# Search Behavior Prediction: A Hypergraph Perspective

Yan Han
University of Texas at Austin
Austin, TX, USA
yh9442@utexas.edu

Edward W Huang
Amazon
Palo Alto, CA, USA
ewhuang@amazon.com

Wenqing Zheng
University of Texas at Austin
Austin, TX, USA
w.zheng@utexas.edu

Nikhil Rao
Amazon
Palo Alto, CA, USA
nikhilsr@amazon.com

Zhangyang Wang
University of Texas at Austin
Austin, TX, USA
atlaswang@utexas.edu

Karthik Subbian
Amazon
Palo Alto, CA, USA
ksubbian@amazon.com

## Abstract

At E-Commerce stores such as Amazon, eBay, and Taobao, the shopping items and the query words that customers use to search for the items form a bipartite graph that captures search behavior. Such a query-item graph can be used to forecast search trends or improve search results. For example, generating query-item associations, which is equivalent to predicting links in the bipartite graph, can yield recommendations that can customize and improve the user search experience. Although the bipartite shopping graphs are straightforward to model search behavior, they suffer from two challenges: 1) The majority of items are sporadically searched and hence have noisy/sparse query associations, leading to a *long-tail* distribution. 2) Infrequent queries are more likely to link to popular items, leading to another hurdle known as *disassortative mixing*.

To address these two challenges, we go beyond the bipartite graph to take a hypergraph perspective, introducing a new paradigm that leverages auxiliary information from anonymized customer engagement sessions to assist the main task of query-item link prediction. This auxiliary information is available at web scale in the form of search logs. We treat all items appearing in the same customer session as a single hyperedge. The hypothesis is that items in a customer session are unified by a common shopping interest. With these hyperedges, we augment the original bipartite graph into a new *hypergraph*. We develop a ***D**ual-**C**hannel **A**ttention-Based **H**ypergraph Neural Network* (**DCAH**), which synergizes information from two potentially noisy sources (original query-item edges and item-item hyperedges). In this way, items on the tail are better connected due to the extra hyperedges, thereby enhancing their link prediction performance. We further integrate DCAH with self-supervised graph pre-training and/or DropEdge training, both of which effectively alleviate disassortative mixing. Extensive experiments on three proprietary E-Commerce datasets show that DCAH yields significant improvements of up to **24.6% in mean reciprocal rank (MRR)** and **48.3% in recall** compared to GNN-based baselines. Our source code is available at https://github.com/amazon-science/dual-channel-hypergraph-neural-network.

## CCS Concepts

• **Information systems** → **Recommender systems**.

## Keywords

graph neural network; hypergraph; hypergraph neural networks; DropEdge; contrastive learning; disassortative mixing

**ACM Reference Format:**
Yan Han, Edward W Huang, Wenqing Zheng, Nikhil Rao, Zhangyang Wang, and Karthik Subbian. 2023. Search Behavior Prediction: A Hypergraph Perspective. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (WSDM '23), February 27-March 3, 2023, Singapore, Singapore*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3539597.3570403

## 1 Introduction

Customers shop online in global E-Commerce stores such as Amazon, eBay, or Taobao for a variety of items. The stores' search engines are responsible for understanding each customer's query intent and then retrieving a list of relevant items. The retrieval processes often use algorithms that leverage a mix of lexical and behavioral signals. Lexical features include the query text and item titles, while behavioral signals include actions that customers take after executing queries, such as clicking on or purchasing a retrieved item.

In this work, we focus on modeling behavioral signals by first representing customer queries and items in the store catalogs as nodes in a bipartite graph. The behavioral signals (*e.g.*, clicks and purchases) serve as edges in the graph by capturing interactions between queries and items. Machine learning models can then learn to predict the query-item edges from this graph in the link prediction task. These predicted behavioral signals can further improve retrieval algorithms and therefore the customer experience.

Graph neural networks (GNNs) [1, 2] have achieved state-of-the-art results for link prediction [3] across a wide range of graph-structured data. Despite strong performance on many types of graphs, they still suffer from two challenges caused by the raw query-item bipartite graph.

- First, the success of GNNs hinges on dense and high-quality connections. Unfortunately, in typical bipartite shopping graphs, few items are popular enough to be consistently


*WSDM '23, February 27-March 3, 2023, Singapore, Singapore*

ACM ISBN 978-1-4503-9407-9/23/02…$15.00
https://doi.org/10.1145/3539597.3570403

searched for, such as everyday household items. The majority of items face the cold-start problem [4], where very few interactions from customers lead to sparse and noisy query-item edges. These items form a less reliable part of the graph, known as the long-tail part, and typical GNNs do not perform well on the long-tail part [4, 5].

- The second challenge lies in the disassortative mixing commonly observed in our bipartite shopping graphs [6]: unpopular queries/items are more likely to be connected to popular items/queries. We show through experimental results that disassortative mixing is a major hurdle for query-item predictions. We further demonstrate that disassortative mixing in shopping graphs leads to the over-smoothing [7] problem, triggering performance drops in GNN link prediction. More detailed illustrations of why and how disassortative mixing happens in our graph are in Sec 3.2.2.

We explore the two aforementioned challenges, identifying their causes and proposing solutions. Our solution involves using the wealth of meta-information in the bipartite shopping graph, which is often overlooked. We refer to this meta-information as *auxiliary* information. An example of auxiliary information is anonymized search records. These records are available at web scale but are difficult to model in normal bipartite graphs due to their higher-order relations. However, they can contain information not present in the original pairwise relations. For example, items occurring in the same customer search session may be connected by a common theme that query-item links do not capture. Therefore, such auxiliary information can be leveraged to assist the main task of query-item link prediction. A detailed illustration of the advantages of utilizing the hypergraph to model the auxiliary information is in Sec 3.2.1. Both the original bipartite graph and a hypergraph constructed from auxiliary information provide valuable information that the other graph does not. Therefore, we jointly consider both graphs when predicting user search behavior. Our goal is to determine *how to optimally leverage the auxiliary hypergraph to assist the main task of query-item link prediction.*

In this paper, we study the problem of search behavior prediction in the following setting: given two different yet related graphs, a bipartite graph and a hypergraph, we aim to jointly use them to predict query-item links. Our hypothesis is that using both graphs is better than using just one. We conceptually illustrate this unique problem setting in Figure 1. Taking the E-Commerce problem as an example, we consider two different graphs: a bipartite query-item graph constructed from search logs and a hypergraph constructed from auxiliary information. Although the two graphs share the same node set, their distributions may be different. For instance, everyday items are densely connected nodes in the query-item graph due to their popularity. On the other hand, they may be searched and purchased alone in customer sessions, resulting in few connections in the hypergraph.

To tackle this problem setting, we first propose the ***D****ual-****C****hannel* ***A****ttention-Based* ***H****ypergraph Neural Network*, or **DCAH**, which can jointly learn better representations from the two input graphs. In particular, we use the bipartite graph to model the pairwise relations while leveraging the hypergraph to model the higher-order relations. To integrate these two relations, we leverage an attention mechanism to learn the optimal weight each relation should have for the downstream task. Furthermore, to improve DCAH's ability to generalize to the long tail and alleviate the over-smoothing problem, we incorporate self-supervised learning and the DropEdge [8] strategies. Self-supervised learning works well for few-shot generalization [9], which can alleviate the fact that the long tail faces label sparsity. Meanwhile, DropEdge relieves the over-smoothing problem [10, 11] introduced by disassortative mixing. We provide a detailed illustration of the two strategies in Sec 3.4. In summary, this paper makes the following contributions:

- We study the query-item link prediction problem in a special setting: improving the main task of search behavior prediction (link prediction) with the aid of accessible auxiliary information. We use the auxiliary information to create a second hypergraph supplementing the original bipartite graph.
- We propose a new framework, DCAH, to tackle our unique problem setting by jointly capturing the information of the bipartite graph and the hypergraph. Additionally, we seamlessly incorporate self-supervised learning to improve DCAH's generalization ability on the original graph's long tail and DropEdge to relieve the over-smoothing problem.
- We conduct experiments on three proprietary E-Commerce datasets. Our experimental results show that our approach not only improves the overall link prediction performance, but also generalizes better to the long tail compared to existing off-the-shelf approaches.

