Sequence-Graph Duality: Unifying User Modeling with Self-Attention for Sequential Recommendation

Anonymous Author(s)
Affiliation
Address
email

Abstract

User modeling is of great importance in personalization services. Many existing methods treat users as sequences of interacted items to encode item transition patterns. Another line of research models users as graphs in which interacted items are modelled as nodes, and item-item relations are modelled as edges. The graph-based user modeling provides more flexibility to encode complex item relationships of different types (e.g. co-occurrence, similarity) but usually overlooks sequential patterns. Here, we introduce a novel user representation, Heterogeneous User Graph (HUG), which unifies sequence and graph-based user modeling to take advantage of both methods. An HUG is associated with two types of edges: *sequential edges* that preserve the sequential information and *collaborative edges* that store different item-item relationship. To learn users' latent representations from their HUGs, we propose a multi-head attention based architecture called Heterogeneous User Graph Transformer (HUGT). HUGT is developed on the basis of SASRec and can concurrently capture the sequential transition pattern and complex graph topology. We conduct experiments on four real world datasets from three different application domains. Experimental results show that (1) jointly modeling users as sequences and graphs with HUG provides better recommendation performance over sequence-only and graph-only user modeling; (2) HUGT is effective in learning user latent representations from HUGs; (3) HUGT outperforms the baselines by up to 10% on datasets with long sequences and aligns with the state-of-the-art performance on datasets with short sequences.

1 Introduction

As internet services grow fast, users are facing a vast amount of information and products in their daily life. Recommender systems play a critical role in filtering information and presenting relevant content that matches users' interests. Due to the dynamic nature and complexity of users' interests, learning patterns in users' interaction history and encoding them into the users' latent representations for recommendation have been a prevalent research topic.

There have been a surge of methods that model users by their sequences of item interactions and apply sequential models such as RNNs [8] and Transformers [20] to learn user representations [14, 7, 11, 18, 3]. These methods demonstrated that encoding item transition patterns in users' latent representations helps capture users' evolving interests and improves recommendation accuracy. Another line of research advocates modeling each users as a graph in which the user's interacted items are modelled as nodes, and item-item relations (such as frequent co-occurrence, consecutive clicks) are modelled as edges [25, 1, 23]. The initial features of the same items are shared across graphs that contains the items. These methods usually apply graph neural networks (GNNs) on each

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

user's graph to encode relationships among the items into latent item embeddings and use a readout module (e.g., average) to convert the item embeddings to a latent user representation.

In this work, we propose a method to link the two worlds by jointly modelling users as sequences and graphs to encode both sequential patterns and graph topology. An HUG is a graph of user interacted items, in which nodes are connected by two types of edges: *sequential edges* that preserve the sequential ordering of items (e.g., chronological precedence) and *collaborative edges* that characterize item-item relationships such as collaborative similarity. On the basis of SASRec [11], we design a transformer model to concurrently fuse and encode the sequential patterns and the graph topology into users' latent representations for recommendation. We call our method Heterogeneous User Graph Transformers (HUGT).

We summarize our contributions as the following: First, we unify the sequence modeling and the graph modeling of users with HUG, which provides a flexible framework of injecting inductive bias into self-attention-based recommendation model. Secondly, we design HUGT to effeciently learn user representations from their HUGs. We propose two variants of HUGT: HUGT-SE which models graph structure with spatial encoding, and HUGT-MP which computes a sparse attention matrix to simulate message passing in GNNs. Finally, we conduct comprehensive experiments on four real-world datasets from three different domains. We show that jointly modeling users as sequences and graphs via HUGs outperforms standalone sequence modeling or graph modeling of the users, HUGT is effective in concurrently fusing and encoding the sequence and graph aspects of HUG, and HUGT achieves state-of-the-art performance on all the four datasets.

2 Related Works

2.1 Sequence-based User Modeling

Sequential signal is important in user modeling and can improve recommendation performance. Traditional sequential recommendation models [16, 5, 19] usually rely on factorization methods (e.g., Factorized Item Similarity Models [10]) to capture users' general tastes and Markov Chains (MCs) to learn the sequential signal. This class of methods hold the Markov assumption that the next action is dependent on the previous few interactions and cannot capture complex and long sequential patterns. Moreover, the performance of these methods are limited by the capacity of the factorization models.

