}^{(k)}}{\delta_j} + \boldsymbol{b}^{(k)}\Bigg), \tag{2}$$

where (a) $\boldsymbol{\Theta}^{(k)}$ is a learnable weight matrix, (b) $\boldsymbol{b}^{(k)}$ is a learnable bias, (c) d_i and δ_j are the degrees of each node v_i and hyperedge e_j, respectively, and (d) w_j is a positive weight assigned to each hyperedge $e_j \in \mathcal{E}$. For each node $v_i \in \mathcal{V}$, its degree is defined as $d_i = \sum_{j=1}^{|\mathcal{E}|} w_j h_{ij}$, and for each hyperedge $e_j \in \mathcal{E}$, its degree is defined as $\delta_j = \sum_{i=1}^{|\mathcal{V}|} h_{ij}$. In this work, w_j is fixed to 1 for simplicity. In general, node and hyperedge representations at the last K-th layer are considered as final representations used for downstream tasks.

Table 1: Frequently used symbols.

Notation	Definition
$\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$	hypergraph with nodes $\mathcal{V}$, hyperedges $\mathcal{E}$, and node features $\mathbf{X}$
$\mathcal{H} = (\mathbf{X}, \mathbf{H})$	hypergraph with node features $\mathbf{X}$ and the incidence matrix $\mathbf{H}$
$v_i \in \mathcal{V}, e_j \in \mathcal{E}$	node and hyperedge
$\mathbf{X} \in \mathsf{R}^{j\mathcal{V}j \quad F}$	node feature matrix
$\mathbf{H} \in \{0,1\}^{j\mathcal{V}j \quad j\mathcal{E}j}$	incidence matrix of $\mathcal{H}$
$Enc_\theta(\cdot)$	hypergraph encoder
$C_k \subseteq \mathcal{V}$	cluster of nodes
$\mathcal{P}_k = (\mathcal{P}_k^{\mathcal{V}}, \mathcal{P}_k^{\mathcal{E}})$	partition with nodes $\mathcal{P}_k^{\mathcal{V}}$ and hyperedges $\mathcal{P}_k^{\mathcal{E}}$
$\mathcal{P}_k = (\mathcal{P}_k^{\mathbf{X}}, \mathcal{P}_k^{\mathbf{H}})$	partition with node features $\mathcal{P}_k^{\mathbf{X}}$ and the incidence matrix $\mathcal{P}_k^{\mathbf{H}}$
$\|A\|$	cardinality (i.e., number of elements) of a set A

2.2 Related Work

In this subsection, we review previous studies on hypergraph neural networks, scalable (hyper)graph learning, contrastive learning, and (hyper)graph partitioning, all of which are closely related to our work.

2.2.1 Hypergraph Neural Networks

There has been intense attention on designing message aggregation rules of hypergraph neural networks (HNN): $f_{\mathcal{V}\,!\,\mathcal{E}}(\cdot)$ and $f_{\mathcal{E}\,!\,\mathcal{V}}(\cdot)$ (see Eq (1)). Many recent studies have focused on extending the applicability of graph neural networks (GNNs) to hypergraphs. Some approaches [23, 30, 16] replace each hyperedge by a clique composed of its constituent nodes (i.e., clique-expansion) and employ GNN-based message passing on the resulting graph, which is called the clique-expanded graph. While these models are simple and effective, they suffer from undesired information loss due to structural distortion caused by clique expansion [32, 33, 22]. To mitigate such information loss, HNHN [22] utilizes an approach based on star-expansion, which does not lead to any information loss, with two different weight matrices for node- and hyperedge-side message aggregations. Several studies attempt to generalize the message-passing process in GNNs and HNNs in a unified form [12, 34], and AllSet [14] generalizes message aggregation methods as multiset functions that are learned by DeepSets [35] or SetTransformer [36].

While many HNN models have been developed, the evaluation of their performance in most studies has primarily focused on single-entity-level prediction tasks, such as node classification [23, 12, 14, 15], hyperedge classification [22], and hyperedge prediction [16, 17]. Although these tasks are commonly used as benchmarks for machine-learning models, it is important to note that many real-world applications may not inherently align with these tasks, as explained in greater detail in Section 3.3.

2.2.2 Scalable (Hyper)graph Learning

As real-world graphs grow larger, many studies have been conducted to scale graph neural networks (GNNs) to large graphs through parallelism [37, 38, 39], graph sampling [20, 40, 41, 42, 43, 44], and pre-computed convolutional filters [45, 46]. Here, we focus on sampling-based approaches. Graph sampling, which approximates local graph structures by subsampled ones suitable for computation, has been demonstrated to be effective for scalable graph learning. For instance, GraphSAGE [20] utilizes uniform sampling of a fixed-size set of neighboring nodes to approximate local connectivity. Similarly, FastGCN [40] conducts node-level sampling independently for each layer while incorporating importance sampling to reduce variance. Instead

of node-wise sampling, some works build mini-batches by graph-level sampling. In ClusterGCN [43], non-overlapping clusters are computed at the pre-processing phase and form mini-batches cluster by cluster. Inter-cluster edges are simply disregarded, and this process enables ClusterGCN to avoid the "neighborhood expansion" problem [43]. GraphSAINT [44] adopts a graph sampling approach, and specifically, it uses sampling algorithms for variance reduction with an additional normalization technique for unbiasedness.

The representation learning method for large hypergraphs remains largely underexplored. HyperNetVec, a scalable unsupervised hypergraph embedding method proposed by Maleki et al. [47] leverages multi-level (hierarchical) embedding approaches that adopt existing graph embedding methods [48, 20] on a coarsened hypergraph. While HyperNetVec can handle hypergraphs with millions of hyperedges, it is a transductive embedding method. That is, it directly learns node embeddings and cannot be directly utilized for training HNNs. To the best of our knowledge, our study is the first to propose a scalable training approach for HNNs applicable to hypergraphs with millions of nodes and hyperedges. It should be noticed that our approach is scalable enough to be used with hypergraphs with tens of millions of nodes and hyperedges.

2.2.3 Contrastive Learning (CL)

Due to its effectiveness and generality, contrastive learning (CL) has emerged as a novel solution for alleviating the label-scarcity issue in representation learning in various domains, including computer vision [49, 50], natural language processing [51], graph learning [21, 52, 53, 54, 55, 56], and hypergraph learning [57, 58, 59, 15]. The basic concept of CL is to (a) create two augmented views from the input data and (b) learn an encoder to maximize the agreement between the two views. That is, CL approaches aim to minimize (maximize) the distance between positive (negative) pairs. The learned representations can be utilized for various downstream tasks, such as node classification [21, 53, 15] and recommendation [60, 61].

Among contrastive learning for hypergraphs [57, 58, 59, 15]. S^2-HHGR [57] uses a coarse- and fine-grained node dropout for hypergraph augmentation, and it remedies a data sparsity issue for group recommendation. DHCN [58] employs session-level contrast for session recommendation. TriCL [15] uses a tri-directional contrastive loss, which combines node-, group-, and membership-level contrastive losses, resulting in better performance on several downstream tasks, compared to employing simply a node-level contrastive loss.

2.2.4 (Hyper)graph Partitioning

(Hyper)graph partitioning is a fundamental task where the objective is to divide nodes into multiple groups [62, 63, 64] to minimize the connectivity between groups, and it has extensive applications, including anomaly detection [65], molecular mining [66], and face analysis [67].

Especially for hypergraph partitioning, multi-level approaches have received intensive attention [63, 68, 69]. A multi-level hypergraph partitioning algorithm consists of three phases: coarsening, initial partitioning, and uncoarsening. *(a) Coarsening*: A coarsened hypergraph $\mathcal{H}^{(c)}$ is formed by merging pairs of nodes in the input hypergraph $\mathcal{H}$. This procedure is recursively applied to the coarsened hypergraph. The final hypergraph is the coarsest one that meets predefined termination criteria. *(b) Initial partitioning*: The coarsest hypergraph is partitioned using any partitioning rules. *(c) Uncoarsening*: Partitions found in the second phase are successively projected back towards the original hypergraph $\mathcal{H}$. In this paper, we adopt PaToH [63] for hypergraph partitioning due to its high-scalability. However, other approaches [68, 69, 70, 71] can be employed instead.

While the above hypergraph partitioning (clustering) studies make assumptions about static input hypergraphs and disjoint partitions, there have been efforts to relax these assumptions in ordinary graphs.

Refer to [72, 73] for surveys on this topic. Notably, such extensions include overlapping clustering, which allows entities to belong to multiple clusters [74, 75], clustering in time-varying graphs [76], and overlapping clustering in time-varying graphs [77]. Recently, overlapping clustering is addressed using graph neural networks [75] and also considered in hypergraphs [78].

3 Proposed Hypergraph Learning Tasks

In this section, we propose two new hypergraph learning tasks: *hyperedge disambiguation* and *local clustering*. For each task, we first provide its mathematical formulation and then discuss its application to real-world problems. Lastly, we describe how our proposed tasks differ from commonly considered benchmark tasks, with a focus on advantages in addressing potential issues in practice.

3.1 Hyperedge Disambiguation

3.1.1 Mathematical Description

Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and a subset $S \subseteq \mathcal{E}$ of hyperedges. Every hyperedge $e_i \in S$ is split (disjointly or with partial overlap) into two hyperedges e_{i1} and e_{i2} so that $e_{i1} \cup e_{i2} = e_i$ holds. The hyperedges in S are replaced by these split hyperedges, which results in a new hypergraph $\mathcal{H}' = (\mathcal{V}, \mathcal{E}')$ where

$$\mathcal{E}' = (\mathcal{E} \setminus S) \cup \bigcup_{e_i \in S} \{e_{i1}, e_{i2}\}. \tag{3}$$

Based on this setting, *hyperedge disambiguation* is defined as to predict whether a given pair of hyperedges in the given hypergraph $\mathcal{H}' = (\mathcal{V}, \mathcal{E}', \mathbf{X})$ are split hyperedges or not. Formally,

$$Y(\{e_i, e_j\}) = \begin{cases} 1 & e_i \cup e_j \in S \\ 0 & \text{otherwise} \end{cases}, \tag{4}$$

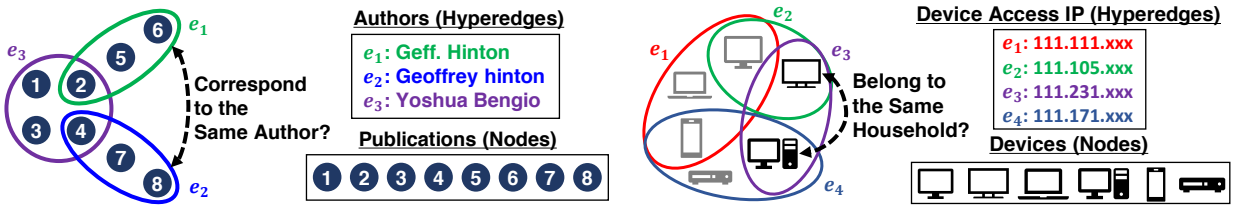
where $e_i, e_j \in \mathcal{E}'$. The goal of this task is to learn a function $f : \binom{E'}{2} \mapsto \mathsf{R} \in [0, 1]$ to approximate $Y(\cdot)$ in Eq (4). In this paper, we consider this task in a semi-supervised setting where a small amount of the ground-truth split pairs of hyperedges are given. However, this problem can also be considered in other settings.[1]

3.1.2 Real-world Applications

The hyperedge disambiguation task can be applied to many real-world applications. Below, we introduce two examples: **researcher disambiguation** and **user identification**.

Researcher disambiguation is a task to identify whether a given pair of researchers are in fact the same researcher or not. Due to namesakes and other reasons, one researcher can be represented as multiple individuals in a system, which complicates searching for experts, surveying related papers, and recommending scholarly papers [79, 80, 81]. This problem can naturally be formulated as a hyperedge disambiguation task on a publication-author hypergraph where nodes denote publications and each hyperedge represents a (potentially partial) set of publications authored by one researcher. Then, predicting whether a given pair of

[1] For example, in an unsupervised setting without any given positive example pairs, one can additionally split edges in $\mathcal{E}'$ to use them as positive example pairs for training.



(a) Applications of a task 1: Researcher disambiguation

(b) Applications of a task 2: Household matching

Figure 1: Example applications of the proposed tasks. (a) In a researcher disambiguation task, we aim to predict whether a given pair of researchers (modeled as hyperedges) are identical or not. (b) In a household matching task, we aim to predict whether a given pair of devices (modeled as nodes) are owned by people in the same household or not.

hyperedges are split ones or not corresponds to predicting whether the two sets of publications are authored in fact by the same researcher or not, as visually depicted in Figure 1(a).

The second task is **user identification**, which is to identify whether a given pair of electronic devices (or accounts for a web service) are owned by the same user [82, 83]. If such a pair can be identified, the data (e.g., behavior log) from both can be used together to improve the quality of services (e.g., personalization and recommendation). To handle this problem, we can build a hypergraph where nodes denote devices (or accounts) and each hyperedge represents the set of devices (or accounts) that each device (or account) communicates with. Then, predicting whether a given pair of hyperedges are split ones or not corresponds to predicting whether the two devices (or accounts) are owned by the same user or not.