## 2 Related Work

### 2.1 Session-Based Recommendation

Session-based recommendation (SBR) shares some characteristics with our problem setting. However, though both settings involve user search behavior, they differ in their motivations, solutions, and final goals. Generally, a session is a transaction with multiple purchased items in one shopping event. SBR focuses on next-item prediction by using real-time user behavior [12]. It can be seen as a form of personalized recommendation. However, we focus on query-item link prediction rather than user-oriented recommendation, and our problem setting does not involve inter-user relationships.

Additionally, initial exploration in SBR mainly focuses on sequence modeling, including traditional Markov decision processes [13] and deep learning models such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs) [14]. More recently, GNN has shown promising results in SBR [15, 16]. To capture the complex higher-order item correlations, DHCN [12] and SHARE [17] constructed session-induced hypergraphs to model item correlations and user intent in SBR, which achieved superior performance against previous GNN-based models. This also indicates the advantages of hypergraphs over normal graphs when modeling high-order relations.

### 2.2 Graphs/Hypergraphs in Recommendation

Since graphs can naturally model relational data such as item-item, user-user, or user-item interactions, GNN-based models have performed well in recommendation systems [18, 19]. GCMC [20] utilized neural graph autoencoders to reconstruct a user-item rating

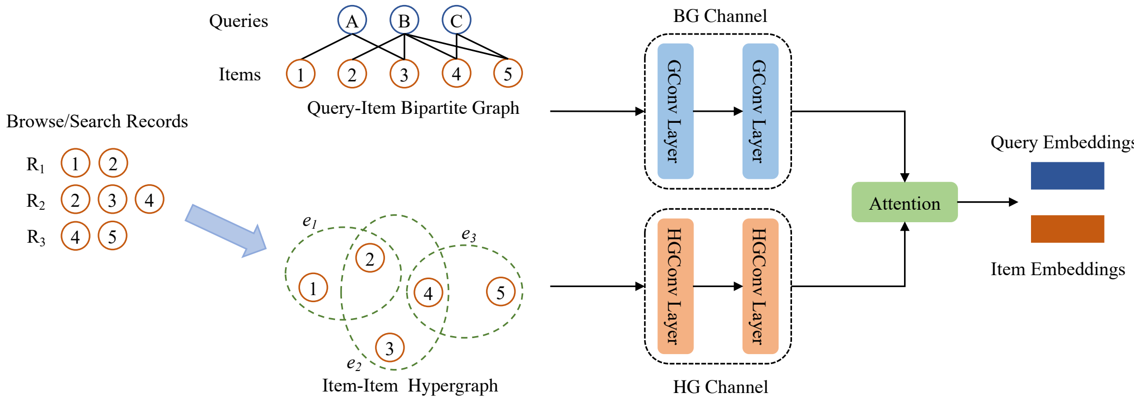


**Figure 1: Construction of the hypergraph and the pipeline of the proposed DCAH model.**

graph. NGCF [21] proposed to construct a user-item bipartite graph and leverage GNNs to model relations between users and items. In social recommendation, relations between users can be exploited with social graphs [16, 22]. For example, DiffNet [22] adopted GCNs to model the diffusion of user embeddings among their social connections. In addition, inspired by the ability of hypergraphs to model complex higher-order dependencies [23, 24], recent works have attempted to capture interactions via hypergraph structure and uniform node-hyperedge connections, including HyRec [25], DHCF [26], and MHCN [27]. HyRec attempted to propagate information among multiple items by considering users as hyperedges. DHCF modeled hybrid multi-order correlations between users and items based on the hypergraph structure. MHCN leveraged social relations to create a hypergraph that improves recommendation quality when user-item interaction data is sparse.

## 2.3 Contrastive Representation Learning

Contrastive learning effectively captures consistent feature representations under different views [28]. It has achieved promising results in various domains, such as visual data representation [29], language data understanding [30], graph representation learning [31, 32], and recommendation systems [27, 33]. These contrastive learning approaches use auxiliary signals specific to their data or tasks. Concurrently, HCCF [33] jointly captured local and global collaborative relations with a hypergraph-enhanced cross-view contrastive learning architecture. However, their self-supervised learning setting is different from ours, since we augment the original bipartite graph and hypergraph and maximize the agreement between the two generated variants, following the standard graph contrastive learning pipeline [32].

## 2.4 Disassortative Mixing

Many scale-free networks in the real world show the tendency where highly connected nodes link with other highly connected nodes, which is defined as assortative mixing. The reverse is also true in some networks, where highly connected nodes are more likely to make links with isolated, less connected nodes, which is defined as disassortative mixing. Biological and technological networks typically show a disassortative mixing pattern. In [34], the authors found that online social networks also exhibit a disassortative mixing pattern, and [6] observed disassortative mixing in the customer interaction networks of E-Commerce stores. Although disassortative mixing has been observed in the E-Commerce domain, to the best of our knowledge, no formal study on how disassortative mixing affects the performance of recommendation models and how to relieve the issue currently exists. In this paper, we reveal how disassortative mixing triggers the over-smoothing problem and utilize two existing off-the-shelf approaches, self-supervised learning and DropEdge [8], to mitigate it.

# 3 Methodology

In this section, we first introduce the notations and definitions used throughout the paper, and then we show how the auxiliary information is modeled as a hypergraph. After that, we revisit the two challenges in the query-item graph. Then, we present the DCAH framework. Finally, we discuss how we integrate self-supervised learning and DropEdge to improve the generalization ability of our model and relieve the over-smoothing problem.

## 3.1 Notations, Definitions, and Hypergraph Construction

***Notations.*** Let $Q = \{q_1, q_2, q_3, \dots, q_{N_Q}\}$ and $I = \{i_1, i_2, i_3, \dots, i_{N_I}\}$ denote the sets of queries and items, where $N_Q$ and $N_I$ are the number of queries and the number of items, respectively. $N$ denotes the total number of queries and items. Each browse/search record is represented as a set $\{i_{r,1}, i_{r,2}, i_{r,3}, \dots, i_{r,m}\}$, where $i_{r,k} \in I (1 \le k \le m)$ represents an interaction between an anonymous user and an item within the browse/search record $r$.

***Hypergraph Definition.*** A hypergraph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of $N$ unique vertices and $\mathcal{E}$ is a set of $M$ hyperedges. Each hyperedge $e \in \mathcal{E}$ connects two or more vertices. The hypergraph can be represented by an incidence matrix $\mathbf{H} \in \mathbb{R}^{N \times M}$, where $H_{ie} = 1$ if the hyperedge $e \in \mathcal{E}$ contains a vertex $v_i \in \mathcal{V}$, otherwise 0. For each vertex and hyperedge, their degrees $D_{ii}$ and $B_{ee}$ are defined as $D_{ii} = \sum_{e=1}^{M} H_{ie}$ and $B_{ee} = \sum_{i=1}^{N} = H_{ie}$, respectively. $\mathbf{D}$ and $\mathbf{B}$ are diagonal matrices. $D_{ii}$ and $B_{ee}$ are the $i$-th and $e$-th diagonal elements of $\mathbf{D}$ and $\mathbf{B}$, respectively.