In the past decade, researchers developed RNN-based approaches to model users' interaction trajectories. Hidasi et al. [7] proposed GRU4Rec that leverages a variants of RNNs, gated recurrent units (GRU), to capture sequential patterns in sessions. Ma et al. [14] successfully built real world commercial recommender systems based on hierarchical RNNs. The authors of [27] combined RNNs with CNNs to capture long and short term sequential patterns, respectively. Despite the success stories, RNN-based recommendation models suffer from data efficiency (they require a lot of training data to perform well) and data sparsity issues.

2.2 Graph-based User Modeling

Graph-based user modeling is a rising direction that models users as graphs in which their interacted items are nodes. Nodes in these graphs are usually connected via edges that indicate transition/similarity relations [25, 1, 23]. Methods that fall into this class usually use graph neural networks (GNNs) to encode the graph structured information into item embeddings and use a learnable function to readout user embeddings for recommendation. Although this class of methods captures graph structured information, they usually overlook sequential patterns that are important for recommendation.

2.3 Transformer and Self-Attention Mechanism

Self-attention mechanism and the Transformer architecture have demonstrated their superior ability for sequence modeling and have become the default methodology in natural language processing tasks [20, 4]. In recommendation community, self-attention mechanism is widely adopted for sequential user modeling [11, 18, 24, 9, 13]. He et al [6] propose to augment BERT4Rec by enforcing items in sequences to attend to items that are sequentially close to reinforce the influence of local items while Chen et al [2] propose to learn a sparse mask over the attention matrix to reduce the influence of

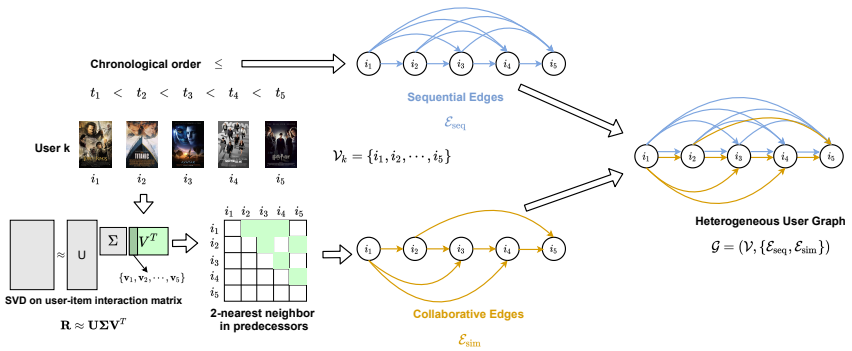

Figure 1: Construction of a heterogeneous user graph from a user's interaction sequence.

noisy items. In our work, we propose a formulation that allows more flexible design of inductive bias in self-attention-based recommendation models.

Recently, there has been discussions that connect Transformers with GNNs. In [22], the authors show that self-attention mechanism is a special form of graph convolutions. The authors of [28] generalizes the definition of self-attention by directed graphs. Ying et al. [26] successfully apply transformers on graph prediction tasks and outperform major GNN variants by augmenting transformers with graph structural knowledge. These works motivate us to use Transformer as the backbone for to concurrently learn sequential patterns and graph topology in users' behaviours.

3 Heterogeneous User Graph

To unify the sequence representation and the graph representation of a user, we first introduce a data structure, heterogeneous user graph, that has the capacity to store both sequence transition patterns and graph topology.

3.1 Graph Construction

We construct a heterogeneous user graph (HUG) $\mathcal{G} = (\mathcal{V}, \{\mathcal{E}_{\text{seq}}, \mathcal{E}_{\text{sim}}\})$ from each user's interaction sequence. We define the nodes $\mathcal{V}$ in the HUG as the interacted items in the sequence. With the heterogeneity of HUGs, there could be various types of edges to encode different transition patterns. In this paper, we construct two types of edges, sequential edges $\mathcal{E}_{\text{seq}}$ and collaborative edges $\mathcal{E}_{\text{sim}}$, to capture sequential signal and collaborative similarity, respectively. Collaborative similarity measures the similarity between two items in terms of the two sets of users that interacted with them. We show the construction process in Figure 1. Note that the collaborative edge can be easily generalized to capture other types of item-item relation as well.

Sequential Edges. The sequential edges are to store the chronological order of each pair of items. For $i, j \in \mathcal{V}$, $(i, j) \in \mathcal{E}_{\text{seq}}$ if $t_i < t_j$ where t_i denotes the position of i in the sequence.