3.2 Local Clustering

3.2.1 Mathematical Description

Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ and (disjoint or partially overlapping) clusters of nodes $\mathcal{C} = \{C_1, C_2, \cdots, C_{|\mathcal{C}|}\}$ where each cluster is a subset of nodes (i.e., $C_k \subseteq \mathcal{V}$, $\forall C_k \in \mathcal{C}$). Based on this setting, *local clustering* is defined as to predict whether a given pair of nodes in a given hypergraph $\mathcal{H}$ belongs together to the same cluster or not. Formally,

$$Y(\{v_i, v_j\}) = \begin{cases} 1 & \exists C_k \in \mathcal{C} \text{ such that } \{v_i, v_j\} \subseteq C_k \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where $v_i, v_j \in \mathcal{V}$. The goal of this task is to find a function $f : \binom{\mathcal{V}}{2} \mapsto \mathsf{R} \in [0, 1]$ to approximate $Y(\cdot)$ in Eq (5). In this paper, we consider this task in a semi-supervised setting where the members of a few clusters are given, while any information about the other clusters (e.g., the number of them) is not provided. However, this problem can also be considered in other settings.[2]

3.2.2 Real-world Applications

The local clustering task can be used to formulate a wide range of real-world problems. Here, we provide two examples: **sub-field detection** and **household matching**.

[2]For example, it can be considered in unsupervised settings especially when the clustering membership is strongly correlated with node features and/or topological information.

The first task is **sub-field detection**, which is to classify whether a given pair of publications belong to the same sub-field or not. Note that the boundary of a sub-field is often unclear, and thus a single publication may belong to multiple sub-fields. This problem is formulated as a local clustering task on hypergraphs where a node denotes a publication, a hyperedge represents the set of publications that are co-cited by one publication, and clusters indicate sub-fields.

The second task is **household matching**, a task to predict whether a given pair of electronic devices are owned by people in the same household or not. By grouping devices from the same households, recommendations and advertisements can be better personalized. For example, consider a hypergraph where nodes denote devices and each hyperedge indicates the set of devices access to the same IP within a specific time interval (e.g., set of devices access to IP 123.456.789 between 9 AM and 10 AM on May 16, 2023). If each cluster is formed by the devices from the same household, inferring whether two nodes belong to the same cluster or not corresponds to inferring whether two devices are from the same household or not, as visually depicted in Figure 1(b). Note that household matching is not formulated accurately as an entity classification task, since (1) it is hard to know in advance the number of households, which is required and fixed in entity classification tasks, and (2) the number of households changes over time.

3.3 Differences from Existing Tasks

In this subsection, we discuss some difficulties when formulating the aforementioned real-world applications as commonly-used single-entity-level tasks (e.g. node classification). Note that our proposed pair-level tasks mitigate such difficulties.

3.3.1 Entity Classification

One may suspect that the aforementioned applications can also be formulated as single-entity-level tasks. However, formulating these applications as our tasks offer several advantages over formulating them as an entity classification task: (1) our tasks do not require the number of labels. which is equivalent to the number of split hyperedges and the number of clusters in our tasks, in advance, and (2) a model for our tasks does not need to be retrained whenever the number of labels changes in time-evolving data.

For example, one may consider formulating the household matching problem as a node classification problem where each class indicates a household. Here, it is important to note that the number of such classes is typically impossible to know in advance, and it also changes ceaselessly over time. Moreover, no example can be provided for some classes during training. However, in node-classification formulation, the output dimension of classifiers should be predetermined by the number of classes, and they cannot output classes that are not observed during training.

On the other hand, our formulation as the local clustering task is free from such unrealistic requirements. That is, we do not require the number of households in advance and require only some pairs of devices from some households. Moreover, machine learning models trained for the local clustering task can naturally be generalized to households unseen during training.

Formulating the researcher disambiguation problem as a hyperedge classification task also encounters the same issues above. The number of real identities, which change over time, is impossible to know in advance. However, considering the researcher disambiguation problem as a hyperedge disambiguation task is free from the above-mentioned issues.

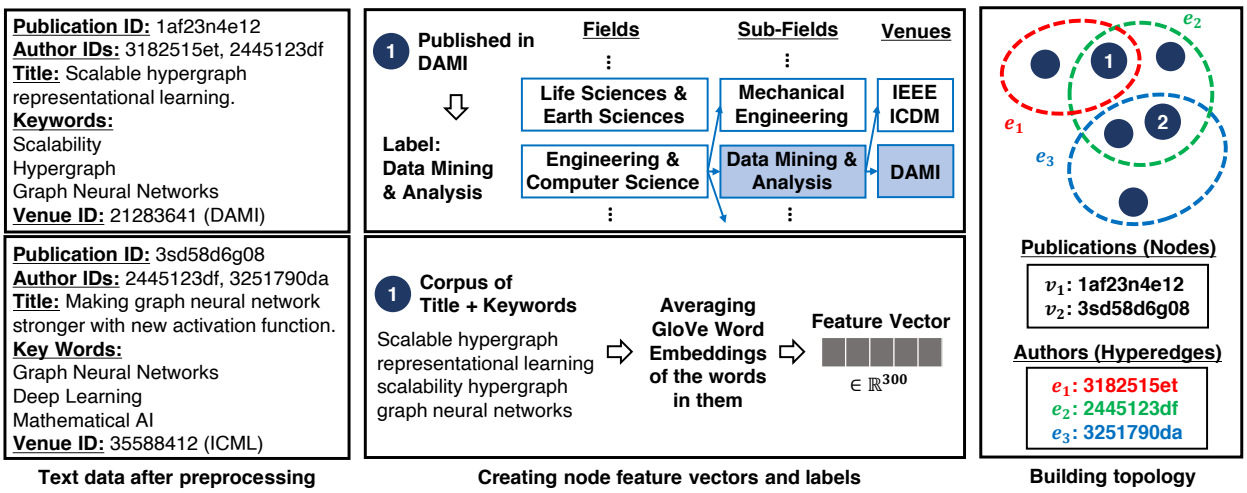


Figure 2: Examples for the construction process of the AMiner and MAG datasets. (**Left**) From raw text data, we extract the (a) publication ID, (b) author IDs, (c) title, (d) keywords, and (e) venue ID of each publication. (**Middle**) We obtain the 300-dimensional features for each publication by averaging the embeddings of all words in its title and keywords. Each publication is labelled based on the sub-field it belongs to. (**Right**) We create nodes corresponding to publications and hyperedges corresponding to authors.

3.3.2 Overlapping Clustering

The proposed local clustering problem is also closely related to the overlapping clustering problem, whose goal is to identify clusters where an object (e.g., a node) can belong to multiple clusters (see Section 2.2.4). The problem has been considered mostly in ordinary graphs and more importantly in transductive settings. That is, clusters are identified based on the current state of a graph, and once the graph changes, one needs to re-run the entire clustering process from scratch. Thus, the task is not proper to be used for evaluating HNNs (especially their generalization capabilities), which are particularly useful in inductive settings. Moreover, in many cases, the number of ground truth clusters is required, while it is unavailable in many realistic settings.

4 Proposed Large-scale Hypergraph Datasets

In this section, we introduce two large-scale hypergraph-structured datasets. We build them by processing raw data from AMiner [84] and Microsoft Academic Graph (MAG) [85]. First, we describe how the node features and labels are obtained from the raw data, and then we present how the hypergraph topologies are constructed. Figure 2 provides a visual description of the hypergraph construction process.

4.1 Extracting Basic Information

The source of the datasets is Open Academic Graph[3] [28], and it provides two large academic bibliographic datasets: AMiner and MAG. These bibliographic datasets consist of huge raw text data containing information about over 100 million authors and publications. From this raw data, we first extract the (a) publication ID, (b) author IDs, (c) the title, (d) keywords, and (e) the venue ID of each publication.

[3] Available at `https://www.aminer.cn/oag-2-1`.

Table 2: Statistics of five real-world hypergraph datasets. MAG has 158× more nodes and 129× more hyperedges than Trivago, a commonly used hypergraph dataset.

Source	Dataset	$\|V\|$	$\|E\|$	# features	# classes
Existing	DBLP	41,302	22,363	1,425	6
	Trivago*	172,738	233,202	300	160
Transformed**	OGBN-MAG	736,389	1,134,649	128	349
Proposal	AMiner	13,262,573	22,552,647	300	257
	MAG	27,320,375	30,175,013	300	257

* In the Trivago dataset, we use the first 300 left singular vectors of the incidence matrix as the node features.
** Transformed from a heterogeneous graph.

4.2 Building Node Features

We construct a 300-dimensional feature vector for each publication, which corresponds to a node. To this end, we average the embeddings of all the words appearing in its title and keywords, where the word embeddings are obtained by the pre-trained GloVe [86] model[4]. The node features are created separately for AMiner and MAG.

4.3 Building Node Labels

We classify publications based on the academic fields they belong to. Since the original raw data does not contain any information about the publications' academic fields, we utilize the venue name (e.g., conference and journal) of each publication to infer its field. Specifically, we use the category hierarchy provided by Google Scholar[5] for mapping venue names to academic fields. The hierarchy consists of three levels: fields, sub-fields, and venues. For example, the journal *Data Mining and Knowledge Discovery* (DAMI) belongs to the sub-field *Data Mining & Analysis*, which in turn belongs to the field *Engineering & Computer Science*. We label each publication based on the sub-field (i.e., the middle level) it belongs to. Note that, in the category that we use, each venue (node) is matched with exactly one sub-field (label).

4.4 Building Hypergraph Structures

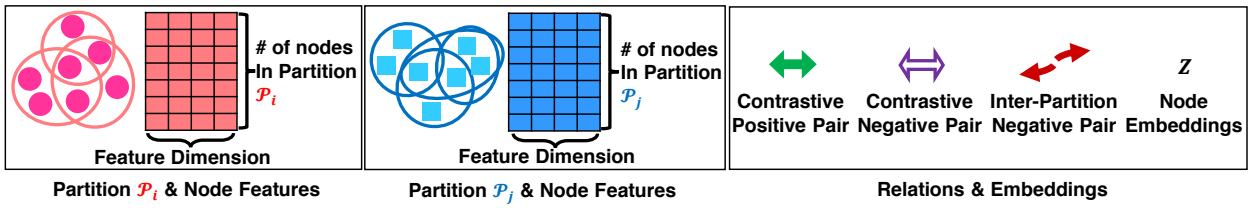
The raw data are far from complete with many missing values (e.g., missing authors) and meaningless information (e.g., incomprehensible words). For the quality of the created hypergraphs, we first filter out all publications that satisfy at least one of the following three conditions:

- *It does not have author information (i.e., names or IDs),*
- *Its title and keywords consist of less than three words,*
- *It is not matched with any academic field.*

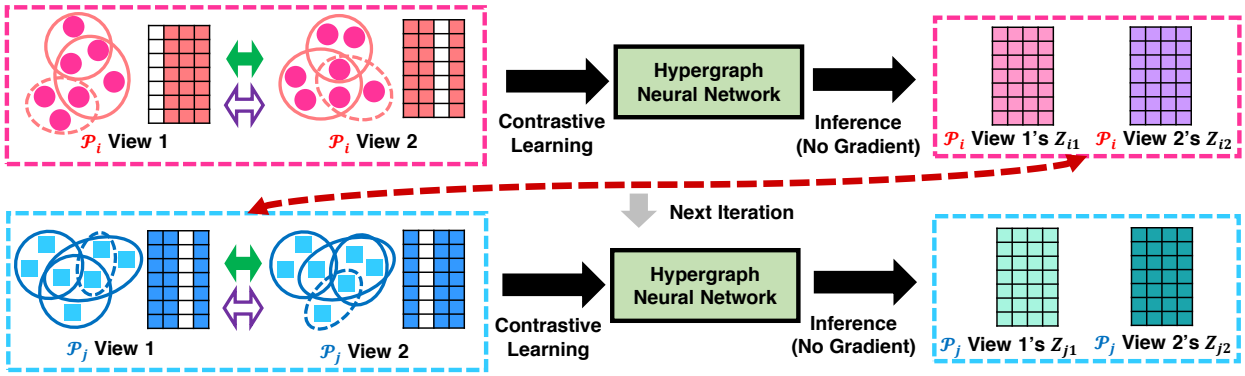
Then, we build hypergraphs using the remaining publications.

As mentioned in Section 4.2, publications and authors are represented by nodes and hyperedges, respectively. To be specific, for each author ID, the publications that contain the ID as an author form a

[4]Available at `https://github.com/UKPLab/sentence-transformers`.
[5]Available at `https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng`. In this taxonomy, a single venue is associated with a single sub-field.



(a) Notations & Symbols



(b) Proposed methods: Basic PCL and PCL+PINS

Figure 3: Overview of Basic PCL and PCL+PINS, the proposed large-scale hypergraph learning methods. We consider each partition ($\mathcal{P}_i$ and $\mathcal{P}_j$ in the example) as a mini-batch and perform contrastive learning on each mini-batch. In Basic PCL, after a single contrastive learning iteration is done on a partition ($\mathcal{P}_i$), it moves to another partition ($\mathcal{P}_j$) and starts the next iteration. In PCL+PINS, after a model is updated on a partition ($\mathcal{P}_i$), PCL+PINS extracts the node embeddings of $\mathcal{P}_i$'s two views ($\mathbf{Z}_{i1}$ and $\mathbf{Z}_{i2}$ in our example) and uses $\mathbf{Z}_{i1}$ and $\mathbf{Z}_{i2}$ to compute inter-partition CL loss in the next partition $\mathcal{P}_j$.