***Hypergraph Construction.*** To capture relationships beyond pairwise ones in the browse/search records, we adopt a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to represent each record as a hyperedge. Formally, we

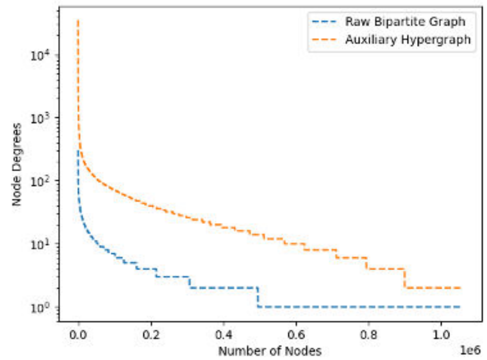


**Figure 2: Node-degree distribution of one E-Commerce query-item bipartite graph and the related auxiliary hypergraph.**

denote each hyperedge as $\{i_{r,1}, i_{r,2}, i_{r,3}, \ldots, i_{r,m}\} \in \mathcal{E}$ and each item as $i_{r,k} \in \mathcal{V}(1 \leq k \leq m)$. The hypergraph construction process is shown in the left part of Figure 1. As illustrated, the original record data is organized as linear sequences where two items, $i_{r,m-1}$ and $i_{r,m}$, are only connected if a user interacted with $i_{r,m-1}$ before $i_{r,m}$. After transforming the record data into a hypergraph, any two items browsed/searched in the same record are connected. This allows us to concretize the many-to-many higher-order relations.

## 3.2 Two Challenges in the Query-Item Graph

In this section, we revisit the two unique challenges of the raw query-item bipartite shopping graphs: the long-tail distribution and disassortative mixing. Specifically, we investigate the node-degree distribution of the query-item graph and show how the auxiliary hypergraph helps relieve the problem. Additionally, we empirically demonstrate why disassortative mixing is a major hurdle for link prediction models.

*3.2.1 Long-Tail Distribution* Many real networks are scale free in nature, i.e, the node degrees follow power-law distributions [35]. As a type of real network, query-item graphs also tend to have a similar power distribution pattern. The degree distribution plot of one E-Commerce query-item graph used in our paper is presented in Figure 2 (blue line). In the figure, we can find that only a few (less than 10%) nodes have node degree greater than ten, while the majority of nodes have fewer than ten neighbors. In summary, the degree distribution of the E-Commerce query-item graph follows the long-tail degree distribution. The long-tail degree distribution indicates that while there exist some popular queries/items which connect with many other items/queries, most queries/items connect only to a few items/queries, which is called the long-tail part. In addition, the long-tail part is less reliable since behavioral query-item relations (*e.g.*, from clicks or views) can introduce noise (*e.g.*, misclicks) to the constructed interaction graph [36]. In this case, existing GNNs fail to perform well on the long-tail part due to data sparsity and noisy interactions. In addition, most GNN-based methods design the message-passing mechanism such that the embedding propagation is only performed with neighbors in the original graph.

As mentioned in the introduction, there exists one type of auxiliary information that we can introduce to alleviate the above issue: anonymized search records. Previous methods, such as the K-Nearest Neighbor (KNN) graph based on node feature similarity [37, 38], methods based on conventional Markov chains [39, 40], and methods based on factorization [41, 42], have been proposed to model some kinds of auxiliary information in similar problem settings. However, these methods cannot capture higher-order dependencies [43, 44] such as those in customer search sessions. Hypergraphs [45], which generalize the concept of an edge to connect more than two nodes, can organically model complex, higher-order relations. In addition, the auxiliary hypergraph can generally enrich the connectivity of the nodes of the raw bipartite graph. As shown in Figure 2 (yellow line), the node degrees are boosted via a huge margin, especially on the long-tail part.

*3.2.2 Disassortative Mixing* Since there exist only a few popular queries and items in the query-item graph, we assume it is not uncommon that some popular queries/items may link to unpopular items/queries. To verify this phenomenon, we investigate the popular queries of the graph, and find that the queries with highest degrees tend to be very general queries, such as "toothbrush", "tissue", "TV", etc. Such queries connect to many related items, some of which are unpopular or specialized items, such as *premium bamboo toothbrush*, *baby tissue*, *LED 4K UHD TV* etc. Similarly, for popular everyday household items like *water*, *toilet paper*, etc., due to popularity, many customers may search for them a large number of times. In their search, the customers may generate queries with infrequent terms, such as "best spring water in the world" and "ultra soft toilet paper for infant." Based on this observation, we believe that disassortative mixing exists in the E-Commerce query-item graph and conduct the degree assortativity analysis of the graph.

Degree assortativity, $r \in [-1, 1]$, is a measure of similarity between nodes and their neighbors in terms of degree [46]. Formally, degree assortativity is defined as the Pearson Correlation Coefficient of degrees between all pairs of connected nodes. The value $r = -1$ implies that the network is totally disassortative (negative correlation) and $r = 1$ implies that the network is totally assortative (positive correlation). The assortativity plot of one E-Commerce query-item graph used in this paper is shown in Figure 3. The query-item bipartite graph is relatively disassortative with $r = -0.07$. In the figure, we observe that the neighbors of high-degree nodes have relatively low degrees, while the neighbors of some low-degree nodes have very high degrees. In summary, the assortativity plot of the query-item graph suggests that unpopular queries/items are more likely to connect to popular items/queries, making the graph disassortative, which agrees with our intuition.

Relative degree evaluates a node's degree compared to its neighbors' degrees [47]. When all the nodes have the same degree, the relative degree equals 1. On the other hand, relative degree will be low if a node's degree differs significantly from its neighbors' degrees. As a type of disassortative graph, the relative degree of the E-Commerce query-item bipartite graph is relatively low compared to that of assortative graphs. In [47], the authors theoretically showed that the low relative degree will trigger the over-smoothing problem, such that all nodes' representations will converge to a stationary point with an increase in the number of layers of graph

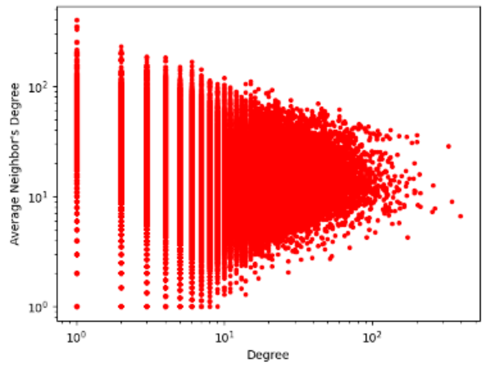


**Figure 3: Assortativity plot (degree vs average neighbor's degree) for one E-Commerce query-item bipartite graph.**

convolutions. Although over-smoothing is usually discussed in the node classification task, it also acts as a major hurdle in the link prediction task. We show that this is true in our query-item prediction task with our experiment results.

## 3.3 A General DCAH Framework

Figure 1 sketches the overall framework of DCAH. At a high level, DCAH consists of a bipartite graph channel and a hypergraph channel. The bipartite graph channel captures pairwise query-item relations while the hypergraph channel captures higher-order item-item relations. DCAH uses an attention layer to compute the optimal weighted combination of the channel outputs for the downstream task, query-item prediction.

