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  
nikhilrs@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 Dual-Channel Attention-Based Hypergraph 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
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 Dual-Channel Attention-Based Hypergraph 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]. GCMI [20] utilized neural graph autoencoders to reconstruct a user-item rating
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 \( Q = \{q_1, q_2, q_3, \ldots, q_{N_Q}\} \) and \( I = \{i_1, i_2, i_3, \ldots, i_{N_I}\} \) 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 \( \{i_{r,1}, i_{r,2}, i_{r,3}, \ldots, i_{r,m}\} \), where \( i_{r,k} \in \{1 \leq k \leq m\} \) represents an interaction between an anonymous user and an item within the browse/search record \( r \).

**Hypergraph Definition.** A hypergraph is defined as \( G = (\mathcal{V}, \mathcal{E}) \), 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 \( H \in \mathbb{R}^{N \times M} \), 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 \( D_{ii} = \sum_{e \in \mathcal{E}} H_{ie} \) and \( B_{ee} = \sum_{i \in \mathcal{V}} H_{ie} \), respectively. \( D \) and \( B \) are diagonal matrices. \( D_{ii} \) and \( B_{ee} \) are the \( i \)-th and \( e \)-th diagonal elements of \( D \) and \( B \), respectively.

**Hypergraph Construction.** To capture relationships beyond pairwise ones in the browse/search records, we adopt a hypergraph \( G = (\mathcal{V}, \mathcal{E}) \) to represent each record as a hyperedge. Formally, we
As a type of real network, query-item graphs also tend to have a
majority of nodes have fewer than ten neighbors. In summary, the
degree distribution of the E-Commerce query-item graph follows
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,
even 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 aux-
iliary 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 de-
pendencies [43, 44] such as those in customer search sessions. Hy-
pergraphs [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 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,
even 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.
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, $X_I^{(0)}$. For the query node, the raw text of the user query is available to generate node features using Byte-Pair Encodings [48], denoted as $X_Q^{(0)}$. The adjacency matrix for this bipartite graph $\hat{A}$ is defined as $\hat{A} = A + I$, where $I$ is the identity matrix. $D \in \mathbb{R}^{N \times N}$ is a diagonal degree matrix where $D_{p,p} = \sum_{q=1}^{n} \hat{A}_{p,q}$. The bipartite graph convolution is then defined as:

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

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 $X$ can capture the pairwise query-item relation information. We then pass $X^{(0)}$ through $L$ graph convolutional layers to obtain the final output embeddings $X^{(L)}$. We use $X_Q^{(L)}$ and $X_I^{(L)}$ 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 $X_I^{(0)}$. Following the spectral hypergraph convolution proposed in [23], we define our hypergraph convolution as:

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

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 $H^T$ 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 $X^{(0)}_{HG}$ through $L$ hypergraph convolutional layers, we obtain the final item embeddings $X_I^{(L)}_{HG}$.

Inter-Channel Aggregation. With the dual-channel structure, we generate two groups of item embeddings, $X^{(L)}_{BG}$ and $X^{(L)}_{HG}$, 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 $q$. Finally, we average the importance of all channel-specific item embeddings. We can interpret this as the importance of each channel, denoted as $w_{\phi_c}$.

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

Here, $\mathbf{W}$ is the weight matrix, $b$ is the bias vector, $\mathbf{x}_i \phi_c$ is the channel-level embedding, $\text{tanh}(\cdot)$ is the activation function, $q$ is the channel-level attention vector, and $V$ 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:

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

$\beta_{\phi_c}$ 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 $X_J^{(L)}$ as follows:

$$X_J^{(L)} = \beta_{\phi_{BG}} \cdot X_I^{(L)}_{BG} + \beta_{\phi_{HG}} \cdot X_I^{(L)}_{HG}$$  