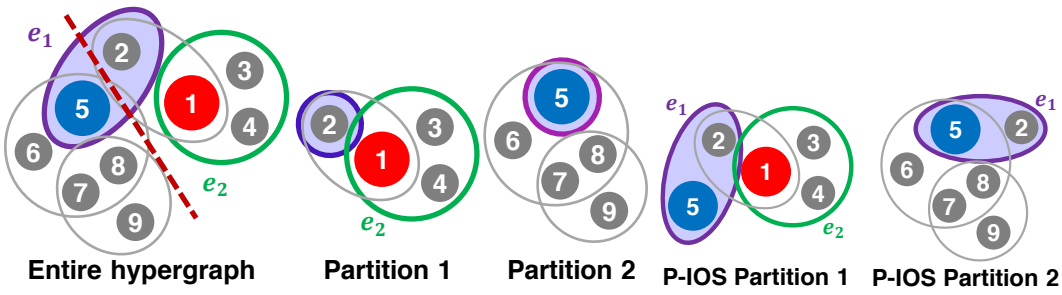
hyperedge. In this way, we create two co-authorship hypergraphs, AMiner and MAG, which contain over 20 million hyperedges and 30 million hyperedges, respectively. Numerically, the MAG dataset contains 158× more nodes and 129× more hyperedges than the Trivago dataset, one of the commonly used hypergraph datasets. Table 2 compares the statistics of the proposed datasets and some existing datasets. Note that, the AMiner and MAG datasets can be used not only for the proposed pair-level tasks but also for common downstream tasks, such as node classification, hyperedge prediction, and clustering.

5 Proposed Scalable Hypergraph Learning Method

In this section, we introduce PCL (Partitioning-based Contrastive Learning), a scalable hypergraph learning algorithm for pair-level tasks. We first provide an overview of PCL together with the rationale of its components. Next, we describe the details of PCL. At last, we give the complexity analysis of PCL. A pictorial description of PCL is provided in Figure 3.

5.1 Challenges and Main Ideas

We analyze two challenges encountered in training representation learning models for pair-level downstream tasks on large-scale hypergraphs. Then, we present our proposed solutions for overcoming these problems.



(a) Hypergraph and resulting partitions: depicting difficulties of representation learning on partitions.

(b) An example of P-IOS.

Figure 4: (a): Assume the two nodes v_1 and v_5 belong to the same cluster, and the two hyperedges e_1 and e_2 are split from one original hyperedge. Note that v_1 and v_5 belong to different partitions, and $e_1 = \{v_2, v_5\}$ does not even exist in any partitioned hypergraph since $v_2 \in \mathcal{P}_1^V$ and $v_5 \in \mathcal{P}_2^V$. Thus, $\{v_1, v_5\}$ and $\{e_1, e_2\}$ cannot be used as labeled pairs when one trains a model on partitioned hypergraphs in a supervised manner. (b): In order to mitigate information loss caused by partitioning, P-IOS restores each hyperedge by adding its missing nodes back to the hyperedge.

Challenge 1. Considerable Time and Space Cost: How can we train hypergraph neural network models on large-scale hypergraph datasets? The commonly-used full-batch training is typically not possible, since such hypergraphs cannot be entirely loaded into GPU memory, and this causes the out-of-memory problem. To mitigate this issue, a mini-batch training method [20] loads only the subgraph from which each node aggregates messages into GPU memory at a time. Although it reduces memory requirements, it introduces significant computational overhead to extract such subhypergraphs for each node and load them repeatedly into GPU memory. Even worse, such subhypergraphs are often large since real-world hypergraphs tend to have a small diameter [87, 3].

Solution 1: Our solution to the scalability issue in large-scale hypergraph representation learning is ***partitioning*** (or equivalently node clustering). The process involves dividing the input hypergraph into smaller partitions, treating each partition as a distinct mini-batch. Partitioning-based approaches have proved efficient in training models (i.e., graph neural networks) for representation learning in ordinary graphs, mitigating the neighborhood expansion problem [43]. Despite their efficiency, partitioning-based approaches have not been considered for training HNNs on large-scale hypergraphs. Motivated by this efficiency, we split the entire hypergraph into K partitions. Here, we use a well-known hypergraph partitioning algorithm PaToH[6] [63], while any partitioning method can be used instead. We load a single partition into GPU memory at a time, which requires K times more processes of loading and unloading partitions into GPU memory. Alternatively, one can put two different partitions into GPU memory, but it requires at most $K(K-1)/2$ times more processes, which are computationally too costly for large K.

Challenge 2. Information Loss Caused by Partitioning: Although the partitioning approach can mitigate the scalability issues, it has several limitations in handling pair-level downstream tasks presented in Section 3.

- Some hyperedges do not exist in any partitioned hypergraph if their constituent nodes belong to different partitions, and thus such hyperedges cannot be utilized in the training phase (see e_1 of Figure 4 (a)).

[6]PaToH is a balanced partitioning method. It ensures that all generated partitions are of similar sizes [63], specifically satisfying $|\mathcal{P}_k^V| \leq \frac{(1+\epsilon)}{|P|} \sum_{i=1}^{|P|} |\mathcal{P}_i^V|, \ \forall k = 1, \cdots, |\mathcal{P}|$. As shown in Table 8 in Section 6.3.5, partitions obtained by PaToH from real-world hypergraphs are well balanced. Specifically, the standard deviation of the number of nodes in each partition is less than 0.5% of the average number of nodes per partition.

- Certain node pairs and hyperedge pairs that are given as labeled pairs may be split into different partitions (see $\{v_1, v_5\}$ of Figure 4 (a)). In such scenarios, these divided pairs cannot be used as labeled pairs for the partitioning-based approach to train HNNs.
- A single partition may not represent the entire hypergraph adequately since partitioning algorithms tend to group similar nodes together in the same partition [43, 88]. In other words, each partition may exhibit bias.

Solution 2: Our countermeasure for the second challenge is three-fold.

- The first one is to leverage a **contrastive learning** (CL) framework [49, 15] to train hypergraph encoders (i.e., HNNs) on partitioned hypergraphs. The basic concept of CL is to train a hypergraph encoder by (a) creating two augmented views from the input data and (b) letting the encoder maximize the agreement between the two views. Note that CL does not require any label supervision, and this characteristic ensures that the encoder is properly trained even in the case where labeled pairs are split or lost.
- The second one is **PINS**, which mitigates the bias problem caused by hypergraph partitioning. Since it processes a single partition as a mini-batch, a hypergraph encoder has limited exposure to the differences between nodes in different partitions. To enable the hypergraph encoder to incorporate information from other partitions, PINS utilizes the node representations from the previous partition.
- The last one is **P-IOS**, restores each hyperedge by adding its missing nodes back to the hyperedge, to alleviate information loss caused by partitioning.

5.2 Details of Proposed Methods

In this subsection, we explain the details of the three proposed solutions discussed in the previous subsection. We first introduce **Basic PCL**, a contrastive learning (CL) framework with partitioned hypergraphs. Then, we introduce PCL + PINS, a method that uses the embeddings of the previous partitions as negative samples. In addition, we provide PCL + P-IOS, a method that augments each partition without additional hyperedges. Lastly, we describe how a classifier of downstream tasks is trained with pre-trained hypergraph encoders.

5.2.1 Basic PCL

Basic PCL (**P**artitioning-based **C**ontrastive **L**earning) is a contrastive learning framework that trains HNNs on partitioned hypergraphs. An example of Basic PCL is illustrated in Figure 3, and its pseudocode is provided in Algorithm 1. We denote the set of all partitions as $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \cdots, \mathcal{P}_K\}$, where each partition is defined as $\mathcal{P}_k = (\mathcal{P}_k^{\mathsf{V}}, \mathcal{P}_k^{\mathsf{E}})$, where $\mathcal{P}_k^{\mathsf{V}} \subseteq \mathcal{V}$ and $\mathcal{P}_k^{\mathsf{E}} = \{e \cap \mathcal{P}_k^{\mathsf{V}} : e \in \mathcal{E}, e \cap \mathcal{P}_k^{\mathsf{V}} \neq \emptyset\}$. Here, K denotes the number of partitions[7]. Note that $\mathcal{P}_k$ can also be expressed as $(\mathcal{P}_k^{\mathbf{X}}, \mathcal{P}_k^{\mathbf{H}})$, where $\mathcal{P}_k^{\mathbf{X}}$ denotes the node features of $\mathcal{P}_k^{\mathsf{V}}$, and $\mathcal{P}_k^{\mathbf{H}}$ denotes the incidence matrix of $\mathcal{P}_k$.

In a nutshell, Basic PCL receives a set of partitions $\boldsymbol{\mathcal{P}}$ as an input and treats each partition $\mathcal{P}_k \in \boldsymbol{\mathcal{P}}$ as a mini-batch (line 3) for contrastive learning. It trains a hypergraph encoder (i.e., hypergraph neural network) aiming to minimize a contrastive learning loss, which will be further elaborated. Details are as follows:

[7] One can set K based on the available amount of space (low K takes more memory consumption in general). Note that the performance of the proposed method is not significantly affected by K, which will be demonstrated in Section 6.3.5.

Algorithm 1 Basic PCL and PCL + PINS: Partitioned contrastive learning for hypergraph

Input: (1) Hypergraph encoder Enc_θ, projection head $Proj_\Theta$
(2) Partitioned hypergraph $\boldsymbol{\mathcal{P}} = f\mathcal{P}_1, \quad , \mathcal{P}_K g$
(3) Feature augmentation prob. p_v, hyperedge augmentation prob. p_e
(4) Number of epochs EP, Optimizer $Optim$, Number of negative samples N
(5) PCL+PINS scalar λ (if $\lambda = 0$; then it is Basic PCL)

Output: Pretrained hypergraph encoder Enc_θ

1 **for** $ep \quad 1, \quad , EP$ **do**
2 Shuffle the order of partitions in $\boldsymbol{\mathcal{P}}$
3 **foreach** $\mathcal{P}_k$ *2* $\boldsymbol{\mathcal{P}}$ **do**
4 $\mathcal{P}^0_{k1} \quad Au(\mathcal{P}_k, p_v, p_e)$; $\mathcal{P}^0_{k2} \quad Au(\mathcal{P}_k, p_v, p_e)$ ▷ Augmentation
5 $\mathbf{P}_{k1} \quad Enc_\theta(\mathcal{P}^0_{k1})$; $\mathbf{P}_{k2} \quad Enc_\theta(\mathcal{P}^0_{k2})$ ▷ Node embedding
6 $\mathbf{Z}_{k1} \quad Proj_\Theta(\mathbf{P}_{k1})$; $\mathbf{Z}_{k2} \quad Proj_\Theta(\mathbf{P}_{k2})$ ▷ Projection
7 $Loss \quad \mathcal{L}_{CL,k}$ (use $\mathbf{Z}_{k1}$, $\mathbf{Z}_{k2}$, and N) ▷ Contrastive loss in Eq (10)
8 **if** $(\lambda > 0)$ ^ $(ep > 1)$ **then**
9 $Loss \quad Loss + \lambda \sum_{i=1,j=1}^{2,2} \mathcal{L}_{I,k}(i,j)$ ▷ Inter-partition loss in Eq (11)
  (use $\mathbf{Z}^0_1$ and $\mathbf{Z}^0_2$ as node embeddings of two views of the previous partition)
10 Update Enc_θ and $Proj_\Theta$ by $Optim(Loss, \theta, \Theta)$ ▷ Back propagation
11 **if** $\lambda > 0$ **then**
12 $\mathbf{Z}^0_1 \quad Proj_\Theta(Enc_\theta(\mathcal{P}^0_{k1}))$; $\mathbf{Z}^0_2 \quad Proj_\Theta(Enc_\theta(\mathcal{P}^0_{k2}))$ ▷ Only for PINS
13 **return** Enc_θ

First, Basic PCL creates two augmented views of an input partition $\mathcal{P}_k$ (line 4). It mainly utilizes a masking-based augmentation strategy [15], which corrupts node features and membership (each group and its members). For node feature augmentation, Basic PCL employs a single random binary mask, with each entry sampled from a Bernoulli distribution $B(1 - p_v)$. This mask is utilized to set certain columns of $\mathcal{P}^{\mathbf{X}}_k$, which is a feature matrix of the current partition, to zero vectors. Similarly, for membership augmentation, Basic PCL randomly removes nodes from the hyperedges that they belong to. Specifically, Basic PCL generates a binary mask of size $nnz(\mathcal{P}^{\mathbf{H}}_k)$ (i.e., the number of nonzero entries of an incidence matrix $\mathcal{P}^{\mathbf{H}}_k$ of the current partition) with each entry sampled from a Bernoulli distribution $B(1 - p_e)$. This mask is leveraged to remove certain node-hyperedge memberships. We use $\mathcal{P}^0_{k1}$ and $\mathcal{P}^0_{k2}$ to denote the two resulting views created through this process, as in Eq (6):

$$\mathcal{P}^0_{k1} = (\mathcal{P}^{0\mathbf{X}}_{k1}, \mathcal{P}^{0\mathbf{H}}_{k1}) \leftarrow Au(\mathcal{P}_k, p_v, p_e); \mathcal{P}^0_{k2} = (\mathcal{P}^{0\mathbf{X}}_{k2}, \mathcal{P}^{0\mathbf{H}}_{k2}) \leftarrow Au(\mathcal{P}_k, p_v, p_e), \tag{6}$$

where Au denotes the augmentation process described above.