***Bipartite Graph Channel and Convolution.*** The original bipartite shopping graph is an undirected graph containing only query-item relations. First, we initialize the item embeddings specific to the bipartite graph channel, $\mathbf{X}_I^{(0)BG}$. For the query node, the raw text of the user query is available to generate node features using Byte-Pair Encodings [48], denoted as $\mathbf{X}_Q^{(0)BG}$. The adjacency matrix for this bipartite graph $\mathcal{G}_B$ is defined as $\mathbf{A} \in \mathbb{R}^{N \times N}$. Let $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. $\hat{\mathbf{D}} \in \mathbb{R}^{N \times N}$ is a diagonal degree matrix where $\hat{\mathbf{D}}_{p,p} = \sum_{q=1}^{n} \hat{\mathbf{A}}_{p,q}$. The bipartite graph convolution is then defined as:

$$\mathbf{X}^{(l+1)BG} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}^{(l)BG}\Theta_{BG} \tag{1}$$

where $\Theta_{BG}$ is the learnable shared parameter of the convolutional layer. In each convolution, the query/item node gathers information from its neighboring item/query nodes. By doing so, the learned $\mathbf{X}$ can capture the pairwise query-item relation information. We then pass $\mathbf{X}^{(0)}$ through $L$ graph convolutional layers to obtain the final output embeddings $\mathbf{X}^{(L)}$. We use $\mathbf{X}_Q^{(L)BG}$ and $\mathbf{X}_I^{(L)BG}$ to denote query and item embeddings, respectively.

***Hypergraph Channel and Convolution.*** The hypergraph channel is used to capture the item-level higher-order relations. The primary challenge of defining a convolution operation over the hypergraph is how the embeddings of items are propagated. Similar to the bipartite graph channel, we first initialize the hypergraph-channel-specific item embeddings $\mathbf{X}_I^{(0)HG}$. Following the spectral hypergraph convolution proposed in [23], we define our hypergraph convolution as:

$$\mathbf{X}_I^{(l+1)HG} = \mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^T\mathbf{D}^{-1/2}\mathbf{X}_I^{(l)HG}\Theta_{HG} \tag{2}$$

Here, $\mathbf{W}$ is the hyperedge weight matrix, where we assign all hyperedges the weight of 1, and $\Theta_{HG}$ is the learnable shared parameter of the convolutional layer. The hypergraph convolution can be viewed as a two-stage refinement performing a node-hyperedge-node feature transformation on the hypergraph structure. The multiplication of $\mathbf{H}^T$ implements the information aggregation from nodes to hyperedges. Pre-multiplying $\mathbf{H}$ can be viewed as aggregating information from hyperedges to nodes. Note that $\mathbf{D}$ and $\mathbf{B}$ play the role of normalization. After passing $\mathbf{X}_I^{(0)HG}$ through $L$ hypergraph convolutional layers, we obtain the final item embeddings $\mathbf{X}_I^{(L)HG}$.

***Inter-Channel Aggregation.*** With the dual-channel structure, we generate two groups of item embeddings, $\mathbf{X}_I^{(L)BG}$ and $\mathbf{X}_I^{(L)HG}$, via the bipartite graph and hypergraph channels, respectively. Since both groups of item embeddings will contain some valuable information, we add an additional attention layer to learn the optimal weighted combination of the two groups of embeddings for our downstream task. This strategy will improve robustness when fusing the two groups' embeddings. Specifically, we first transform both channels' embeddings through a nonlinear transformation (*e.g.*, a single-layer MLP). Then, we measure the importance of a channel-specific embedding as the similarity between the transformed embedding and a channel-level attention vector $\mathbf{q}$. Finally, we average the importance of all channel-specific item embeddings. We can interpret this as the importance of each channel, denoted as $w_{\phi_c}$.

$$w_{\phi_c} = \frac{1}{|\mathcal{V}|}\sum_{i \in \mathcal{V}} \mathbf{q}^T \cdot \tanh\left(\mathbf{W} \cdot \mathbf{x}_i^{\phi_c} + \mathbf{b}\right),\ \phi_c \in \{BG, HG\} \tag{3}$$

Here, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, $\mathbf{x}_i^{\phi_c}$ is the channel-level embedding, $\tanh(\cdot)$ is the activation function, $\mathbf{q}$ is the channel-level attention vector, and $\mathcal{V}$ is the set of the item nodes. Note that, for the sake of meaningful comparison, all the above parameters are shared across all channels and channel-specific embeddings. After obtaining the importance of each channel, we normalize them with the softmax function. The weight of channel $\phi_c$, denoted as $\beta_{\phi_c}$, can be obtained by normalizing the above importance of all channels using the softmax operator:

$$\beta_{\phi_c} = \frac{\exp w_{\phi_c}}{\sum_{\phi_c} \exp w_{\phi_c}},\ \phi_c \in \{BG, HG\} \tag{4}$$

$\beta_{\phi_c}$ can be interpreted as the contribution of channel $\phi_c$ for a specific task, with a higher $\beta_{\phi_c}$ indicating a higher importance for channel $\phi_c$. With the learned weights as coefficients, we can fuse the two channel-specific embeddings to obtain the final item embedding $\mathbf{X}_I^L$ as follows:

$$\mathbf{X}_I^L = \beta_{\phi_{BG}} \cdot \mathbf{X}_I^{(L)BG} + \beta_{\phi_{HG}} \cdot \mathbf{X}_I^{(L)HG} \tag{5}$$

## 3.4 Enhancing DCAH with Self-Supervised Learning and DropEdge

Although the auxiliary hypergraph can augment the original bipartite graph, data sparsity, the long-tail issue, and disassortative mixing still exist, which might impede the generalization ability of our model. Inspired by prior studies of self-supervised learning and DropEdge on graphs, we seamlessly integrate these two simple but effective training tricks into DCAH.

***Self-Supervised Learning (SSL).*** In this work, we follow the graph self-supervised learning framework proposed in [32], which consists of the four major components: 1) Graph data augmentation; 2) GNN-based encoder; 3) Projection head; 4) Contrastive loss function. It should be noted that we made some modifications to components 1, 2, and 4. Specifically, for the graph data augmentation, the original framework only has one input graph, while we consider two input graphs, the bipartite graph $\mathcal{G}_B$ and the hypergraph $\mathcal{G}_H$. Although the hypergraph is different from the bipartite graph, the representation formats are similar. Therefore, both given graphs <$\mathcal{G}_B, \mathcal{G}_H$> can undergo graph data augmentations proposed in [32] to obtain two correlated views, <$\hat{\mathcal{G}}_{B_i}, \hat{\mathcal{G}}_{H_i}$>, <$\hat{\mathcal{G}}_{B_j}, \hat{\mathcal{G}}_{H_j}$>, as a positive pair. Then, our model DCAH extracts node-level embeddings $\mathbf{h}_i$ and $\mathbf{h}_j$ for two augmented graph pairs <$\hat{\mathcal{G}}_{B_i}, \hat{\mathcal{G}}_{H_i}$> and <$\hat{\mathcal{G}}_{B_j}, \hat{\mathcal{G}}_{H_j}$>, respectively. After that, as advocated in [49], a projection head in the form of a two-layer perceptron (MLP) is applied to obtain $z_i$ and $z_j$. Finally, a node-level contrastive loss $\mathcal{L}(\cdot)$ is utilized to enforce maximizing the consistency between positive pairs $z_i$ , $z_j$ compared with negative pairs:

$$\mathcal{L} = -\log \frac{\exp(sim(z_{n,i}, z_{n,j})/\tau)}{\sum_{n'=1,n'\neq n}^{N} \exp(sim(z_{n,i}, z_{n',j})/\tau)} \qquad (6)$$

where $sim(z_{n,i}, z_{n,j}) = z_{n,i}^T z_{n,j}/||z_{n,i}||||z_{n,j}||$, $z_{n,i}$, $\tau$ is the temperature parameter and $N$ is number of nodes.