Collaborative Edges. Collaborative edges are to encode collaborative similarity, a measure of similarity in regards to the users' preference patterns on the items. In order to measure collaborative similarities among items, we first conduct a truncated singular value decomposition (SVD) on the user-item interaction matrix $\mathbf{R} \in \{0, 1\}^{|\mathcal{U}| \times |\mathcal{I}|}$ to obtain a set of low-dimensional item embeddings $\mathbf{V} \in \mathcal{R}^{d \times |\mathcal{I}|}$. We lookup the embedding of each item $i \in \mathcal{V}$ and compute its cosine similarity with all its predecessors, i.e., $\{j | j \in \mathcal{V}, t_j \leq t_i\}$. Each item is connected to its k-nearest predecessors.

4 Heterogeneous User Graph Transformer

In this section, we present heterogeneous user graph transformer (HUGT) that operates on the HUG to concurrently encode sequential signal and graph topology into latent user representations. HUGT is built on the basis of a multi-head attention user modeling framework. We divide the multiple attention heads into two groups, sequential modeling heads and graph modeling heads, to encode the two types of item-item relations, respectively. We encode the sequential information via relative

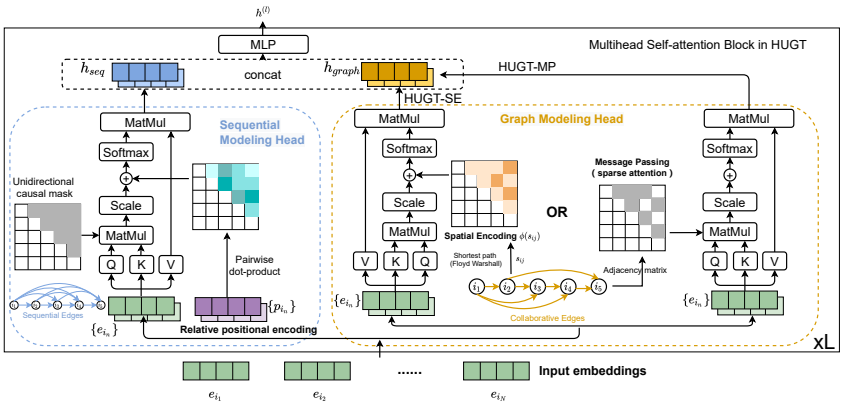

Figure 2: Computation flow of Heterogeneous User Graph Transformer (HUGT).

positional encoding. We propose two strategies, Message Passing (HUGT-MP) and Spatial Encoding (HUGT-SE), to learn graph topology stored in the collaborative edges. The message passing strategy iteratively aggregates information from each node's graph neighbor while the spatial encoding method pre-compute structural information from the graph and learn to leverage this information into attention scores. Both groups of attention heads utilize the generalized self-attention mechanism.

4.1 Generalized Self-Attention Mechanism

In the standard self-attention, item at each position aggregates weighted information from all items in the sequence to selectively build the context. Generalized self-attention (GSA) mechanism generalizes the standard self-attention by allowing each item to attend to any subset of the positions in the sequence and enlarges the design space of self-attention.

Let $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n] \in \mathcal{R}^{d \times n}$ be the latent representation of a sequence $\mathcal{S}$ from previous layer, where n and d denotes the length of the sequence and the dimension, respectively. GSA computes the representation of an item i as

$$\text{GSA}(\mathbf{h}_i) = \sum_{j \in \mathcal{N}(i)} \frac{\exp(\pi_{ij})}{\sum_{j' \in \mathcal{N}(i)} \exp(\pi_{ij'})} \mathbf{v}_j \tag{1}$$

$$\mathbf{q} = \mathbf{W}_q \mathbf{h}, \quad \mathbf{k} = \mathbf{W}_k \mathbf{h}, \quad \mathbf{v} = \mathbf{W}_v \mathbf{h}, \text{ and } \pi_{ij} = \mathbf{q}_i^T \mathbf{k}_j / \sqrt{d},$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are learnable parameters that map the input embedding to query, key, and value domain, respectively. $\mathcal{N}(i)$ denotes the set of items that item i attends to. When $\mathcal{N}(i) = \mathcal{S}$, the GSA fully recovers the standard self-attention. In unidirectional attention with causal masking (i.e., each item can only attend to its precedents) used in SASRec [11], $\mathcal{N}(i) = \{j | t_j < t_i\}$.