Then, Basic PCL obtains node representations of the two views by sequentially passing the two views through a hypergraph encoder Enc_θ (see Section 2.1.2 for details) and a projection head $Proj_\Theta$ (line 5-6):

$$\mathbf{Z}_{k1} = Proj_\Theta(Enc_\theta(\mathcal{P}^0_{k1}))); \mathbf{Z}_{k2} = Proj_\Theta(Enc_\theta(\mathcal{P}^0_{k2}))), \tag{7}$$

where $Proj_\Theta$ is a one-layer MLP with the ReLU [89] activation function. Lastly, PCL employs the InfoNCE loss [90] for parameter updates to make the representations of the corresponding nodes in the two views similar and those of different nodes dissimilar[8]:

$$\ell(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i}) = -\log \frac{\exp\left(s(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i})/\tau\right)}{\sum_{t=1}^{j\mathcal{P}^{\mathcal{V}}_k j} \exp\left(s(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,t})/\tau\right)}, \tag{8}$$

[8]Note that other self-supervised losses (e.g., [91]) can be used alternatively.

where $\mathbf{z}_{ka,i}$ is a i^{th} row vector of $\mathbf{Z}_{ka}$, τ is a temperature parameter, and s is a similarity function. As the similarity function s, we use the consine similarity, i.e., $s(\mathbf{u},\mathbf{v}) = (\mathbf{u}^T\mathbf{v})/(\|\mathbf{u}\| \cdot \|\mathbf{v}\|))$. As shown in the denominator of Eq (8), for each node, its similarity with all other nodes of $\mathcal{P}_k^{\mathsf{V}}$ is computed. For computational efficiency, instead of using the entire $\mathcal{P}_k^{\mathsf{V}}$, we obtain a uniform sample $\mathcal{P}_k^{\emptyset\mathsf{V}} \subset (\mathcal{P}_k^{\mathsf{V}} \setminus \{v_i\})$, where $|\mathcal{P}_k^{\emptyset\mathsf{V}}| = \min(|\mathcal{P}_k^{\mathsf{V}}|, N)$ and N is a hyperparameter, and use it as negative samples. That is,

$$\ell(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i}) = -\log \frac{\exp\left(s(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i})/\tau\right)}{\exp\left(s(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i})/\tau\right) + \sum_{v_t \mathsf{2} P_k^{\mathcal{V}}} \exp\left(s(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,t})/\tau\right)}. \tag{9}$$

In practice, we symmetrize Eq (8) so that the final CL loss (line 7) becomes:

$$\mathcal{L}_{CL,k} = \frac{1}{2|\mathcal{P}_k^{\mathsf{V}}|} \sum_{i=1}^{j P_k^{\mathcal{V}} j} \left(\ell(\mathbf{z}_{k1,i}, \mathbf{z}_{k2,i}) + \ell(\mathbf{z}_{k2,i}, \mathbf{z}_{k1,i})\right). \tag{10}$$

By using Eq (10), Basic PCL updates Enc_θ and $Proj_\Theta$ via gradient descent.

5.2.2 PCL + PINS

PCL + PINS (**P**revious part**I**tion's **N**egative **S**amples) encourages a hypergraph encoder Enc_θ to learn dissimilarity between nodes at different partitions (see Figure 3). A pseudocode of PCL + PINS is provided in Algorithm 1. As Basic PCL does, PCL + PINS treats each partition as a mini-batch for contrastive learning. However, while Basic PCL only utilizes contrastive loss obtained within each partition, PCL + PINS incorporates information from other partitions.

In PCL + PINS, before moving to the next partition (after updating the encoder in the current partition), the encoder extracts node embeddings of two augmented views of the current partition $\mathcal{P}_k$ (denoted by $\mathbf{Z}_{k1}$ and $\mathbf{Z}_{k2}$) (line 12). For the next partition $\mathcal{P}_t$, PCL + PINS uses not only the CL loss in Eq (10), which is obtained within a partition $\mathcal{P}_t$, but also an inter-partition CL loss to maximize the dissimilarity between the node embeddings from a view of $\mathcal{P}_t$ and those from a view of $\mathcal{P}_k$. Specifically, the inter-partition CL loss between a view $\mathcal{P}_{t1}^{\emptyset}$ of $\mathcal{P}_t$ and a view $\mathcal{P}_{k1}^{\emptyset}$ of $\mathcal{P}_k$ is defined as:

$$\mathcal{L}_{I,t}(1,1) = \frac{1}{|\mathcal{P}_t^{\mathsf{V}}|} \sum_{i=1}^{j P_t^{\mathcal{V}} j} \log\left(\sum_{j \mathsf{2} P_{k,s}^{\mathcal{V}}} \exp\left(\frac{s(\mathbf{z}_{t1,i}, \mathbf{z}_{k1,j})}{\tau}\right)\right), \tag{11}$$

where $\mathcal{P}_{k,s}^{\mathsf{V}} \subset \mathcal{P}_k^{\mathsf{V}}$ are sampled nodes from $\mathcal{P}_k$ for negative samples. We consider all possible pairs of a view from $\mathcal{P}_t$ and a view from $\mathcal{P}_t$, and based on them we define $\mathcal{L}_{I,t}(1,1)$, $\mathcal{L}_{I,t}(1,2)$, $\mathcal{L}_{I,t}(2,1)$, and $\mathcal{L}_{I,t}(2,2)$, accordingly. Then, the final loss of PCL + PINS for each partition $\mathcal{P}_t$ is defined as:

$$\mathcal{L} = \mathcal{L}_{CL,t} + \lambda\left(\mathcal{L}_{I,t}(1,1) + \mathcal{L}_{I,t}(1,2) + \mathcal{L}_{I,t}(2,1) + \mathcal{L}_{I,t}(2,2)\right), \tag{12}$$

where $\mathcal{L}_{CL,t}$ is the loss within $\mathcal{P}_t$ in Eq (10), and λ is a hyperparameter that controls the strength of the inter-partition CL loss.

In summary, PCL + PINS uses node embeddings from the previous partitions as negative samples for the current partition. It is important to note that $\mathbf{Z}_{k1}$ and $\mathbf{Z}_{k2}$ are obtained from a partition already residing in GPU memory, and does not require any additional data loading process. In addition, $\mathbf{Z}_{k1}$ and $\mathbf{Z}_{k2}$ are not outdated since they are extracted after updating the encoder.

5.2.3 PCL+P-IOS

PCL+P-IOS (**P**artitions with the **I**nclusion of **O**utsider **S**et) restores each hyperedge by adding its constituent nodes missed during partitioning back to the hyperedge, to mitigate information loss caused by partitioning. An example of P-IOS is illustrated in Figure 4(b).

Formally, for a given set of hypergraph partitions $\boldsymbol{\mathcal{P}}$, PCL+P-IOS builds a new hypergraph partition set $\tilde{\boldsymbol{\mathcal{P}}}$, which is defined as:

$$\tilde{\boldsymbol{\mathcal{P}}} = \left\{ \tilde{\mathcal{P}}_k = \left(\tilde{\mathcal{P}}_k^{\mathsf{V}}, \tilde{\mathcal{P}}_k^{\mathsf{E}} \right) : k \in \{1, \cdots, K\} \right\}, \tag{13}$$

where $\tilde{\mathcal{P}}_k^{\mathsf{E}} = \{e \in \mathcal{E} : e \cap \mathcal{P}_k \neq \emptyset\}$, and $\tilde{\mathcal{P}}_k^{\mathsf{V}} = \bigcup_{e 2 \tilde{P}_k^{\mathcal{E}}} e$. The training strategy of PCL+P-IOS is the same as that of Basic PCL except that PCL+P-IOS uses $\tilde{\boldsymbol{\mathcal{P}}}$, as input hypergraph partitions, instead of $\boldsymbol{\mathcal{P}}$.

5.2.4 Classifier for Pair-level Downstream Tasks

Note that the proposed methods (Basic PCL, PCL+PINS, and PCL+P-IOS) are for obtaining representations of nodes. Lastly, we present how PCL makes predictions for node-pair-level tasks and hyperedge-pair-level tasks with acquired representations. We denote node representations obtained by a hypergraph encoder (trained by one of the proposed methods) by $\mathbf{P}_v \in \mathsf{R}^{j\mathsf{V}j \;\; F'}$. It should be noticed that the representations are the output of the hypergraph encoder Enc_θ not the projection head $Proj_\Theta$, which is used only for contrastive learning.

For a node pair-level task, PCL directly uses $\mathbf{P}_v$ as node features. To make a prediction for a pair of nodes $\{v_i, v_j\}$, PCL uses a classifier $h_v : (\mathsf{R}^{F'} \times \mathsf{R}^{F'}) \mapsto [0, 1]$. Specifically, h_v is structured as follows:

$$h_v(\boldsymbol{p}_{v,i}, \boldsymbol{p}_{v,j}) = \sigma(W_{12}(R(W_{11}\boldsymbol{p}_{v,i} + b_{11}) \otimes R(W_{11}\boldsymbol{p}_{v,j} + b_{11})) + b_{12}), \tag{14}$$

where $\otimes$ denotes the element-wise product; σ denotes the sigmoid function; R denotes the ReLU activation function [89]; W_{11} and W_{12} are learnable weight matrices; and b_{11} and b_{12} are learnable bias matrices. All parameters (i.e., W_{11}, W_{12}, b_{11}, and b_{12}) are updated using a classification loss. Note that in the proposed local clustering task, this prediction is used to identify whether the pair of nodes $\{v_i, v_j\}$ belong to the same cluster or not.

For a hyperedge pair-level task, PCL first creates hyperedge features by aggregating node representations that belong to the corresponding hyperedge. Formally, hyperedge features $\mathbf{P}_e \in \mathsf{R}^{j\mathsf{E}j \;\; F'}$ (or $\mathbf{P}_e \in \mathsf{R}^{j\mathsf{E}'j \;\; F'}$ in our hyperedge disambiguation task) are computed as follows:

$$\boldsymbol{p}_{e,i} = \sum_{v_k 2 e_i} p_{v,k}, \;\; \forall e_i \in \mathcal{E} \text{ (or } \forall e_i \in \mathcal{E}^{\emptyset} \text{ in hyperedge disambiguation)}. \tag{15}$$

Although we have adopted a summation for the aggregation function, other permutation invariant functions (e.g., average and maximum) can also be utilized. To make a prediction for a pair of hyperedges $\{e_i, e_j\}$, PCL uses a classifier $h_e : (\mathsf{R}^{F'} \times \mathsf{R}^{F'}) \mapsto [0, 1]$. Specifically, h_e is structured as follows:

$$h_e(\boldsymbol{p}_{e,i}, \boldsymbol{p}_{e,j}) = \sigma(W_{22}(R(W_{21}\boldsymbol{p}_{e,i} + b_{21}) \otimes R(W_{21}\boldsymbol{p}_{e,j} + b_{21})) + b_{22}), \tag{16}$$

where $\otimes$ denotes an element-wise product; σ denotes the sigmoid function; R denotes the ReLU activation function [89], W_{21} and W_{22} are learnable weight matrices; and b_{21} and b_{22} are learnable bias matrices. All parameters (W_{21}, W_{22}, b_{21}, and b_{22}) are updated using a classification loss. Note that in the proposed

hyperedge disambiguation task, this prediction is used to identify whether a given pair of hyperedges $\{e_i, e_j\}$ are in fact split from one hyperedge.

When training the classifiers h_v and h_e, we freeze the parameters of the hypergraph encoder Enc_θ, and thus the node representations $\mathbf{P}_v$ and hyperedge representations $\mathbf{P}_e$ are also frozen. However, it is also possible to fine-tune Enc_θ while training the h_v and h_e for downstream tasks.

5.3 Complexity Analysis

We examine the time complexity and memory requirement of PCL, focusing on a forward pass of Basic PCL. Note that obtaining node embeddings with a hypergraph neural network incurs time complexity of $\mathcal{O}(|\mathcal{V}| + \sum_{e_i \in E} |e_i|)$, as outlined in Eq (1) and (2).

We first present the time complexity of a forward pass of the HCL (hypergraph contrastive learning), a contrastive learning framework based on the entire hypergraph dataset. As stated above, the time complexity of obtaining node embeddings of two views using a hypergraph encoder is $O(|\mathcal{V}| + \sum_{e_i \in E} |e_i|)$. After this, node embeddings of two views are fed into a projection head, which takes $O(|\mathcal{V}|)$ time. At last, it computes the contrastive loss between each node and negative samples, whose time complexity is $O(|\mathcal{V}|N)$ where N is the number of negative samples. The overall time complexity of a forward pass of HCL becomes

$$\mathcal{O}\left((|\mathcal{V}| + \sum_{e_i \in E} |e_i|) + |\mathcal{V}| + |\mathcal{V}|N\right) = \mathcal{O}\left(|\mathcal{V}| + \sum_{e_i \in E} |e_i|\right). \tag{17}$$

A forward pass of HCL requires $\mathcal{O}(|\mathcal{V}| + \sum_{e_i \in E} |e_i|)$ (GPU) memory space for storing the entire input hypergraph and (intermediate) embeddings.