In summary, our contrastive learning component maximizes the agreement between two variants of two graph augmentations. Furthermore, the setting is different from that of cross-channel views contrastive learning [12] and global-local views contrastive learning [33].

***DropEdge.*** As mentioned in Sec 3.2.2, the disassortative mixing of the query-item graph triggers the over-smoothing problem, which acts as a major hurdle to the models' performance on the query-item predictions. Previous literature [8, 10, 11] have theoretically and empirically shown that the DropEdge [8] can help relieve over-smoothing. Because DropEdge randomly removes a certain number of edges from the input graph at each training epoch, it can be easily applied to hypergaphs, functioning as a data augmentation mechanism. Therefore, it can be flexibly incorporated into our model DCAH, similar to self-supervised learning.

# 4 Evaluation

Our experiments aim to answer the following research questions:

- **RQ1**: What is the performance of our DCAH compared to baselines?
- **RQ2**: How does the auxiliary hypergraph contribute to relieving the long-tail issue of the raw bipartite graph?
- **RQ3**: How do SSL and DropEdge perform in alleviating the over-smoothing problem triggered by disassortative mixing?

## 4.1 Experimental Settings

*4.1.1 Dataset Overview.* We evaluate the proposed framework on three proprietary E-Commerce datasets of queries and items, which are sampled from three different market locales of this E-Commerce platform. They can all be naturally constructed as bipartite shopping graphs. The query-item associations are sampled so as not to reveal raw traffic distributions. We also create three related auxiliary item-item hypergraphs from the three locales' corresponding anonymized customer engagement sessions, where all customer-identifiable information has been properly anonymized. The full dataset statistics are summarized in Table 1. In the table, the density indicates that all the three E-Commerce datasets are generally sparse, among which E-Commerce 3 is the sparsest. As for the degree assortativity, E-Commerce 3 is more disassortative than E-Commerce 1 and 2.

**Table 1: Dataset statistics.**

| | E-Commerce 1 | E-Commerce 2 | E-Commerce 3 |
|---|---|---|---|
| Num. of Nodes | 1,181,247 | 1,059,963 | 1,375,842 |
| Num. of Edges | 1,783,333 | 1,634,402 | 2,148,822 |
| Num. of Auxiliary HyperEdges | 329,852 | 238,342 | 388,042 |
| Degree Assortativity | 0.132 | 0.089 | -0.07 |
| Density | $1.28 \times e^{-6}$ | $1.45 \times e^{-6}$ | $1.14 \times e^{-6}$ |

*4.1.2 Data Splits.* We create training, validation, and test splits of the data to specifically study tail and cold-start nodes. Because we have a bipartite graph and a hypergraph, we manually split the nodes into four parts, which we refer to as the *head, tail1, tail2*, and *isolation* parts.

We create the *head* part by first selecting nodes that are in the top 20% highest-degree nodes of both the bipartite graph and the hypergraph, then inducing a subgraph with the selected nodes. The *tail1* part induces a subgraph on the nodes shared between the top 20% highest-degree nodes of bipartite graph and the top 20% – 60% highest-degree nodes of hypergraph. Similarly, the *tail2* part induces a subgraph on the nodes shared between the top 20% highest-degree nodes of hypergraph and the top 20% – 60% highest-degree nodes of the bipartite graph. The *isolation* part induces a subgraph on the nodes in the bottom 40% highest-degree nodes of both the bipartite graph and the hypergraph.

For each part, we randomly sample 20% of the edges for testing and 10% for validation. The remaining edges are used for training. In summary, our data split is 70% training, 10% validation, and 20% testing. Within the test set, each of the four parts, *head, tail1, tail2*, and *isolation*, counts as 25%. In addition, since the downstream task is link prediction, we must sample negative edges. For each positive edge in the original graph, containing (*source*, *destination*) nodes, we corrupt the edge by replacing its *source* or *destination* with 100 randomly sampled negative edges (50 for *source* and 50 for *destination*), while ensuring the resulting negative edges are not already present in the original graph. We ensure no overlap between the positive and negative edges across all split parts.

*4.1.3 Evaluation Metrics.* We use *Recall@N* and Mean Reciprocal Rank (*MRR*) as the evaluation metrics, which are widely used in link prediction tasks [50]. $N$ is the number of the top predictions. In our experiments, we set $N$ to be the number of the true positive edges

**Table 2: Performance comparison on the E-Commerce 1 in terms of *MRR* and *Recall*. The best results are bold.**

| Models | Head | | Tail1 | | Tail2 | | Isolation | |
|---|---|---|---|---|---|---|---|---|
| | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N |
| GCN | 14.68 ±3.10 | 6.35 ±1.30 | 13.83 ±2.66 | 4.86 ±1.36 | 15.70 ±1.60 | 3.44 ±0.85 | 25.35 ±6.48 | 3.77 ±1.16 |
| GCN+DropEdge | 23.24 ±2.37 | 14.11 ±1.74 | 23.22 ±2.07 | 14.73 ±1.51 | 20.95 ±5.15 | 11.02 ±3.44 | 34.59 ±6.52 | 18.71 ±1.85 |
| GCN+SSL | 58.72 ±2.89 | 34.24 ±2.37 | 58.05 ±2.65 | 33.85 ±2.01 | 50.48 ±2.46 | 31.69 ±2.38 | 51.10 ±0.79 | 31.48 ±0.41 |
| GCN+SSL+DropEdge | 56.68 ±2.04 | 33.95 ±2.33 | 56.74 ±2.25 | 31.56 ±1.50 | 52.20 ±1.59 | 33.43 ±2.15 | 55.74 ±3.34 | 32.87 ±0.44 |
| HyperGCN | 7.47 ±0.84 | 2.10 ±0.79 | 10.77 ±1.00 | 2.36 ±0.71 | 14.26 ±1.89 | 3.34 ±2.14 | 26.15 ±3.99 | 4.08 ±0.95 |
| HyperGCN+DropEdge | 12.48 ±1.68 | 10.45 ±4.89 | 16.60 ±1.90 | 9.73 ±4.25 | 21.02 ±4.08 | 10.28 ±3.06 | 24.39 ±4.16 | 12.68 ±5.21 |
| HyperGCN+SSL | 40.23 ±1.05 | 30.07 ±1.62 | 36.25 ±0.53 | 27.45 ±0.74 | 57.13 ±3.00 | 38.63 ±1.44 | 58.80 ±3.80 | 38.73 ±0.25 |
| HyperGCN+SSL+DropEdge | 41.22 ±0.16 | 31.01 ±1.01 | 35.67 ±0.29 | 28.29 ±0.92 | 62.18 ±0.27 | 44.39 ±0.51 | 62.33 ±6.52 | 44.46 ±1.48 |
| **DCAH** | 16.06 ±4.36 | 6.40 ±1.62 | 15.37 ±3.83 | 5.09 ±1.23 | 16.25 ±4.09 | 4.02 ±0.94 | 30.10 ±9.54 | 5.10 ±2.11 |
| **DCAH**+DropEdge | 24.26 ±2.97 | 15.83 ±2.40 | 22.96 ±2.75 | 15.35 ±2.19 | 23.81 ±4.31 | 17.25 ±2.99 | 34.47 ±6.32 | 17.84 ±1.96 |
| **DCAH**+SSL | 59.47 ±3.55 | 35.56 ±2.29 | 58.03 ±3.30 | 34.47 ±2.05 | 60.88 ±1.05 | 42.69 ±2.17 | 64.00 ±3.60 | 43.78 ±0.92 |
| **DCAH**+SSL+DropEdge | **60.96** ±3.21 | **36.67** ±2.32 | **59.59** ±2.78 | **36.32** ±2.44 | **63.79** ±3.24 | **45.85** ±2.83 | **65.70** ±3.10 | **45.80** ±0.68 |