4.2 User Modeling with Multi-head Self-Attention

Embedding Layer. We use a learnable embedding table $\mathbf{E} \in \mathcal{R}^{d \times |\mathcal{I}|}$ to convert discrete item ids $[i_1, i_2, \cdots, i_n]$ to a matrix representation, $\mathbf{X} = [\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \cdots, \mathbf{e}_{i_n}] \in \mathcal{R}^{d \times n}$ where d is the dimensionality of the embeddings and n is the maximum sequence length that the model can take. The latent representation is fed into a stack of self-attention blocks to generate user embeddings.

Self-Attention Block. A self-attention block (SAB) consists of a generalized self-attention (GSA) layer that is introduced in Section 4.1 and a feed-forward layer (MLP) that adds non-linearity and considers interactions between latent dimensions, i.e.,

$$\text{MLP}(\mathbf{h}_i) = \mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{h}_i + \mathbf{b}_1) + \mathbf{b}_2, \tag{2}$$

A SAB takes the output from the previous SAB as input, and passes the result to the next block, i.e.,

$$\mathbf{h}_i^{(l+1)} = \text{SAB}(\mathbf{h}_i^{(l)}) = \text{MLP}(\text{GSA}(\mathbf{h}_i^{(l)})), \tag{3}$$

where $\mathbf{h}_i^{(l)}$ denotes the output of the l-th SAB block, and $\mathbf{h}_i^{(0)} = \mathbf{e}_i$ is the output of the initial embedding layer. Following [11], we use layer normalization to accelerate training and residual connections to preserve low-level information. We stack L SABs and use the output embedding at the last position of the last SAB as the final user representation for recommendation, i.e., $\mathbf{h}_u = \mathbf{h}_n^{(L)}$.

4.3 Sequential Modeling with Relative Positional Encoding

In contrast to RNNs that rely on recurrent operations to capture sequential signals, Transformers use positional encoding to break the order equivariance. We use an embedding table $\mathbf{P} \in \mathcal{R}^{d \times n}$ to map the positions to continuous representations $[\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n]$. The positional embedding table could be either learnbale or pre-computed by a set of sinusoid functions. In this paper we follow [11] and use learnable positional embeddings.

There are two major ways to incorporate the positional embeddings into self-attention, adding them to the input item features at the corresponding positions (absolute positional encoding) [20] and adding a relative score computed by the two embeddings of the corresponding positions to the attention score computed in Equation 1 (relative positional encoding) [17], i.e.,

$$\pi_{ij} = \frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d}} + \psi(\mathbf{p}_{t_i}, \mathbf{p}_{t_j}),$$

where $\psi(\cdot, \cdot)$ is a learnable function. Empirically, we found that relative positional encoding outperforms absolute positional embedding in sequential recommendation tasks.

4.4 Graph Modeling

Message Passing. Graph neural networks that rely on message passing along graph edges have shown their ability to encode graph topology into latent node/graph embeddings. Note that, Equation 1 could be viewed as a message passing operation on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{\text{sim}})$ by setting $i \in \mathcal{V}$ and $\mathcal{N}(i) = \{j | (i, j) \in \mathcal{E}_{\text{sim}}\}$. In fact, Equation 1 is a special implementation of graph attention networks (GAT). We skip the details of GAT and refer the readers to [21] for more information.

In this paper, we directly apply GSA on the collaborative edges in HUGs as one way to encode graph structural information. With such formulation, each node adaptively aggregates information from its graph neighbors and updates its representation. The stacked SABs allows nodes to accumulate information that are several hops away and encode graph topology into their latent representations.

Graph Spatial Encoding. The standard self-attention allows each item to attend to all items in the sequence. This brings the advantage of a global receptive field, but also raises the challenge of encoding graph structural information into item representations. We tackle this challenge by pre-extracting graph related properties and learning to incorporate the properties into the computation of self-attention.

In this paper, we first compute the length of the shortest-path $s_{ij} \in \mathcal{Z}^+$ between each pair of nodes $i, j \in \mathcal{V}$ via collaborative edges to measure the spatial distance between the two nodes. We encode this information via a learnable mapping $\phi : \mathcal{Z}^+ \mapsto \mathcal{R}$ that converts the distance to a real scalar $\phi(s_{ij})$ and directly add this value to the attention base, i.e., $\pi_{ij} = \mathbf{q}_i^T \mathbf{k}_j / \sqrt{d} + \phi(s_{ij})$. We clip the shortest-path length to an upper bound S, i.e., length larger than S will be set to S. The clipping enables the model to generalize to unseen lengths during training, controls the model size, and provides encoding for disconnected item pairs.