Now, we elaborate on the complexity of our proposed method, Basic PCL. The time complexity in Eq (17) is applied to each partition (instead of the entire hypergraph), and as a result, the overall time complexity of a forward pass of Basic PCL becomes

$$\mathcal{O}\left(\sum_{k=1}^{jKj} |\mathcal{P}_k^{\mathcal{V}}| + \sum_{k=1}^{jKj} \sum_{e_i' \in P_k^{\mathcal{E}}} |e_i^{0}|\right) = \mathcal{O}\left(|\mathcal{V}| + \sum_{e_i \in E} |e_i|\right), \tag{18}$$

which is equivalent to the time complexity of HCL.

The (GPU) memory requirement of Basic PCL differs from that of HCL. Since Basic PCL loads only one partition into (GPU) memory at a time, the amount of (GPU) memory required becomes

$$\mathcal{O}\left(\max_{P_k \in \mathcal{P}} \left(|\mathcal{P}_k^{\mathcal{V}}| + \sum_{e_i' \in P_k^{\mathcal{E}}} |e_i^{0}|\right)\right), \tag{19}$$

which indicates that the (GPU) memory requirement of Basic PCL is less than or equal to that of the HCL. This fact is also further supported by experiments, where HCL encounters out-of-memory issues when dealing with large-scale datasets, whereas PCL is not susceptible to such problems (see Table 3 and 4). Regarding speed, empirical observations indicate that HCL is faster in certain real-world datasets. This is because HCL requires less time to load datasets into (GPU) memory compared to PCL, which necessitates additional time for loading and unloading each partition into (GPU) memory.

6 Experiments

In this section, we review our experiments to answer the following questions:

- **RQ1:** How accurate is PCL for the proposed pair-level tasks?
- **RQ2:** How effective is PINS?
- **RQ3:** How much GPU memory is required by PCL with and without PINS?
- **RQ4:** How effective is P-IOS?
- **RQ5:** How does the performance of PCL (+ PINS) depend on the number (or size) of partitions?

We first describe the problem settings for the proposed pair-level prediction tasks. Then, we provide details of experimental settings and datasets. Lastly, we provide answers to the above questions based on our experimental results.

6.1 Problem Settings

In this subsection, we introduce how we compose train and test pairs for the two pair-level prediction tasks described in Section 3.

6.1.1 Hyperedge Disambiguation (Task-I)

Given an original hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, we split hyperedges in a disjoint way. Let $\mathcal{E}_S \subseteq \mathcal{E}$ be the subset of the original hyperedge set that consists of all hyperedges of sizes greater than 10 (i.e., $\mathcal{E}_S = \{e \in \mathcal{E} : |e| > 10\}$). We randomly split every hyperedge e_i in $\mathcal{E}_S$ into two sub-hyperedges $\{e_{i1}, e_{i2}\}$, under a constraint that the size of both sub-hyperedges should be at least 5 (i.e., $e_{i1} \cap e_{i2} = \emptyset, e_{i1} \cup e_{i2} = e_i,$ and $\min(|e_{i1}|, |e_{i2}|) \geq 5$). Then, we create a new hypergraph $\mathcal{H}^0$ whose hyperedge set $\mathcal{E}^0$ is defined as:

$$\mathcal{H}^0 = (\mathcal{V}, \mathcal{E}^0), \text{ where } \mathcal{E}^0 = (\mathcal{E} \setminus \mathcal{E}_S) \cup \bigcup_{e_i \in E_S} \{e_{i1}, e_{i2}\}. \tag{20}$$

We denote the set of ground-truth split hyperedge pairs as $G_S = \{\{e_{i1}, e_{i2}\} : e_i \in \mathcal{E}_S\}$. We use a subset G_S^0 of G_S as a train positive pair set, and the others (i.e., $G_S \setminus G_S^0$) as a test positive pair set.

For training and testing of classifiers of this task, we also create a negative-pair set. We constrain the size of hyperedges in negative pairs since all positive pairs consist of hyperedges of size at least 5. Specifically, we use the set $\mathcal{E}_5^0 = \{e \in \mathcal{E}^0 : |e| \geq 5\}$ of hyperedges whose size is at least 5. We compose a negative pair set with pairs of hyperedges in $\mathcal{E}_5^0$, ensuring that these negative pairs are not included in G_s. Specifically, the train negative pair set is the set of pairs of hyperedges $\{e_i, e_j\}$ that satisfy at least one of the two following conditions:

- **Condition 1:** $\left(e_i, e_j \in \bigcup_{g_s \in G'_S} g_s\right) \wedge (\{e_i, e_j\} \notin G_S^0)$,
- **Condition 2:** $\left(e_i \in \bigcup_{g_s \in G'_S} g_s\right) \wedge \left(e_j \notin \bigcup_{g_s \in G'_S} g_s\right)$.

The test negative pair set is the set of pairs $\{e_i, e_j\}$ of hyperedges from $\mathcal{E}_5^0$ that satisfy the following condition:

$$\left(e_i, e_j \notin \bigcup_{g_s \in G'_S} g_s\right) \wedge (\{e_i, e_j\} \notin G_s). \tag{21}$$

Since the number of possible negative pairs is vast, instead of considering them all, we sample some of them so that (1) the number of train negative pairs matches that of train positive pairs, and (2) the number of test negative pairs matches that of test positive pairs.

6.1.2 Local Clustering (Task-II)

In our setting, the clusters are formed in a disjoint way. From the entire set of clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_{|\mathcal{C}|}\}$, a subset $\mathcal{C}^0 \subset \mathcal{C}$ is given for training. The train positive pair set is defined as the set of node pairs that belong together to a **known** ground-truth cluster, formally,

$$\left\{\{v_i, v_j\} \in \binom{\mathcal{V}}{2} : v_i, v_j \in C^0, C^0 \in \mathcal{C}^0\right\}. \tag{22}$$

The test positive pair set is defined as the set of node pairs that belong together to an **unknown** ground-truth cluster, formally

$$\left\{\{v_i, v_j\} \in \binom{\mathcal{V}}{2} : v_i, v_j \in C^0, C^0 \in (\mathcal{C} \setminus \mathcal{C}^0)\right\}. \tag{23}$$

Instead of using all possible positive pairs within each cluster, we sample a certain portion of node pairs from each cluster.

We define the train node set as $V_{TR} = \bigcup_{C' \in \mathcal{C}'} C^0$ and the test node set as $V_{TE} = \bigcup_{C' \in (\mathcal{C} \setminus \mathcal{C}')} C^0$. Then, the train negative pair set is defined as the node pairs from V_{TR} that are not included in the train positive pair set. Similarly, we define the test negative pair set as the node pairs from V_{TE}, except for the pairs that belong to the test positive pairs. As in the previous task, instead of using all negative pairs, we sample some of them so that (1) the number of train negative pairs matches that of train positive pairs, and (2) the number of test negative pairs matches that of test positive pairs. Throughout the experiments, in each dataset, we use the labels of nodes as the ground-truth clusters of the nodes.

6.2 Experimental Settings

In this section, we provide an overview of the experimental setup for our study.

6.2.1 Baseline Methods

We compare the proposed methods against 16 baseline methods in the two proposed tasks (Task-I and Task-II), which include Multi-Layer Perceptron (MLP) [92], graph encoders trained via supervised learning or self-supervised learning[9], and hypergraph encoders trained via supervised learning and (full-batch) self-supervised learning. These baseline methods serve as a means to assess the effectiveness of several ideas behind the proposed methods for Task-I and Task-II: the ideas are (1) hypergraph data modeling (2) self-supervised learning strategy, and (3) partitioning.

For graph encoders trained via supervised learning, we use GCN [19] and GAT [21], and for self-supervised learning methods, we use GCN trained by BGRL [91] or GGD [93]. For hypergraph encoders trained via supervised learning, we use HGNN [23], UniGCNII [12], and AllSet [14]. All the above methods are trained and evaluated on both entire and partitioned (hyper)graphs. Lastly, we use HGNN trained via self-supervised learning on the entire hypergraph, which we call Hypergraph Contrastive Learning (HCL), as an additional baseline method.

[9]Since graph encoders require a graph topology as an input, we convert original hypergraphs into graphs by Clique Expansion, described in Appendix A.2.

6.2.2 Implementations

We implement our methods and HCL using PyTorch 1.11.0 [94] and PyTorch Geometric 2.0.4 [95], and for all other baseline methods, we use their official code.

6.2.3 Machine Specification

All experiments are conducted on a machine with NVIDIA RTX 8000 D6 GPUs (48GB memory) and two Intel Xeon Silver 4214R Processors.

6.2.4 Datasets

We conduct experiments on five real-world hypergraph datasets: two existing small-scale datasets (DBLP [96] and Trivago [97]), one medium-scale hypergraph dataset transformed from a heterogeneous graph (OGBN-MAG [98]), and the two proposed large-scale hypergraph datasets (AMiner and MAG). Some statistics of these datasets are reported in Table 2. Details regarding nodes, hyperedges, and labels of these datasets are described in Appendix A.1.

6.2.5 Experimental Set-ups

We use the average precision score (AP) and the area under the ROC curve score (AUROC) as two quantitative evaluation metrics since the proposed tasks are sort of binary classification tasks. For each dataset and for each task, we evaluate each model using 10 data splits. Specifically, we measure average AP and average AUROC together with their standard deviations over 10 splits.

For Task-I, we split the ground-truth hyperedge pairs into 10% and 90% and use them as the positive train pairs and positive test pairs, respectively. In addition, we use half of the positive train pairs for validation and use the remaining half for training. By following the procedures described in Section 6.1, we create the same number of negative train/validation/test pairs.

Similarly for Task-II, we split the ground-truth clusters (i.e., classes) into 10% and 90% and use them as known and unknown ground-truth clusters, respectively. Here also, we use half of the known clusters for validation and use the remaining half for training. Since the DBLP dataset has only six clusters, we use four of them (specifically, two for training and two for validation) as known ground-truth clusters, which results in a 66%/34% split for Task-II. Based on the clusters, by following the procedures described in Section 6.1, we create positive and negative train/validation/test pairs. Specifically, for each cluster, we randomly sample the following number of positive pairs from all possible positive pairs that can be created from the corresponding cluster (and use the same number of negative pairs) in each dataset: 10000 in DBLP and Trivago and 50000 in OGBN-MAG, AMiner, and MAG.

For all partition-based methods, the number of partitions (i.e., $|\mathcal{P}|$) is set to 4 for DBLP, 32 for Trivago, 128 for OGBN-MAG, and 256 for AMiner and MAG, unless otherwise stated. The effect of the number of partitions is also explored in Section 6.3.5. For all cases, we utilize the Adam optimizer [99] with a fixed weight decay scalar of 10^{-6}. The learning rate for each case is tuned as a hyperparameter. The specific hyperparameters for the proposed methods and all baseline methods are described in Appendix A.3. For PCL, unless otherwise stated, PINS is used for Task-II but not for Task-I, and P-IOS is not used for both tasks. This is because PINS does not lead to accuracy gains in most cases for Task-I, and P-IOS limits the scalability of PCL. PINS, however, is helpful for prediction accuracy, as shown in Section 6.3.4.

Table 3: **Task-I**: On the hyperedge disambiguation task, Basic PCL (proposed) is comparable to and often better than several baseline methods, including those requiring the full data to be loaded in memory for training. We report the average and standard deviation of each metric on 10 random data splits. The best performance in each setting is highlighted in **bold**, and the second best performance is highlighted in underline. OOM indicates "out of memory".