**Table 3: Performance comparison on the E-Commerce 2 in terms of *MRR* and *Recall*. The best results are bold.**

| Models | Head | | Tail1 | | Tail2 | | Isolation | |
|---|---|---|---|---|---|---|---|---|
| | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N |
| GCN | 14.88 ±2.50 | 5.69 ±1.16 | 15.49 ±2.58 | 4.73 ±0.96 | 14.73 ±2.79 | 3.53 ±0.65 | 24.33 ±5.59 | 4.05 ±0.87 |
| GCN+DropEdge | 23.26 ±2.26 | 13.91 ±1.12 | 24.77 ±2.46 | 12.01 ±0.76 | 24.25 ±5.83 | 10.31 ±0.81 | 39.36 ±7.47 | 12.98 ±0.74 |
| GCN+SSL | 56.44 ±2.80 | 33.98 ±1.98 | 58.01 ±2.78 | 35.37 ±1.84 | 58.81 ±4.46 | 35.09 ±2.97 | 64.86 ±4.32 | 36.61 ±0.52 |
| GCN+SSL+DropEdge | 57.86 ±1.74 | 34.20 ±1.59 | 57.98 ±2.70 | 36.91 ±1.30 | 59.55 ±4.33 | 35.88 ±2.80 | 65.19 ±3.45 | 36.90 ±0.45 |
| HyperGCN | 10.36 ±0.94 | 2.76 ±0.76 | 14.70 ±1.08 | 3.21 ±0.59 | 19.64 ±2.19 | 3.30 ±0.49 | 39.14 ±5.92 | 6.63 ±1.89 |
| HyperGCN+DropEdge | 27.48 ±2.39 | 7.02 ±2.34 | 32.59 ±2.10 | 10.40 ±3.39 | 41.76 ±4.03 | 17.50 ±6.04 | 53.14 ±8.64 | 20.98 ±7.69 |
| HyperGCN+SSL | 52.68 ±0.99 | 35.49 ±1.34 | 51.34 ±0.61 | 37.60 ±0.72 | 72.56 ±0.65 | 52.67 ±1.19 | 78.82 ±5.26 | 55.68 ±1.40 |
| HyperGCN+SSL+DropEdge | 56.26 ±0.37 | 37.53 ±1.23 | 53.03 ±0.16 | 36.69 ±0.83 | 75.10 ±0.49 | 54.47 ±1.27 | 80.73 ±5.70 | 57.26 ±1.44 |
| **DCAH** | 17.22 ±5.93 | 6.01 ±1.83 | 17.85 ±5.24 | 5.00 ±1.53 | 17.97 ±2.85 | 3.72 ±0.88 | 36.69 ±6.00 | 4.68 ±1.61 |
| **DCAH**+DropEdge | 28.58 ±1.02 | 15.25 ±2.02 | 34.43 ±4.35 | 15.16 ±1.16 | 44.73 ±2.66 | 13.73 ±1.50 | 54.99 ±5.65 | 12.53 ±1.17 |
| **DCAH**+SSL | 69.53 ±0.82 | 46.50 ±1.79 | 70.80 ±0.78 | 47.85 ±1.84 | 82.48 ±0.49 | 57.46 ±1.71 | 81.14 ±0.92 | 58.13 ±0.59 |
| **DCAH**+SSL+DropEdge | **72.38** ±0.95 | **47.59** ±2.74 | **73.80** ±0.97 | **48.78** ±1.67 | **84.46** ±0.55 | **59.56** ±1.43 | **83.43** ±0.72 | **59.75** ±0.67 |

of the testing part. *Recall@N* measures whether the actual edge is on the top-$N$ predictions. Thus, it denotes the proportion of the number of predictions containing the actual edge among all positive edges. *MRR* measures the ranking performance of the model as an average of reciprocal ranks of the actual edge within the prediction. Both evaluation metrics are appropriate to measure the model's performance, but *Recall@N* is more related to the ability to find unseen edges. Unseen edges are scarce in the tail and isolation parts and are therefore more important to predict. Thus, *Recall@N* is more important than *MRR*, especially under the cold-start scenario. We report the mean and standard deviation of *Recall@N* and *MRR* after 10 runs.

*4.1.4 Baselines.* Due to the uniqueness of our problem setting, the meaningful baselines are limited. In our experiments, we compare our DCAH with two commonly used baselines: GCN [1], and HyperGCN [23]. And for fair comparison, the bipartite graph channel and the hypergraph channel use the GCN and HyperGCN, respectively. Besides, we also adapt the SSL and/or DropEdge strategies to the GCN and HyperGCN as additional baselines.

*4.1.5 Hyperparamter Settings.* We use Adam optimizer with the learning rate of $1e-3$. The hidden state dimension is 64. For both the bipartite graph and hypergraph channels, we stack two propagation layers. The batch size and dropout ratio are set to 1024 and 0.25, respectively. The temperature parameter $\tau$ is set to 0.1 to control the strength of gradients in our contrastive learning.

**Table 4: Performance comparison on the E-Commerce 3 in terms of *MRR* and *Recall*. The best results are bold.**

| Models | Head | | Tail1 | | Tail2 | | Isolation | |
|---|---|---|---|---|---|---|---|---|
| | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N | MRR | Recall@N |
| GCN | 15.49 ±3.48 | 7.28 ±1.22 | 13.44 ±1.50 | 4.08 ±0.63 | 17.04 ±2.11 | 3.38 ±0.40 | 33.93 ±5.55 | 4.12 ±0.89 |
| GCN+DropEdge | 32.33 ±3.16 | 19.67 ±1.77 | 31.02 ±2.45 | 20.59 ±1.50 | 32.54 ±1.86 | 17.11 ±1.70 | 56.67 ±4.04 | 22.50 ±1.87 |
| GCN+SSL | 55.72 ±3.34 | 31.55 ±2.20 | 53.70 ±3.13 | 31.20 ±1.41 | 56.36 ±4.69 | 28.76 ±2.27 | 74.35 ±4.19 | 38.70 ±1.17 |
| GCN+SSL+DropEdge | 56.56 ±3.99 | 33.19 ±2.75 | 55.87 ±3.50 | 33.57 ±2.96 | 59.57 ±3.17 | 29.17 ±2.68 | 77.64 ±3.88 | 40.89 ±1.43 |
| HyperGCN | 8.63 ±0.79 | 2.51 ±0.72 | 14.38 ±1.09 | 2.99 ±0.59 | 15.09 ±1.21 | 2.66 ±1.50 | 45.52 ±5.38 | 7.67 ±5.83 |
| HyperGCN+DropEdge | 18.01 ±1.97 | 13.16 ±0.82 | 31.12 ±1.86 | 15.91 ±0.76 | 32.12 ±2.68 | 19.31 ±2.16 | 61.65 ±4.39 | 23.94 ±5.18 |
| HyperGCN+SSL | 63.20 ±0.99 | 41.11 ±0.93 | 68.05 ±0.52 | 47.85 ±1.84 | 71.59 ±0.58 | 50.78 ±0.78 | 79.14 ±0.92 | 55.68 ±1.40 |
| HyperGCN+SSL+DropEdge | 61.35 ±1.50 | 41.62 ±1.70 | 67.31 ±0.19 | 48.42 ±1.64 | 69.59 ±0.46 | 49.31 ±0.70 | 91.82 ±1.72 | 66.29 ±0.09 |
| **DCAH** | 16.07 ±1.38 | 7.54 ±0.65 | 15.03 ±2.72 | 4.20 ±0.47 | 19.56 ±3.99 | 4.22 ±0.67 | 39.54 ±3.82 | 6.09 ±1.99 |
| **DCAH**+DropEdge | 43.66 ±2.41 | 24.76 ±2.35 | 40.95 ±3.82 | 27.24 ±2.94 | 39.87 ±3.50 | 25.88 ±2.91 | 70.34 ±3.46 | 27.22 ±1.78 |
| **DCAH**+SSL | 69.53 ±0.82 | 46.50 ±1.79 | 70.80 ±0.78 | 50.78 ±0.60 | 82.48 ±0.49 | 57.46 ±1.71 | 88.26 ±3.45 | 66.00 ±0.33 |
| **DCAH**+SSL+DropEdge | **72.28** ±1.22 | **48.82** ±1.27 | **73.10** ±0.37 | **52.98** ±0.64 | **84.78** ±0.68 | **59.80** ±1.08 | **93.87** ±3.90 | **69.79** ±0.93 |