4.5 Information Fusion with Multi-head Attention

Recall that in Equation 1, the input matrix $\mathbf{H}$ to each layer is mapped to a key $\mathbf{K}$, query $\mathbf{Q}$, and value $\mathbf{V}$ domain for the computation of self-attention. In a multi-head self-attention mechanism, $\mathbf{H}$ is mapped by multiple sets of projection matrices to several key-query-value spaces (each space corresponds to one attention head) to compute self-attention in parallel. The resulting representation from all attention heads are concatenated and projected to be the final representation, i.e.,

$$\mathbf{h} = \mathbf{W}_o \text{Concat}\left(\mathbf{h}^1, \mathbf{h}^2, \cdots, \mathbf{h}^m\right), \tag{4}$$

where m is the number of heads, $\mathbf{W}_o \in \mathcal{R}^{d\times(d\times m)}$ is a projection matrix, $\mathbf{h}^i$ is the output of the i-th attention head.

The multi-head formulation allows each item to aggregate information from different attention paths in one attention block. This provides flexibility to cope with the heterogeneity of HUGs. In each attention block, We separate the attention heads into two groups, the sequential modeling heads that learn from the sequential edges and the graph modeling heads which learn from the collaborative edges. The two groups of attention heads adopt different mechanisms to capture the two sources of information. By stacking such multi-head attention blocks, we can effectively fuse and encode information from the two types of edges into user representations.

4.6 Prediction and Model Training

With the learned user representation at position t, i.e. $\mathbf{h}_u^t$, the final prediction $\hat{y}_{ui}^t$ of an item i is computed by

$$\hat{y}_{ui}^t = \mathbf{w}_i^T \mathbf{h}_u^t + b_i, \tag{5}$$

where $\mathbf{w}_i$ and b_i are decoder weight and bias for item i. We notice that the model generalizes better by sharing weights of the input embedding table with the item decoder.

We train the model by optimizing the cross-entropy loss

$$\mathcal{L} = -\sum_{u\in\mathcal{B}} \sum_{1\leq t\leq n} \log\left(\frac{\exp\left(\hat{y}_{ui}^t\right)}{\sum_{j\in\mathcal{C}_u^t}\exp\left(\hat{y}_{uj}^t\right)}\right), \tag{6}$$

where $\mathcal{B}$ is a batch of users and $\mathcal{C}_u^t$ is a set of candidate items for user u at t. n is the maximum length of the sequence.

4.7 Complexity Analysis

Let $\mathcal{G}$ be an user HUG with n nodes. The time and space complexity of a m-head HUGT is the same with the standard Transformer, i.e., $\mathcal{O}(mn^2)$. In the pre-processing steps, we apply a truncated SVD on a sparse user-item interaction matrix which costs $\mathcal{O}((|\mathcal{U}| + |\mathcal{I}|)d^2 + Td)$ where T is the cost of a sparse matrix-vector multiplication and d is the dimension. When applying HUGT-SE, pre-computing the shortest path between each pair of nodes costs $\mathcal{O}(n^3)$. Since these steps only need to be applied once and could be computed in parallel, they are not efficiency barrier to our method.

5 Experiments

In this section, we carefully design experiments to investigate three research questions in regards of the heterogeneous user graph (HUG) modeling of users and the heterogeneous user graph transformers (HUGT):

- **RQ1**: How does HUGT perform in sequential recommendation compared to the state-of-the-art methods?
- **RQ2**: Is jointly modeling users as graphs and sequences better than a standalone sequence/graph user model?
- **RQ3**: Is a concurrent sequence-graph model better than modeling a user as graph first and then a sequence?

5.1 Experimental Setup

5.1.1 Datasets

We experiment with four benchmark datasets, **ML1M**, **Netflix**, **Steam**, **Taobao**, that fall into three different application domains, movie recommendation, game recommendation, and E-commerce. The datasets are different in regards their size, shape, density, and sequence length. The detailed statistics of the datasets are shown in Table 1. For all the datasets, we convert the ratings/reviews/interactions to implicit feedback (i.e., existing entries are assumed positive while missing entries are treated as negative signal).