		DBLP		Trivago		OGBN-MAG		AMiner		MAG		Avg
Data Type	Methods	AP	AUROC	AP	AUROC	AP	AUROC	AP	AUROC	AP	AUROC	Rank
Only X	MLP	65.2 ± 3.4	62.2 ± 3.4	60.2 ± 1.6	60.7 ± 2.5	73.1 ± 2.6	71.9 ± 2.8	91.7 ± 0.3	92.9 ± 0.3	91.2 ± 0.6	92.6 ± 0.6	9.8
Full Graph	GCN	83.6 ± 1.2	84.6 ± 1.5	61.6 ± 0.8	58.1 ± 2.4	80.1 ± 1.7	79.8 ± 2.0	OOM	OOM	OOM	OOM	9.7
	GAT	83.5 ± 1.0	84.6 ± 1.0	61.6 ± 0.5	60.0 ± 2.7	76.5 ± 2.3	75.7 ± 3.0	OOM	OOM	OOM	OOM	10.3
	BGRL	84.5 ± 1.2	84.3 ± 1.7	72.0 ± 1.8	72.2 ± 2.2	83.1 ± 0.6	83.9 ± 0.5	OOM	OOM	OOM	OOM	8
	GGD	80.9 ± 1.6	80.6 ± 2.1	71.8 ± 0.9	72.5 ± 1.6	76.1 ± 1.1	76.1 ± 1.3	OOM	OOM	OOM	OOM	9.7
Full Hypergraph	HGNN	77.3 ± 2.5	80.0 ± 3.1	72.1 ± 5.1	76.6 ± 1.4	91.2 ± 0.7	92.9 ± 0.6	OOM	OOM	OOM	OOM	7.7
	UniGCNII	78.1 ± 3.4	77.1 ± 3.9	71.8 ± 1.5	75.6 ± 2.3	86.9 ± 5.6	88.3 ± 6.7	OOM	OOM	OOM	OOM	8.7
	ALLSET	53.1 ± 1.2	55.8 ± 2.2	53.7 ± 1.2	56.8 ± 2.2	65.3 ± 1.4	73.4 ± 1.6	OOM	OOM	OOM	OOM	13.1
	HCL	**91.0 ± 1.7**	**92.6 ± 2.1**	73.7 ± 0.7	78.8 ± 1.1	**94.2 ± 0.4**	**95.5 ± 0.3**	OOM	OOM	OOM	OOM	4.5
Partitioned Graph	GCN	84.5 ± 1.6	85.9 ± 1.6	65.9 ± 1.9	66.7 ± 2.0	82.3 ± 2.6	83.4 ± 2.8	81.2 ± 0.6	81.5 ± 0.6	OOM	OOM	7.9
	GAT	84.7 ± 1.3	86.4 ± 1.2	62.2 ± 3.1	62.5 ± 4.6	78.0 ± 1.8	79.6 ± 1.8	81.3 ± 0.5	81.0 ± 0.6	OOM	OOM	8.2
	BGRL	85.8 ± 1.7	85.6 ± 1.7	82.4 ± 2.2	83.5 ± 2.1	91.4 ± 0.5	92.8 ± 0.5	90.1 ± 0.8	90.9 ± 0.8	89.7 ± 0.7	90.8 ± 0.9	3.9
	GGD	80.1 ± 2.4	80.8 ± 2.3	76.9 ± 1.6	78.3 ± 1.5	88.7 ± 1.0	90.2 ± 0.9	82.5 ± 1.2	83.6 ± 1.5	80.1 ± 1.3	81.2 ± 1.4	6.2
Partitioned Hypergraph	HGNN	84.1 ± 2.0	86.2 ± 1.8	71.8 ± 0.8	75.8 ± 1.0	85.9 ± 1.1	88.1 ± 1.0	91.3 ± 0.3	92.5 ± 0.3	89.2 ± 0.7	91.6 ± 0.5	5.4
	UniGCNII	79.9 ± 1.8	80.0 ± 1.8	71.5 ± 2.6	74.2 ± 3.5	77.4 ± 0.9	75.7 ± 1.2	91.8 ± 0.5	92.1 ± 0.5	90.4 ± 0.4	91.9 ± 0.1	7.9
	ALLSET	54.2 ± 1.6	57.6 ± 2.7	53.6 ± 1.2	56.7 ± 2.0	59.3 ± 1.6	65.6 ± 2.4	61.5 ± 1.8	68.6 ± 2.5	OOM	OOM	13.2
Partitioned Hypergraph	**Basic PCL (proposed)**	87.1 ± 1.9	88.3 ± 2.0	**88.2 ± 0.9**	**88.9 ± 0.7**	94.1 ± 0.4	95.1 ± 0.3	**95.5 ± 0.7**	**96.0 ± 0.8**	**96.2 ± 0.3**	**96.9 ± 0.3**	**1.4**

6.3 Experimental Results

6.3.1 RQ1. Overall Performance on Downstream Tasks

As shown in Table 3 and 4, PCL achieves overall the best AP and AUROC scores among all the considered methods (i.e., best in terms of average rank) on both tasks. There are two notable observations, which we describe below.

First, in the two proposed large-scale hypergraphs AMiner and MAG, PCL shows the best performance with a significant gap from the second best model (spec., 3.7% higher AP on Task-I and 9.7% higher AP on Task-II in AMiner dataset). This result demonstrates the effectiveness of PCL in pair-level tasks on large-scale hypergraph datasets. Note that the graph representation methods operating on clique-expanded graphs show poor performance on both tasks in large-scale datasets. This result highlights the effectiveness of the hypergraph modeling of large-scale group interactions when tackling pair-level tasks.

Second, although PCL is trained on partitioned hypergraphs, surprisingly, it shows performance comparable to or even better than that of HCL, which uses entire hypergraphs, without partitioning, for contrastive learning. That is, the topological information loss due to partitioning is not severe enough to harm the overall performance of models, and sometimes it is even helpful. We suspect that partitioning may increase the *hardness* of negative samples that are used during contrastive learning, which may lead to performance improvement. Specifically, when selecting negative samples for contrastive learning, there is inherent randomness in the choice of which negative samples. If we choose negative samples from the entire hypergraph (as in HCL), it is likely that distant nodes are chosen. In this case, representations of different nodes can easily be dissimilar, since it is likely that distant nodes have different neighbors. However, if we select negative samples within a partition (as in PCL), it is relatively more likely that nodes sharing many neighbors are chosen[10], and this makes the encoder hard to maximize the dissimilarity between representations of such

[10]This is because partitioning algorithms generally assign such nodes in the same partition.

Table 4: **Task-II**: On the local clustering task, PCL+PINS (proposed) is comparable to and often better than several baseline methods, including those requiring the full data to be loaded in memory for training. We report the average and standard deviation of each metric on 10 random data splits. The best performance in each setting is highlighted in **bold**, and the second best performance is highlighted in underline. OOM indicates "out of memory".

Data Type	Methods	DBLP AP	DBLP AUROC	Trivago AP	Trivago AUROC	OGBN-MAG AP	OGBN-MAG AUROC	AMiner AP	AMiner AUROC	MAG AP	MAG AUROC	Avg. Rank
Only X	MLP	52.2 ± 2.3	51.3 ± 3.5	50.6 ± 0.4	50.9 ± 0.7	72.1 ± 1.0	73.9 ± 0.8	70.5 ± 0.9	72.9 ± 0.8	76.2 ± 1.3	79.4 ± 0.9	7.1
Full Graph	GCN	54.6 ± 4.0	54.1 ± 4.7	50.1 ± 0.2	50.2 ± 0.1	53.5 ± 0.3	53.8 ± 0.4	OOM	OOM	OOM	OOM	11.1
	GAT	55.1 ± 4.2	55.3 ± 5.9	50.1 ± 0.0	50.0 ± 0.0	OOM	OOM	OOM	OOM	OOM	OOM	11.6
	BGRL	55.1 ± 5.8	52.9 ± 3.1	50.3 ± 0.2	50.4 ± 0.2	53.2 ± 0.3	53.3 ± 0.3	OOM	OOM	OOM	OOM	10.9
	GGD	58.6 ± 9.5	56.0 ± 8.6	50.3 ± 0.2	50.3 ± 0.2	53.0 ± 0.3	53.1 ± 0.3	OOM	OOM	OOM	OOM	10.1
Full Hypergraph	HGNN	56.9 ± 5.3	54.9 ± 6.3	54.8 ± 3.7	54.8 ± 3.6	78.2 ± 1.0	80.1 ± 0.7	OOM	OOM	OOM	OOM	6.3
	UniGCNII	55.2 ± 4.6	52.9 ± 4.5	51.6 ± 1.0	51.1 ± 0.9	76.7 ± 1.2	79.0 ± 1.1	OOM	OOM	OOM	OOM	7.8
	ALLSET	54.2 ± 4.1	56.1 ± 5.4	51.0 ± 0.7	51.7 ± 0.7	60.0 ± 2.2	61.8 ± 2.5	OOM	OOM	OOM	OOM	8.6
	HCL	63.6 ± 6.9	61.4 ± 7.9	58.6 ± 2.6	58.8 ± 4.2	**78.5 ± 1.0**	**80.9 ± 0.7**	OOM	OOM	OOM	OOM	4.8
Partitioned Graph	GCN	51.9 ± 0.4	52.2 ± 0.4	50.1 ± 0.0	50.1 ± 0.0	51.6 ± 0.4	51.7 ± 0.4	54.9 ± 0.7	54.3 ± 0.7	OOM	OOM	12.7
	GAT	52.4 ± 0.6	53.1 ± 0.6	50.4 ± 0.7	50.5 ± 1.0	54.1 ± 0.7	54.3 ± 0.8	55.2 ± 0.4	54.6 ± 0.4	OOM	OOM	10.3
	BGRL	58.8 ± 7.7	56.8 ± 5.6	**61.2 ± 3.9**	**61.4 ± 3.9**	65.6 ± 1.0	66.4 ± 0.8	65.8 ± 0.5	67.1 ± 0.6	71.0 ± 1.0	72.7 ± 0.9	3.6
	GGD	**67.7 ± 12.7**	**67.2 ± 13.3**	59.4 ± 3.8	59.4 ± 3.5	64.8 ± 1.0	65.5 ± 0.6	62.8 ± 0.8	64.3 ± 0.8	68.5 ± 0.9	70.5 ± 0.9	3.8
Partitioned Hypergraph	HGNN	55.2 ± 6.5	55.2 ± 7.3	52.0 ± 1.0	52.4 ± 2.0	68.8 ± 1.3	69.9 ± 1.6	57.6 ± 1.6	57.5 ± 2.1	62.8 ± 3.0	62.5 ± 3.6	6.6
	UniGCNII	52.6 ± 1.7	52.3 ± 0.9	50.4 ± 0.3	50.1 ± 0.2	54.9 ± 0.6	54.6 ± 0.5	59.9 ± 1.2	59.6 ± 1.3	65.9 ± 2.2	66.0 ± 2.2	9.8
	ALLSET	53.2 ± 1.6	54.6 ± 2.1	51.5 ± 0.3	52.5 ± 0.5	57.5 ± 1.9	59.2 ± 2.8	58.1 ± 1.1	60.9 ± 1.4	OOM	OOM	8.2
Partitioned Hypergraph	PCL+PINS **(proposed)**	63.2 ± 10.6	64.0 ± 11.5	58.6 ± 1.0	59.2 ± 1.3	77.6 ± 0.8	79.8 ± 0.6	**80.2 ± 1.7**	**81.6 ± 0.8**	**83.1 ± 1.4**	**86.1 ± 1.5**	**2.1**

Table 5: Effectiveness of PINS on Task-II. PCL+PINS outperforms PCL w/o PINS in most datasets.

Data	Metric	PCL w/o PINS	PCL+PINS
DBLP	AP	62.4 ± 11.5	63.2 ± 10.6
	AUROC	61.1 ± 9.9	64.0 ± 11.5
Trivago	AP	58.6 ± 1.0	58.6 ± 1.0
	AUROC	58.7 ± 1.0	59.2 ± 1.3
OGBN-MAG	AP	77.3 ± 0.7	77.6 ± 0.8
	AUROC	79.6 ± 0.3	79.8 ± 0.6
AMiner	AP	78.6 ± 1.3	80.2 ± 1.7
	AUROC	81.3 ± 1.2	81.6 ± 0.8
MAG	AP	83.3 ± 1.1	83.1 ± 1.4
	AUROC	86.3 ± 0.7	86.1 ± 1.5

nodes. As a result, the hardness of negative samples increases in PCL, and learning to distinguish hard negative samples from positive samples potentially enhances the quality of the trained encoder's output representation [100, 55].

6.3.2 RQ2. Effectiveness of PINS

We demonstrate the effectiveness of PINS on Task-II by comparing the performances of PCL w/o PINS (Basic PCL) and PCL+PINS. As shown in Table 5, PCL+PINS outperforms PCL w/o PINS in four out of five datasets. The performance gain by PINS is up to +2.9% (AUROC on DBLP), while the performance degradation is up to -0.2% (AUROC on MAG). Thus, we conclude that PINS is effective on Task-II. However, in our preliminary study, PINS does not increase the performance on Task-I.

6.3.3 RQ3. Efficiency of PINS

Despite the effectiveness of PINS on Task-II, one may concern with additional computational and memory costs caused by PINS. Regarding the concern, we compare the average running time per epoch of

Table 6: Effectiveness of P-IOS. PCL+P-IOS outperforms PCL w/o P-IOS in all cases.

Task	Data	Metric	PCL w/o P-IOS	PCL+P-IOS
Task-I	DBLP	AP	87.1 ± 1.9	90.8 ± 1.8
		AUROC	88.3 ± 2.0	92.9 ± 1.2
	Trivago	AP	88.2 ± 0.9	89.4 ± 1.2
		AUROC	88.9 ± 0.7	89.5 ± 0.9
	OGBN-MAG	AP	94.1 ± 0.4	94.8 ± 0.4
		AUROC	95.1 ± 0.2	96.2 ± 0.2
Task-II	DBLP	AP	62.4 ± 11.5	64.7 ± 11.5
		AUROC	61.1 ± 9.9	62.3 ± 12.3
	Trivago	AP	58.6 ± 1.0	59.2 ± 1.0
		AUROC	58.7 ± 1.0	59.3 ± 1.0
	OGBN-MAG	AP	77.3 ± 0.7	78.8 ± 0.9
		AUROC	79.6 ± 0.3	80.9 ± 0.7

Table 7: Cost of PINS. The additional cost due to PINS is not large in terms of time and marginal in terms of memory requirements.

	Method	OGBN-MAG	AMiner	MAG
Average Running Time Per Epoch (Sec)	PCL w/o PINS	20.102	89.505	103.450
	PCL+PINS	24.025	118.338	133.739
Average GPU Memory Usage (GB)	PCL w/o PINS	2.344	10.573	23.942
	PCL+PINS	2.347	10.575	23.945

PCL+PINS and PCL w/o PINS (Basic PCL), and we also measure their average GPU memory usage during the contrastive learning process[11].

As reported in Table 7, the average running time of PCL+PINS is 20%, 32%, and 23% longer than that of PCL w/o PINS on OGBN-MAG, AMiner, and MAG, respectively[12]. Moreover, it uses 0.1%, 0.01%, and 0.01% more average GPU memory, compared to PCL w/o PINS, for the above three datasets. Therefore, the additional cost due to PINS is not substantial.