## 4.2 Overall Performance Validation (RQ1)

As the experimental results shown in Table 2, 3 and 4, our DCAH consistently outperforms the GCN and HyperGCN across different E-Commerce datasets in terms of all evaluation metrics, whether enhanced by self-supervised graph pre-training and/or DropEdge or not. This observation validates the superiority of our DCAH method, which can be attributed to: 1) By jointly considering the auxiliary hypergraph, DCAH can not only model the query-item relationship, but also preserve the latent high-order item-item similarity relations. 2) Benefiting from the way that the DCAH combine the bipartite graph and the auxiliary hypergraph. Note that for the HyperGCN method, the original bipartite query-item graph is also utilized since the downstream task is still query-item link prediction. The key difference is that message passing is not performed in the original bipartite graph compared with DCAH. And in the single-channel HyperGCN scenario, the query embeddings are query node features obtained using Byte-Pair Encodings [48]. And HyperGCN is only adapted to the auxiliary hypergraph to generate the item embeddings. Although the HyperGCN can only generate the item embeddings, they work well on the long-tail part. This could be due to the fact that the auxiliary hypergraph can provide richer information, which is beneficial for the scarce query-item interactions of the long-tail part. For example, some unpopular items from the original bipartite graph may face the cold-start problem, where the GCN method failed to generate robust embeddings, while these items can be updated during HyperGCN training thanks to the high-order item-item relations of the auxiliary hypergraph. In contrast, the performance of the HyperGCN drops drastically on the head part, which also agrees with our intuition that the hypergraph can also introduce some inevitable noise, *e.g.*, a customer's interest may change while browsing, which can potentially link unrelated items to completely different categories.

In summary, although the auxiliary hypergraph can provide valuable information, it also introduces the inevitable noise, and our model DCAH, a simple yet effective framework, can jointly capture and adaptively fuse the two complementary information.

## 4.3 In Depth Analysis of Hypergraph in Alleviating Data Sparsity (RQ2)

In this section, we easily investigate how the hypergraph helps alleviate the data sparsity. Due to the unique setting of our problem, we

**Table 5: Graph smoothness degrees (measured by MAD) with the encoded query/item embeddings by comparing with the different models.**

| Datasets | Type | GCN | GCN+DropEdge | GCN+SSL | GCN+SSL+DropEdge | HyperGCN | HyperGCN+DropEdge | HyperGCN+SSL | HyperGCN+SSL+DropEdge | DCAH | DCAH+DropEdge | DCAH+SSL | DCAH+SSL+DropEdge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E-Commerce 1 | Query | 0.5911 | 0.6234 | 0.6751 | 0.7298 | 0.5436 | 0.5436 | 0.5436 | 0.5436 | 0.6147 | 0.6385 | 0.7776 | 0.8237 |
| | Item | 0.6438 | 0.6739 | 0.7158 | 0.7439 | 0.6699 | 0.6931 | 0.7396 | 0.7839 | 0.6913 | 0.7246 | 0.7618 | 0.8270 |
| E-Commerce 2 | Query | 0.5737 | 0.6022 | 0.6493 | 0.7029 | 0.5216 | 0.5216 | 0.5216 | 0.5216 | 0.5955 | 0.6147 | 0.7567 | 0.8069 |
| | Item | 0.6273 | 0.6582 | 0.6913 | 0.7254 | 0.6468 | 0.6735 | 0.7197 | 0.7636 | 0.6731 | 0.7064 | 0.7411 | 0.8085 |
| E-Commerce 3 | Query | 0.5552 | 0.5813 | 0.6220 | 0.6872 | 0.5023 | 0.5023 | 0.5023 | 0.5023 | 0.5726 | 0.5938 | 0.7488 | 0.7995 |
| | Item | 0.5999 | 0.6327 | 0.6753 | 0.7083 | 0.6273 | 0.6493 | 0.6983 | 0.7458 | 0.6563 | 0.6883 | 0.7208 | 0.7894 |

partition the items into four different groups by jointly considering their node degrees in bipartite graph and hypergraph. In general, the *head* and *tail1* parts refer to the head part of the original bipartite graph, while the *tail2* and *isolation* parts refer to the long-tail part of the original bipartite graph. From the results shown in Table 2, 3 and 4, we observe that the HyperGCN and DCAH outperform the GCN on the *tail2* and *isolation* parts. Although the HyperGCN only takes the auxiliary hypergraph to generate item embeddings, the superior performance on the long-tail part still shows the potential of the hypergraph in addressing the data sparsity issue. However, due to the inevitable noise of the hypergraph, the performance of the HyperGCN on the *head* and *tail1* part degrades compared to the GCN. Furthermore, in E-Commerce 1 dataset, we find that after adapting the DropEdge strategy, the performance of the HyperGCN on the *head* and *tail1* parts is boosted, while the performance on the *isolation* part drops a little in terms of *MRR*. The possible reason is that the DropEdge may randomly drop some noisy item-item hyperedges, which is beneficial for the *head* and *tail1* parts. While for the *tail2* and *isolation* parts, randomly dropping edges may discard important information of the item-item hyperedges, and these two parts do not contain too much valuable information of the original bipartite graph neither, making the performance worse.

In summary, based on the results, we could conclude that the auxiliary item-item hypergraph has the ability to alleviate the data sparsity issue of the original bipartite graph. Furthermore, if the auxiliary item-item hypergraph is more reliable, *e.g.*, contains less inevitable noise, the benefits will be more. And improving the robustness of the auxiliary item-item hypergraph is our future research direction.

## 4.4 Effect of SSL and DropEdge in Addressing Over-Smoothing (RQ3)

From the Table 1, we find that the three E-Commerce datasets are neither assortative nor disassortative, which indicates that some parts of the original query-item interaction structure are disassortative while the others are assortative. Based on our discussion in Sec 3.2.2, we observe the phenomenon that some popular queries/items may make links with unpopular items/queries. Thus, we make an assumption that the head part of the original bipartite graph is more likely to be disassortative while the long-tail part is more likely to be assortative. Then the over-smoothing issue is more likely to be observed in the head part rather than the long-tail part. And from the results of Table 2, 3 and 4, we notice that the performance of the *head* and *tail1* parts are worse than the *tail2* and *isolation* parts, which validates our assumption.