Table 1: Statistics of the datasets

Dataset	# Users	# Items	# Interactions	Density	Avg. Len.
ML1M	6040	3398	997923	4.86%	165.11
Netflix	376076	17673	31324749	0.47%	83.29
Steam	1198725	13055	6419630	0.04%	5.36
Taobao	40023	205495	3933299	0.05%	98.28

5.1.2 Baselines and Implementation Details

We compare the performance of HUGT with state-of-the-art sequential recommendation baselines that fall into the categories of factorization-based methods (**Fossil** [5]), RNN-based methods (**GRU4Rec** [7]), GNN-based methods (**SR-GNN** [25]), and Self-Attention-based methods (**FISSA** [13], **BERT4Rec** [18], and **SASRec** [11]).

For Fossil [1] and FISSA [2], we use the implementation associated with the original paper. For GRU4Rec, we use the implementation from [14], and we use an PyTorch implementation of BERT4Rec[3]. We implement SASRec using PyTorch an we find that our implementation greatly outperforms the original SASRec implementation. In our experiments, we select the hidden dimensions from $\{50, 300\}$. For Markov chain(MC)-based methods, we consider the MC order from $\{1, 2, 3, 4, 5\}$. For attention-related methods such as SASRec and BERT4Rec, we choose the number of attention heads from $1, 2, 3, 4$. For batch size and optimization related hyper-parameters, we follow the instructions from the original paper if provided. Otherwise, we set the batch size to 128 and optimize the model using the Adam [12] optimizer with a learning rate of 0.001.

We implement our HUGT model using PyTorch[15]. The maximum length for each dataset is set to be able to cover the average sequence lengths of the dataset. We choose 200 for ML1M, 100 for Netflix and Taobao, and 50 for Steam. For constructing collaborative edges in HUG, we consider the k value from $\{5, 10, 20, 30, 50, 70\}$ for ML1M, Netflix and Taobao. For the Steam dataset, we only consider $\{2, 3\}$ since its average sequence length is 5. When computing the ranking objective in Equation 6, we use all the items as candidate items for ML1M, Netflix, and Steam. Since the Taobao dataset has a large number of items, we compute the loss on a randomly selected $0.5\sqrt{|\mathcal{I}|}$ candidate items to reduce GPU memory consumption. We use the Adam algorithm to optimize the parameters of HUGT with a learning rate of 0.001. We set the batch size for ML1M, Netflix, Steam to 128, and Taobao to 32. In our experiments, we set the number of attention heads to be 4 and consider different combinations of sequence heads and graph heads.

5.2 Experimental Settings

We use the first 90% of the interactions of each user as training set and the last 10% of the interactions for evaluation purpose. We follow an auto-regressive manner in evaluating the recommendation models, i.e., we compute a user's state/representation from her current interaction sequence, make a prediction (a ranked list of items) for the next interaction, evaluate the recommendation list with the ground truth item, append the ground truth item to the tail of her sequence, and update the user's state/representation for next prediction. We validate on 30% of the users for model selection and early stopping and report test performance on the remaining 70% of the users.

We train HUGT and all the baselines on the training set and evaluate their performance on the validation set after each training epoch. We stop training if the validation score does not decrease in ten epochs or the number of training epochs reaches 100. We report the test performance of the checkpoint that performs the best on the validation set.

5.2.1 Evaluation Metrics

In this paper, we use HR@k and NDCG@k to evaluate the performance of the models. HR@k and NDCG@k compute the proportion of cases when the ground truth item is in the top-k prediction

[1] https://drive.google.com/file/d/0B9Ck8jw-TZUEeEhSWXU2WWloc0k/view?usp=sharing

[2] http://csse.szu.edu.cn/staff/panwk/publications/Code-FISSA.zip

[3] https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch

Table 2: Performance comparison of the two HUGT variants, Message Passing (MP) and Spatial Encoding (SE), against baselines on the four datasets. In each row, we highlight the best performance as boldface and underline the second best performance.