6.3.4 RQ4. Effectiveness of P-IOS

To demonstrate the effectiveness of P-IOS, we compare PCL w/o P-IOS (Basic PCL) with PCL+P-IOS on both Task-I and Task-II in the DBLP, Trivago, and OGBN-MAG datasets. As shown in Table 6, the prediction performance is improved with P-IOS in all the settings. This result is intuitive since P-IOS mitigates information loss caused by partitioning via restoring each hyperedge. However, PCL+P-IOS is not applicable to large-scale hypergraphs since with P-IOS, resulting partitions are too large, causing an out-of-memory error.

6.3.5 RQ5. Tendency of PCL+PINS with Respect to the Number (or Size) of Partitions

At last, we investigate how the performance of PCL+PINS changes with respect to the number (or size) of partitions on both tasks. Note that the number of partitions and the size of each partition (i.e., number of

[11] At each mini-batch (partition) of contrastive learning, we record the GPU memory usage after completing the gradient computation (spec., execute `loss.backward()` and check the current GPU memory allocation using `torch.cuda.memory_allocated(device)`). After training an encoder in every mini-batch, we calculate the average GPU memory usage for the current epoch by averaging the usage across all partitions. Finally, we compute the average GPU memory usage across all epochs.

[12] The total contrastive training epochs are 50.

Table 8: Detailed statistics regarding partitions. $|\mathcal{P}|$ indicates the number of partitions, $\overline{|\mathcal{P}_i^V|}$ indicates the average number of nodes in each partition, $\overline{|\mathcal{P}_i^E|}$ indicates the average number of hyperedges in each partition, and $sd(|\mathcal{P}_i^V|)$ and $sd(|\mathcal{P}_i^E|)$ indicate the standard deviations of the numbers of nodes and hyperedges, respectively, in each partition.

	OGBN-MAG			AMiner			MAG		
	Large	Medium	Small	Large	Medium	Small	Large	Medium	Small
$\lvert\mathcal{P}\rvert$	32	64	128	256	512	1024	256	512	1024
$\overline{\lvert\mathcal{P}_i^V\rvert}$	23012.2	11506.1	5753.0	51806.9	25903.5	12951.7	106720.2	53360.1	26680.1
$\overline{\lvert\mathcal{P}_i^E\rvert}$	45454.3	24265.5	12659.5	122358.5	62491.5	31772.9	205835.5	108065.1	56134.2
$sd(\lvert\mathcal{P}_i^V\rvert)$	72.8	25.7	20.1	190.9	78.3	41.0	334.4	174.3	51.6
$sd(\lvert\mathcal{P}_i^E\rvert)$	15408.3	7765.6	5984.1	45433.8	25306.5	11955.4	37684.6	21684.5	12252.0

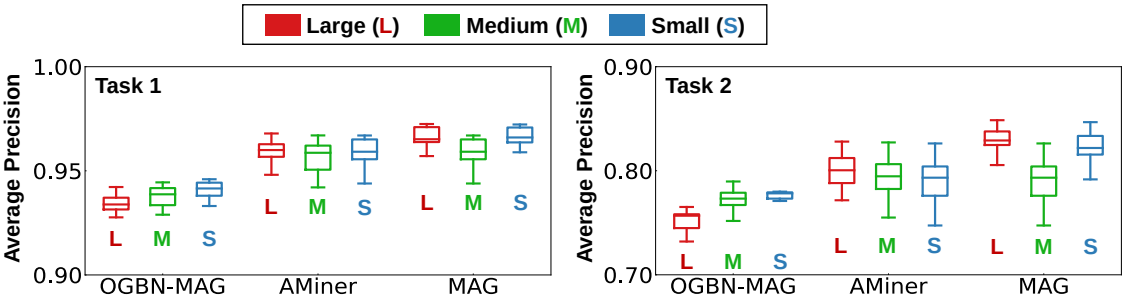


Figure 5: Performance tendency of PCL+PINS with respect to the size of partitions. The boxplot describes the test AP score distribution over 10 data splits. For each dataset, boxplots are arranged in order of Large, Medium, and Small sizes. No clear tendency exists between the number/size of the partition and the performance of PCL+PINS.

nodes and number of hyperedges) is in inverse proportion (see Table 8). To this end, for each partition size, we show the AP score distribution of PCL+PINS over 10 data splits of each dataset. As shown in Figure 5, there is no clear tendency between the number of partitions (or the size of partitions) and the performance of PCL+PINS, and especially in AMiner, there is no clear difference in the distributions depending on the size of partitions.

7 Conclusion

In this work, in order to bridge the gap between previous studies and real-world applications of hypergraph learning, we make three contributions that are summarized as follows:

- In terms of tasks, we provide two novel pair-level hypergraph-learning tasks (hyperedge disambiguation and local clustering) that can be used for formulating various real-world problems.
- In terms of datasets, we propose two large-scale hypergraph datasets (AMiner and MAG) that enable the evaluation of hypergraph neural networks at scale.
- In terms of training methods, we suggest PCL, a scalable hypergraph learning method. PCL is based on hypergraph partitioning and contrastive learning, equipped with two additional techniques (PINS and P-IOS) for reducing information loss caused by partitioning. We experimentally verify the superiority of PCL over 16 baseline methods on the proposed pairwise prediction tasks and the effectiveness of PCL+PINS and PCL+P-IOS.

For reproducibility, we make the source code and datasets used in the paper available at `https://github.com/kswoo97/pcl`.

Acknowledgements

This work was supported by Samsung Electronics Co., Ltd., National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1C1C1008296), and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00157, Robust, Fair, Extensible Data-Centric Continual Learning) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)).

References

[1] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48): E11221–E11230, 2018. doi: 10.1073/pnas.1800683115.

[2] Geon Lee, Minyoung Choe, and Kijung Shin. How do hyperedges overlap in real-world hypergraphs?-patterns, measures, and generators. In *Proceedings of the Web Conference 2021 (WWW)*, pages 3396–3407, 2021. doi: 10.1145/3442381.3450010.

[3] Jihoon Ko, Yunbum Kook, and Kijung Shin. Growth patterns and models of real-world hypergraphs. *Knowledge and Information Systems*, 64(11):2883–2920, 2022. doi: 10.1007/s10115-022-01739-9.

[4] Qi Luo, Dongxiao Yu, Zhipeng Cai, Xuemin Lin, and Xiuzhen Cheng. Hypercore maintenance in dynamic hypergraphs. In *IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2051–2056, 2021. doi: 10.1109/ICDE51399.2021.00199.

[5] Leo Torres, Ann S Blevins, Danielle Bassett, and Tina Eliassi-Rad. The why, how, and when of representations for complex systems. *SIAM Review*, 63(3):435–485, 2021. doi: 10.1137/20M1355896.

[6] Nan Yin, Fuli Feng, Zhigang Luo, Xiang Zhang, Wenjie Wang, Xiao Luo, Chong Chen, and Xian-Sheng Hua. Dynamic hypergraph convolutional network. In *IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1621–1634, 2022. doi: 10.1109/ICDE53745.2022.00167.

[7] Zhonghang Li, Chao Huang, Lianghao Xia, Yong Xu, and Jian Pei. Spatial-temporal hypergraph self-supervised learning for crime prediction. In *IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2984–2996, 2022. doi: 10.1109/ICDE53745.2022.00269.

[8] Sunwoo Kim, Minyoung Choe, Jaemin Yoo, and Kijung Shin. Reciprocity in directed hypergraphs: Measures, findings, and generators. In *IEEE International Conference on Data Mining (ICDM)*, 2022. doi: 10.1109/ICDM54844.2022.00122.

[9] Elena V Konstantinova and Vladimir A Skorobogatov. Application of hypergraph theory in chemistry. *Discrete Mathematics*, 235(1-3):365–383, 2001. doi: 10.1016/S0012-365X(00)00290-9.

[10] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5):e1000385, 2009. doi: 10.1371/journal.pcbi.1000385.

[11] Chao Qu, Ming Tao, and Ruifen Yuan. A hypergraph-based blockchain model and application in internet of things-enabled smart homes. *Sensors*, 18(9):2784, 2018. doi: 10.3390/s18092784.

[12] Jing Huang and Jie Yang. UniGNN: A unified framework for graph and hypergraph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2563–2569, 2021. doi: 10.24963/ijcai.2021/353.

[13] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. HyperGCN: A new method of training graph convolutional networks on hypergraphs. In *Advances in neural information processing systems (NeurIPS)*, volume 32, pages 1509–1520, 2019. doi: 10.48550/arXiv.1809.02589.

[14] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are AllSet: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations (ICLR)*, 2021. doi: 10.48550/arXiv.2106.13264.

[15] Dongjin Lee and Kijung Shin. I'm me, we're us, and i'm us: Tri-directional contrastive learning on hypergraphs. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, 2023. doi: 10.48550/arXiv.2206.04739.

[16] Naganand Yadati, Vikram Nitin, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. NHP: Neural hypergraph link prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 1705–1714, 2020. doi: 10.1145/3340531.3411870.

[17] Hyunjin Hwang, Seungwoo Lee, Chanyoung Park, and Kijung Shin. AHP: Learning to negative sample for hyperedge prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 2237–2242, 2022. doi: 10.1145/3477495.3531836.

[18] R Zhang, Y Zou, and J Ma. Hyper-SAGNN: A self-attention based graph neural network for hypergraphs. In *International Conference on Learning Representations (ICLR)*, 2020. doi: 10.48550/arXiv.1911.02613.

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. doi: 10.48550/arXiv.1609.02907.

[20] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems (NeurIPS)*, volume 30, 2017. doi: 10.48550/arXiv.1706.02216.

[21] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, 2018. doi: 10.48550/arXiv.1809.10341.

[22] Yihe Dong, Will Sawin, and Yoshua Bengio. HNHN: Hypergraph networks with hyperedge neurons. arXiv:2006.12278, 2020.

[23] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 33, pages 3558–3565, 2019. doi: 10.1609/aaai.v33i01.33013558.

[24] Steven Jecmen, Minji Yoon, Vincent Conitzer, Nihar B Shah, and Fei Fang. A dataset on malicious paper bidding in peer review. In *Proceedings of the ACM Web Conference 2023 (WWW)*, pages 3816–3826, 2023. doi: 10.1145/3543507.3583424.

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

[26] Si Zhang and Hanghang Tong. FINAL: Fast attributed network alignment. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data (KDD)*, pages 1345–1354, 2016. doi: 10.1145/2939672.2939766.

[27] Michael Tynes, Wenhao Gao, Daniel J Burrill, Enrique R Batista, Danny Perez, Ping Yang, and Nicholas Lubbers. Pairwise difference regression: A machine learning meta-algorithm for improved prediction and uncertainty quantification in chemical search. *Journal of Chemical Information and Modeling*, 61(8):3846–3857, 2021. doi: 10.1021/acs.jcim.1c00670.

[28] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, et al. OAG: Toward linking large-scale heterogeneous entity graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 2585–2595, 2019. doi: 10.1145/3292500.3330785.

[29] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *Proceedings of the fourth ACM international conference on Web search and data mining (WSDM)*, pages 177–186, 2011. doi: 10.1145/1935826.1935863.

[30] Song Bai, Feihu Zhang, and Philip HS Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021. doi: 10.1016/j.patcog.2020.107637.

[31] Devanshu Arya, Deepak K Gupta, Stevan Rudinac, and Marcel Worring. HyperSAGE: Generalizing inductive representation learning on hypergraphs. arXiv:2010.04558, 2020.

[32] Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. The total variation on hypergraphs-learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 26, 2013. doi: 10.48550/arXiv.1312.5179.

[33] Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering. In *International Conference on Machine Learning (ICML)*, pages 3014–3023, 2018. doi: 10.48550/arXiv.1803.03833.

[34] Jiying Zhang, Fuyang Li, Xi Xiao, Tingyang Xu, Yu Rong, Junzhou Huang, and Yatao Bian. Hypergraph convolutional networks via equivalency between hypergraphs and undirected graphs. arXiv:2203.16939, 2022.

[35] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems (NeurIPS)*, volume 30, 2017. doi: 10.48550/arXiv.1703.06114.

[36] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning (ICML)*, pages 3744–3753, 2019. doi: 10.48550/arXiv.1810.00825.

[37] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. AGL: A scalable system for industrial-purpose graph machine learning. *Proceedings of the VLDB Endowment (PVLDB)*, 13(12):3125–3137, 2020. doi: 10.14778/3415478.3415539.

[38] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. DistDGL: Distributed graph neural network training for billion-scale graphs. In *IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44, 2020. doi: 10.1109/IA351965.2020.00011.

[39] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. ByteGNN: Efficient graph neural network training at large scale. *Proceedings of the VLDB Endowment (PVLDB)*, 15(6):1228–1242, 2022. doi: 10.14778/3514061.3514069.

[40] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, 2018. doi: 10.48550/arXiv.1801.10247.

[41] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning (ICML)*, pages 942–950, 2018. doi: 10.48550/arXiv.1710.10568.

[42] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems (NeurIPS)*, volume 31, 2018. doi: 10.48550/arXiv.1809.05343.

[43] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*, pages 257–266, 2019. doi: 10.1145/3292500.3330925.

[44] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2019. doi: 10.48550/arXiv.1907.04931.

[45] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning (ICML)*, pages 6861–6871, 2019. doi: 10.48550/arXiv.1902.07153.

[46] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. SIGN: Scalable inception graph neural networks. arXiv:2004.11198, 2020.