3.4 Enhancing DCAH with Self-Supervised Learning and DropEdge

Although the auxiliary DCAH with Self-Supervised Learning and DropEdge
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 $\hat{G}_B$ and the hypergraph $\hat{G}_H$. Although the hypergraph is different from the bipartite graph, the representation formats are similar. Therefore, both given graphs $\{\hat{G}_B, \hat{G}_H\}$ can undergo graph data augmentations proposed in [32] to obtain two correlated views, $\{\hat{G}_B, \hat{G}_H\}$, $\{\hat{G}_B, \hat{G}_H\}$, as a positive pair. Then, our model DCAH extracts node-level embeddings $h_i$ and $h_j$ for two augmented graph pairs $\{\hat{G}_B, \hat{G}_H\}$ and $\{\hat{G}_B, \hat{G}_H\}$, 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:

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

where $\text{sim}(z_{n,i}, z_{n,j}) = z_{n,i}^T z_{n,j}/||z_{n,i}|| ||z_{n,j}||$, $z_{n,i}$, $\tau$ 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 hypergraphs, 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.

| Datasets 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| 0.152       | 0.089       | -0.07       |
| Density             | 1.28 × 10^{-6} | 1.45 × 10^{-5} | 1.14 × 10^{-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.
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 Hyperparameter 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.

#### 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...
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 closer to 0 the MAD is, the more similar 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 experimental 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:1706.02243, 2017.

[3] Muhammad Zafar and Yunhan 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, Yide 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] Qinmai Li, Zhihao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[8] Yu Rong, Wenhong 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 semi-supervised models are strong semi-supervised learners. Advances in neural information processing systems, 33:2243–2255, 2020.

[10] Chen Cai and Yuesheng Wang. A note on over-smoothing for graph neural networks. arXiv preprint arXiv:2006.13318, 2020.

[11] Wenhong 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 Wang, Li Zhen 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 4505–4511, 2021.

[13] Andrew Zimdars, David Maxwell Chichkerov, and Christopher Meek. Using temporal data for making recommendations. arXiv preprint arXiv:1301.2502, 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, Leyun Lin, and Hongyun Yuan. 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 Zhi, Liang Wang, Xing Xie, and Tienniu 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, Zhiwei Cui, and James Caverlee. Session-based recommendation with graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33:22118–22133, 2020.

[18] Shakila Shaikh, Sheetal Rathi, and Prachi Janrao. Recommendation system in the web. Computer networks, 33(1-6):309–320, 2000.

[19] Steffen Rendle, Butler2017RendleRB: Bilinear factorization for sparse predictive inference., 2017.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

[21] Magdalini Eirinaki, Michalis Vazirgiannis, and Dimitris Kepouyiannis. 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.

[22] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[23] Vincent Leroy, B Barla Cambazoglu, and Francesco Bonchi. Self-supervised hypergraph convolutional networks for session-based recommendation. In Proceedings of The Web Conference 2020, pages 1120–1130, 2020.

[24] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization machines with deep embeddings. In Proceedings of the AAAI conference on artificial intelligence, pages 2501–2507, 2017.

[25] 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.

[26] Andrew Zimdars, David Maxwell Chichkerov, and Christopher Meek. Using temporal data for making recommendations. arXiv preprint arXiv:1301.2502, 2013.

[27] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.

[28] Deli Chen, Yankai Lin, Wei Li, Peng Cui, Chuan Shi, and Jian Pei. Am-gen: Adaptive multi-channel graph convolutional networks. In Proceedings of the 36th ACM SIGKDD International Conference on Knowledge discovery & data mining, pages 1243–1253, 2020.

[29] Jian Tang, Meng Qu, Mingze 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.

[30] Lu Li, Farid Ablayev, Qiang Liu, and J. Zico Kolter. Group normalization. In Thirty-third AAAI Conference on Artificial Intelligence, 2019.

[31] Qimai Li, Desheng Zhang, and Zhihong Yin. Deep learning and graph analysis approach. SIAM International Conference on Data Mining (SDM), 2020.

[32] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[33] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[34] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[35] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[36] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[37] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.

[38] Jizong Peng, Ping Wang, Christian Desrosiers, and Marco Pedersoli. Self-paced over-smoothing for general graph convolutional networks. In Proceedings of The Web Conference 2020, pages 1067–1077, 2015.