	ML1M		Netflix		Steam		Taobao	
Metric	HR@10	N@10	HR@10	N@10	HR@10	N@10	HR@10	N@10
Pop	0.0117	0.0054	0.0197	0.0112	0.1029	0.0565	0.0024	0.0012
Fossil	0.0778	0.0314	0.0639	0.0258	0.2253	0.095	0.0027	0.001
GRU4Rec	0.2112	0.1164	0.2684	0.1615	0.1459	0.0811	0.1559	0.0942
SR-GNN	0.1516	0.0819	0.2253	0.1302	0.1059	0.0562	0.0921	0.0573
SASRec	0.1432	0.0599	0.2154	0.1143	0.2715	0.1985	0.1046	0.064
FISSA	0.0911	0.0387	0.1405	0.0711	0.1665	0.0899	-	-
BERT4Rec	0.153	0.0764	0.2462	0.1431	0.2979	0.2504	0.1342	0.0805
SASRec-Ours	0.2052	0.1039	0.3267	0.2028	0.3027	0.2581	0.1612	0.095
HUGT-MP	**0.2233**	**0.1218**	0.3286	0.2043	**0.3028**	**0.2585**	**0.1734**	0.1032
HUGT-SP	0.2225	0.1191	**0.3333**	**0.2085**	0.3022	0.2573	0.172	**0.1048**
Improv.	5.73%	4.64%	2.02%	2.81%	0.03%	0.15%	7.57%	10.32%

list. HR equally weighs the hit items while NDCG weighs the hit items decreasingly by their corresponding rank in the top-k list.

5.3 Recommendation Performance (RQ1)

We compare recommendation performance of the proposed HUGT with the state-of-the-art sequential recommendation methods on the four benchmark datasets. The overall performance is shown in Table 2. We observe that our proposed HUGT significantly outperforms all the baselines on ML1M, Netflix, and Taobao while its performance on Steam is similar to SASRec. This provides evidence that the collaborative graph topology is helpful for user modeling and our HUGT can effectively capture this signal to improve recommendation performance. Note that, the ML1M, Netflix, and Taobao dataset have an average sequence length of 165, 83, and 98, respectively, while the average length of Steam is 5. This shows that our proposed method works well in settings where the interaction sequences are long. When the sequences are in general short, our method performs on par with SASRec.

Table 3: Necessity of concurrent modeling of sequences and graphs. G-S denotes the model architecture that asynchronously encodes graph topology then sequential signal. S-G denotes the architecture that reverses the above order. Concur is the default HUGT that learns sequential signal and graph topology concurrently.

Dataset	Model	G-S	S-G	Concur
ML1M	HR@10	0.2179	0.2136	**0.2206**
	NDCG@10	0.1184	0.1158	**0.1196**
Netflix	HR@10	0.3217	0.3219	**0.3286**
	NDCG@10	0.1991	0.1991	**0.2043**
Steam	HR@10	0.3020	0.3023	0.3020
	NDCG@10	0.2578	0.2580	0.2582
Taobao	HR@10	0.1679	0.1503	**0.1734**
	NDCG@10	0.1007	0.0904	**0.1032**

We observe that the two graph learning schemes, message passing (MP) and spatial encoding (SE), show advantages in different datasets. This demonstrates that some datasets could benefit from the global receptive field of the SE method while others may enjoy the local structure of the MP scheme.

5.4 HUG vs. Standalone User Modeling (RQ2)

In this section, we design experiments to investigate the necessity of the unified sequence-graph user representation and the graph learning scheme. For each dataset, we train five HUGT-MP models that each has four attention heads. The five HUGT-MP models have a number of sequential heads ranging from zero to four while rest of the heads are graph heads. We show the recommendation performance of the models in Figure 3. We observe a performance drop on all the datasets when we set the number of sequence heads to be zero. This shows the importance of sequence pattern in user modeling. On

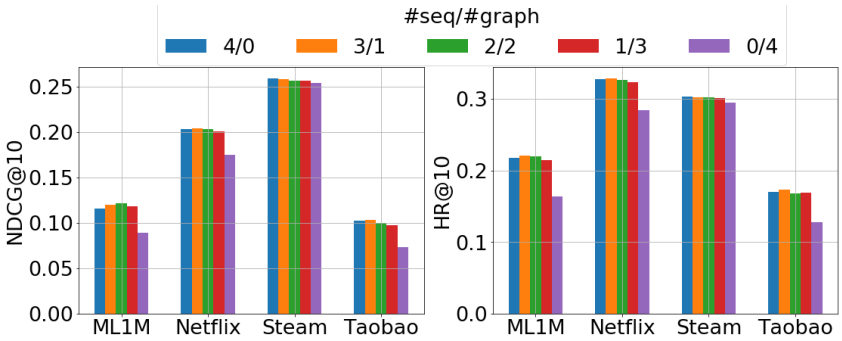

Figure 3: Recommendation performance with different combinations of sequential heads and graph heads on ML1M.

the three datasets with long sequences (ML1M, Netflix, Taobao), we observe a performance drop when we set the number of graph heads to zero. This demonstrates that the collaborative graph topology is important in a long sequence setting and the graph learning schemes that we proposed is effective in capturing this information. On the Steam dataset that has short sequence lengths, the recommendation performance is not influenced by cutting the graph heads. This suggests that the transition patterns in short sequence settings are not very sophisticated and could be very well captured by naive sequential model.