[47] Sepideh Maleki, Donya Saless, Dennis P Wall, and Keshav Pingali. HyperNetVec: Fast and scalable hierarchical embedding for hypergraphs. In *Network Science (NetSci)*, pages 169–183. Springer, 2022. doi: 10.1007/978-3-030-97240-0_13.

[48] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 855–864, 2016. doi: 10.1145/2939672.2939754.

[49] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning (ICML)*, pages 1597–1607, 2020. doi: 10.48550/arXiv.2002.05709.

[50] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 9729–9738, 2020. doi: 10.48550/arXiv.1911.05722.

[51] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6894–6910, 2021. doi: 10.18653/v1/2021.emnlp-main.552.

[52] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in neural information processing systems (NeurIPS)*, volume 33, pages 5812–5823, 2020. doi: 10.48550/arXiv.2010.13902.

[53] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. arXiv:2006.04131, 2020.

[54] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021 (WWW)*, pages 2069–2080, 2021. doi: 10.1145/3442381.3449802.

[55] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International conference on machine learning (ICML)*, pages 4116–4126, 2020. doi: 10.48550/arXiv.2006.05582.

[56] Xiao Luo, Wei Ju, Meng Qu, Chong Chen, Minghua Deng, Xian-Sheng Hua, and Ming Zhang. DualGraph: Improving semi-supervised graph classification via dual contrastive learning. In *IEEE 38th International Conference on Data Engineering (ICDE)*, pages 699–712, 2022. doi: 10.1109/ICDE53745.2022.00057.

[57] Junwei Zhang, Min Gao, Junliang Yu, Lei Guo, Jundong Li, and Hongzhi Yin. Double-scale self-supervised hypergraph learning for group recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 2557–2567, 2021. doi: 10.1145/3459637.3482426.

[58] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. Self-supervised hypergraph convolutional networks for session-based recommendation. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 35, pages 4503–4511, 2021. doi: 10.1609/aaai.v35i5.16578.

[59] Junliang Yu, Hongzhi Yin, Jundong Li, Qinyong Wang, Nguyen Quoc Viet Hung, and Xiangliang Zhang. Self-supervised multi-channel hypergraph convolutional network for social recommendation. In *Proceedings of the web conference 2021 (WWW)*, pages 413–424, 2021. doi: 10.1145/3442381.3449844.

[60] Zhuang Liu, Yunpu Ma, Yuanxin Ouyang, and Zhang Xiong. Contrastive learning for recommender system. arXiv:2101.01317, 2021.

[61] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. Contrastive learning for sequential recommendation. In *IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1259–1273, 2022. doi: 10.1109/ICDE53745.2022.00099.

[62] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002. doi: 10.1073/pnas.122653799.

[63] Ümit V Çatalyürek and Cevdet Aykanat. PaToH (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer, 2011. doi: 10.1007/978-0-387-09766-4_93.

[64] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. arXiv:2006.16904, 2020.

[65] Imtiaz Ahmed, Travis Galoppo, Xia Hu, and Yu Ding. Graph regularized autoencoder and its application in unsupervised anomaly detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(8):4110–4124, 2021. doi: 10.1109/TPAMI.2021.3066111.

[66] Gabriele Grunig, Nedim Durmus, Yian Zhang, Yuting Lu, Sultan Pehlivan, Yuyan Wang, Kathleen Doo, Maria L Cotrina-Vidal, Roberta Goldring, Kenneth I Berger, et al. Molecular clustering analysis of blood biomarkers in world trade center exposed community members with persistent lower respiratory symptoms. *International journal of environmental research and public health*, 19(13):8102, 2022. doi: 10.3390/ijerph19138102.

[67] Zhongdao Wang, Liang Zheng, Yali Li, and Shengjin Wang. Linkage based face clustering via graph convolution network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 1117–1125, 2019. doi: 10.1109/CVPR.2019.00121.

[68] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th annual Design Automation Conference (DAC)*, pages 526–529, 1997. doi: 10.1145/266021.266273.

[69] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. High-quality hypergraph partitioning. *ACM Journal of Experimental Algorithmics*, 27:1–39, 2023. doi: 10.1145/3529090.

[70] Andrew E Caldwell, Andrew B Kahng, and Igor L Markov. Improved algorithms for hypergraph bipartitioning. In *Proceedings of the 2000 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 661–666, 2000. doi: 10.1109/ASPDAC.2000.835182.

[71] Christian Mayer, Ruben Mayer, Sukanya Bhowmik, Lukas Epple, and Kurt Rothermel. HYPE: Massive hypergraph partitioning with neighborhood expansion. In *IEEE International Conference on Big Data (Big Data)*, pages 458–467, 2018. doi: 10.1109/BigData.2018.8621968.

[72] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys*, 45(4):1–35, 2013. doi: 10.1145/2501654.2501657.

[73] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM computing surveys*, 51(2):1–37, 2018. doi: 10.1145/3172867.

[74] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining (WSDM)*, pages 587–596, 2013. doi: 10.1145/2433396.2433471.

[75] Oleksandr Shchur and Stephan Günnemann. Overlapping community detection with graph neural networks. arXiv:1909.12201, 2019.

[76] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. Dynamic structural clustering on graphs. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pages 1491–1503, 2021. doi: 10.1145/3448016.3452828.

[77] Md Nurul Muttakin, Md Iqbal Hossain, and Md Saidur Rahman. Overlapping community detection using dynamic dilated aggregation in deep residual GCN. arXiv:2210.11174, 2022.

[78] Martina Contisciani, Federico Battiston, and Caterina De Bacco. Inference of hyperedges and overlapping communities in hypergraphs. *Nature Communications*, 13(1):7229, 2022. doi: 10.1038/s41467-022-34714-7.

[79] Staša Milojević. Accuracy of simple, initials-based methods for author name disambiguation. *Journal of Informetrics*, 7(4):767–773, 2013. doi: 10.1016/j.joi.2013.06.006.

[80] Emiel Caron and Nees Jan van Eck. Large scale author name disambiguation using rule-based scoring and clustering. In *Proceedings of the 19th international conference on science and technology indicators (STI)*, pages 79–86, 2014. doi: 10.1007/978-981-32-9298-7_12.

[81] Debarshi Kumar Sanyal, Plaban Kumar Bhowmick, and Partha Pratim Das. A review of author name disambiguation techniques for the PubMed bibliographic database. *Journal of Information Science*, 47(2):227–254, 2021. doi: 10.1177/0165551519888605.

[82] Kaikai Deng, Ling Xing, Longshui Zheng, Honghai Wu, Ping Xie, and Feifei Gao. A user identification algorithm based on user behavior analysis in social networks. *IEEE Access*, 7:47114–47123, 2019. doi: 10.1109/ACCESS.2019.2909089.

[83] Apostolos Malatras, Dimitris Geneiatakis, and Ioannis Vakalis. On the efficiency of user identification: a system-based approach. *International Journal of Information Security*, 16(6):653–671, 2017. doi: 10.1007/s10207-016-0340-2.

[84] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 990–998, 2008. doi: 10.1145/1401890.1402008.

[85] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (MAS) and applications. In *Proceedings of the 24th international conference on world wide web (WWW)*, pages 243–246, 2015. doi: 10.1145/2740908.2742839.

[86] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. doi: 10.3115/v1/D14-1162.

[87] Manh Tuan Do, Se-eun Yoon, Bryan Hooi, and Kijung Shin. Structural patterns and generative models of real-world hypergraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 176–186, 2020. doi: 10.1145/3394486.3403060.

[88] Mengying Guo, Tao Yi, Yuqing Zhu, and Yungang Bao. JITuNE: Just-in-time hyperparameter tuning for network embedding algorithms. arXiv:2101.06427, 2021.

[89] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[90] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. arXiv:1807.03748, 2018.

[91] Ravichandra Addanki, Peter Battaglia, David Budden, Andreea Deac, Jonathan Godwin, Thomas Keck, Wai Lok Sibon Li, Alvaro Sanchez-Gonzalez, Jacklynn Stott, Shantanu Thakoor, and Petar Veličković. Large-scale graph representation learning with very deep GNNs and self-supervision. arXiv:2107.09422, 2021.

[92] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

[93] Yizhen Zheng, Shirui Pan, Vincent Cs Lee, Yu Zheng, and Philip S Yu. Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 10809–10820, 2022. doi: 10.48550/arXiv.2206.01535.

[94] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. doi: 10.48550/arXiv.1912.01703.

[95] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. arXiv:1903.02428, 2019.

[96] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, volume 29, 2015. doi: 10.1609/aaai.v29i1.9277.

[97] Philip S Chodrow, Nate Veldt, and Austin R Benson. Generative hypergraph clustering: from block-models to modularity. *Science Advances*, 2021. doi: 10.1126/sciadv.abh1303.

[98] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020. doi: 10.1162/qss_a_00021.

[99] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. doi: 10.48550/arXiv.1412.6980.

[100] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations (ICLR)*, 2021. doi: 10.48550/arXiv.2010.04592.

Algorithm 2 Clique expansion with sampling

Input: Hyperedge set E, sampling threshold k
Output: Clique-expansion graph E

```
1 E ;
2 foreach e 2 E do
3     if jej k then
4         E E [ (e 2)
5     else
6         foreach v 2 e do
7             V' uniformly sample k number of nodes from e n v
8             E E [ ff v, v' g : v' 2 V' g
9 return E
```

A Appendix: Additional Experimental Settings

A.1 Details of Datasets

DBLP is a co-authorship hypergraph where nodes and hyperedges correspond to publications and authors, respectively. Each publication's class is labeled according to its field of study. Trivago is a hotel-web search hypergraph where each node indicates each hotel and each hyperedge corresponds to a user. If a user (hyperedge) has visited the website of a particular hotel (node), the corresponding node is added to the respective user hyperedge. Furthermore, each hotel's class is labeled based on the country in which it is located. OGBN-MAG is originally a heterogeneous graph that contains comprehensive academic information including venue, author, publication, and affiliation information. We transform this heterogeneous graph into a hypergraph as described in Section 4, while a label of each node (publication) indicates a published venue of the corresponding publication.

A.2 Details of Graph-based Baseline Methods

Since graph representation models [19, 21] require ordinary graph structure as an input, we transform original hypergraph datasets into ordinary graph datasets by using *clique expansion*, where each hyperedge is replaced with a clique in the resulting graph. Formally, the clique expansion is a transformation of a given hyperedge set $\mathcal{E}$ to a clique expanded edge set $\mathcal{E}_G = \bigcup_{e \in E} \binom{e}{2}$.

Specifically, for full-graph datasets of DBLP and Trivago, we directly obtain $\mathcal{E}_G$ from $\mathcal{E}$, the entire hyperedge set. For full-graph datasets of OGBN-MAG, the size of the resulting clique expanded edges is too large to be loaded into the main memory. To reduce its scale, we additionally employ sampling, as described in Algorithm 2. Specifically, for each hyperedge e' whose size is greater than k and for each constituent node $v \in e'$, we uniformly sample k other nodes from e' (line 7) and create k edges joining v and each of the k sampled nodes. Here, we set $k = 10$ for the OGBN-MAG dataset. We fail to create full-graph datasets of AMiner and MAG since clique expansion runs out of memory even with small k around 3, and thus we cannot perform experiments on them.

For partitioned-graph datasets of DBLP, Trivago, and OGBN-MAG, we apply clique expansion to the hyperedge set in each partition and use the resulting clique-expanded edge set as that of the corresponding partition. For partitioned-graph datasets of AMiner and MAG, due to the scalability issue, we apply the sampling strategy described in Algorithm 2 to each partition $\mathcal{P}_i$ of $\boldsymbol{\mathcal{P}}$ (i.e., the input is $\mathcal{P}_i^E$ instead of $\mathcal{E}$) and treat the resulting edge set as the edge set of the corresponding partition. Here, we set k to 10.

A.3 Details of Hyperaprameter Settings

We now provide detailed hyperparameter settings of representation models and training methods. The number of layers and hidden dimension of all representation models are fixed to 2 and 128, respectively.

For representation models that are trained via supervised learning methods, we train each model for 100 epochs. We tune a learning rate of each model within $\{0.01, 0.001, 0.0001\}$. For every 10 epochs, we measure the validation AP score and save the model parameters. Then, we designate the checkpoint with the highest validation AP score as the final model parameters.

For representation models that are trained via all versions of PCL, we tune the number of self-supervised learning epochs within $\{25, 50\}$, while we set a broader search space, specifically $\{20, 40, 60, 80, 100\}$, for that of other self-supervised learning methods. We tune the learning rate of the self-supervised learning within $\{0.001, 0.0001\}$ for all self-supervised learning methods. In addition, for methods that require augmentation steps, we tune the extent of node feature augmentation p_v within $\{0.3, 0.4\}$, and the extent of topological augmentation p_e within $\{0.3, 0.4\}$. Furthermore, for methods that require negative samples for contrastive learning, we tune the number of negative samples N within $\{1, 2\}$. The temperature parameter τ for all self-supervised learning methods, and the scalar λ that controls the strength of inter-partition loss in PCL+PINS are both fixed to 0.5. Lastly, we train downstream task classifiers of all self-supervised learning methods with a learning rate of 0.001. We train the classifiers for 100 epochs, and for every 10 epochs, we measure the validation AP score and save the classifier parameters. Then, we designate the checkpoint with the highest validation AP score as the final classifier parameters.