With the self-supervised graph pre-training and DropEdge, the graph-based over-smoothing effect triggered by the disassortative mixing can be alleviated in our framework. To validate the effectiveness of the two strategies in alleviating the over-smoothing effect, in addition to the superior performance produced by our DCAH, we calculate the Mean Average Distance (MAD) [51] over all node embedding pairs learned by the trained DCAH and other three variants: 1) +DropEdge (with the DropEdge); 2) +SSL (with the self-supervised pre-training); 3) +SSL+DropEdge (with the self-supervised pre-training and DropEdge). For complete comparison, we also calculate the MAD for the other methods, GCN and HyperGCN, with their variants. The quantitative MAD metric measures the smoothness of a graph in terms of its node embeddings. The range of MAD is $[0, 1]$, the more closer to 0 the MAD is, the more similar the node representations are to each other, meaning that all the node representations become indistinguishable. The measurement results are shown in Table 5, where *Query* and *Item* refer to the average similarity score between query nodes and item nodes, respectively. And please note that since the HyperGCN method can only generate the item embeddings, hence, the embedding distance scores of the query nodes are the same.

From the results, we can observe a general trend for the three frameworks: GCN, HyperGCN and DCAH. Take DCAH for example, the DropEdge and SSL can both improve the embedding distance scores, and SSL has better ability at addressing the over-smoothing issue. We guess that is because we use the node-level contrastive loss function in our SSL pre-training, which forces the model to push all the other nodes away from the anchor node, making all the node representations distinguishable.

# 5 Conclusions

In this work, we focus on the query-item prediction problem at E-Commerce stores. Specifically, we discuss the two challenges caused by the original bipartite query-item graph: *long-tail distribution* and *disassortative mixing*. To address these two challenges, we construct an auxiliary item-item hypergraph to augment the original bipartite query-item graph, via using the freely available information from anonymized customer engagement sessions. In the auxiliary hypergraph, the hyperedges contain items present in the same customer sessions. All items engaged by a customer in a single customer session are considered a hyperedge. To tackle the unique setting of the problem, we propose a dual-channel attention-based hypergraph neural network, DCAH, to learn the representations of queries and items by jointly considering the two complementary graphs, the original bipartite graph and the auxiliary hypergraph. Furthermore, we integrate DCAH with self-supervised graph pre-training and the DropEdge training to alleviate disassortative mixing. We evaluate our approach on three randomly sampled anonymized E-Commerce query-item datasets. The experiment results validate the superiority of DCAH. In the future, we may explore more robust methods to construct the hypergraph to reduce the inevitable noise.

## References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
[2] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.
[3] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.
[4] Vincent Leroy, B Barla Cambazoglu, and Francesco Bonchi. Cold start link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 393–402, 2010.
[5] Hao Ding, Yifei Ma, Anoop Deoras, Yuyang Wang, and Hao Wang. Zero-shot recommender systems. *arXiv preprint arXiv:2105.08318*, 2021.
[6] Bijaya Adhikari, Parikshit Sondhi, Wenke Zhang, Mohit Sharma, and B Aditya Prakash. Mining e-commerce query relations using customer interaction networks. In *Proceedings of the 2018 World Wide Web Conference*, pages 1805–1814, 2018.
[7] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[8] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.
[9] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems*, 33:22243–22255, 2020.
[10] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.
[11] Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020.
[12] 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*, volume 35, pages 4503–4511, 2021.
[13] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. Using temporal data for making recommendations. *arXiv preprint arXiv:1301.2320*, 2013.
[14] Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 138–146, 2017.
[15] Wen Wang, Wei Zhang, Shukai Liu, Qi Liu, Bo Zhang, Leyu Lin, and Hongyuan Zha. Beyond clicks: Modeling multi-relational item graph for session-based target behavior prediction. In *Proceedings of The Web Conference 2020*, pages 3056–3062, 2020.
[16] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353, 2019.
[17] Jianling Wang, Kaize Ding, Ziwei Zhu, and James Caverlee. Session-based recommendation with hypergraph attention networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 82–90. SIAM, 2021.
[18] Mahdi Kherad and Amir Jalaly Bidgoly. Recommendation system using a deep learning and graph analysis approach. *arXiv preprint arXiv:2004.08100*, 2020.
[19] Shakila Shaikh, Sheetal Rathi, and Prachi Janrao. Recommendation system in e-commerce websites: A graph based approached. In *2017 IEEE 7th International Advance Computing Conference (IACC)*, pages 931–934. IEEE, 2017.
[20] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.
[21] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.
[22] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.
[23] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.
[24] Yue Gao, Zizhao Zhang, Haojie Lin, Xibin Zhao, Shaoyi Du, and Changqing Zou. Hypergraph learning: Methods and practices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.
[25] Jianling Wang, Kaize Ding, Liangjie Hong, Huan Liu, and James Caverlee. Next-item recommendation with sequential hypergraphs. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1101–1110, 2020.
[26] Shuyi Ji, Yifan Feng, Rongrong Ji, Xibin Zhao, Wanwan Tang, and Yue Gao. Dual channel hypergraph collaborative filtering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2020–2029, 2020.
[27] 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*, pages 413–424, 2021.
[28] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv e-prints*, pages arXiv–1807, 2018.
[29] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced contrastive learning for semi-supervised medical image segmentation with meta-labels. *Advances in Neural Information Processing Systems*, 34, 2021.
[30] Hannan Cao, Wenmian Yang, and Hwee Tou Ng. Grammatical error correction with contrastive learning in low error density domains. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4867–4874, 2021.
[31] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.
[32] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.
[33] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Xiangji Huang. Hypergraph contrastive collaborative filtering. *arXiv preprint arXiv:2204.12200*, 2022.
[34] Hai-Bo Hu and Xiao-Fan Wang. Disassortative mixing in online social networks. *EPL (Europhysics Letters)*, 86(1):18003, 2009.
[35] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer networks*, 33(1-6):309–320, 2000.
[36] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. Heterogeneous graph structure learning for graph neural networks. In *35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021.
[37] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. Am-gcn: Adaptive multi-channel graph convolutional networks. In *Proceedings of the 26th ACM SIGKDD International conference on knowledge discovery & data mining*, pages 1243–1253, 2020.
[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.
[39] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
[40] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820, 2010.
[41] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*, pages 59–66, 2016.
[42] Magdalini Eirinaki, Michalis Vazirgiannis, and Dimitris Kapogiannis. Web path recommendations based on page ranking and markov models. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 2–9, 2005.
[43] Nan Wang, Shoujin Wang, Yan Wang, Quan Z Sheng, and Mehmet A Orgun. Exploiting intra-and inter-session dependencies for session-based recommendations. *World Wide Web*, 25(1):425–443, 2022.
[44] Longcan Wu, Daling Wang, Kaisong Song, Shi Feng, Yifei Zhang, and Ge Yu. Dual-view hypergraph neural networks for attributed graph learning. *Knowledge-Based Systems*, 227:107185, 2021.
[45] Alain Bretto. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer*, 2013.
[46] Mark EJ Newman. Mixing patterns in networks. *Physical review E*, 67(2):026126, 2003.
[47] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.
[48] Benjamin Heinzerling and Michael Strube. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12, 2018 2018. European Language Resources Association (ELRA).
[49] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020.
[50] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.
[51] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.