5.5 Importance of Concurrent Modeling (RQ3)

In a HUGT block, to incorporate the heterogeneity of HUG, we split multiple attention heads to sequence heads and graph heads to concurrently encode sequential pattern and graph topology. In this section, we conduct experiments to answer the question that whether this formulation is better than its asynchronous alternatives. We design two architectures that process the sequence heads and the graph heads sequentially. One that applies an HUGT with only graph heads on the HUG to encode only graph topology into the item embeddings, then applies an HUGT with only sequence heads on the HUG whose initial node embeddings are the output of the previous graph-only HUGT to mix graph topology and sequential patterns. The second architecture is similar but applies a sequentce-only HUGT first and then a graph-only HUGT. The recommendation performance of the default HUGT and the two alternative architectures are shown in Table 3. We observe that on the three datasets with long sequences (ML1M, Netflix, Taobao), concurrent modeling of sequences and graphs outperforms the asynchronous alternatives. This demonstrates the effectiveness of the heterogeneous graph modeling of users and the proposed Transformers. On the Steam dataset, there is no significant performance difference between the three architectures.

6 Conclusion

We introduce a novel heterogeneous user graph (HUG) that jointly models users as sequences and graphs to encode item transition patterns and complex item relations. To leverage the item transition patterns and graph topology stored in HUGs for recommendation tasks, we develop heterogeneous user graph transformer (HUGT) that extends the standard self-attention for concurrent sequence learning and graph learning. Experimental results show that the unified sequence-graph user modeling is better than either sequence-only modeling or graph-only modeling and HUGT achieves state-of-the-art performance on all benchmarks.

HUG and HUGT provide a flexible framework for future research to incorporate complex item relations into user modeling. A valuable direction to explore is the choice of graphs for user modeling. We demonstrated that item collaborative similarity graph could provide extra context information for user modeling. There remains a lot of choices such as item temporal graph, or item feature similarity graph. We leave them as future works.

References

[1] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. Sequential recommendation with graph neural networks. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 378–387, 2021.

[2] Huiyuan Chen, Yusan Lin, Menghai Pan, Lan Wang, Chin-Chia Michael Yeh, Xiaoting Li, Yan Zheng, Fei Wang, and Hao Yang. Denoising self-attentive sequential recommendation. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 92–101, 2022.

[3] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation. In *Fifteenth ACM Conference on Recommender Systems*, pages 143–153, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 191–200. IEEE, 2016.

[6] Zhankui He, Handong Zhao, Zhe Lin, Zhaowen Wang, Ajinkya Kale, and Julian McAuley. Locker: Locally constrained self-attentive sequential recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3088–3092, 2021.

[7] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[9] Mingi Ji, Weonyoung Joo, Kyungwoo Song, Yoon-Yeong Kim, and Il-Chul Moon. Sequential recommendation with relation-aware kernelized self-attention. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4304–4311, 2020.

[10] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[11] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206. IEEE, 2018.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Jing Lin, Weike Pan, and Zhong Ming. Fissa: fusing item similarity models with self-attention networks for sequential recommendation. In *Fourteenth ACM Conference on Recommender Systems*, pages 130–139, 2020.

[14] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. Temporal-contextual recommendation in real-time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2291–2299, 2020.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[16] 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.

[17] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, 2018.

[18] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[19] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[22] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[23] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 169–178, 2020.

[24] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. Sse-pt: Sequential recommendation via personalized transformer. In *Fourteenth ACM Conference on Recommender Systems*, pages 328–337, 2020.

[25] 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.

[26] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.

[27] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenburg, and Jure Leskovec. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The World Wide Web Conference*, WWW '19, page 2236–2246, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313747. URL `https://doi.org/10.1145/3308558.3313747`.

[28] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.