

Interactive Recommender System via Knowledge Graph-enhanced Reinforcement Learning

Sijin Zhou¹, Xinyi Dai¹, Haokun Chen¹, Weinan Zhang¹, Kan Ren¹

Ruiming Tang², Xiuqiang He², Yong Yu¹

¹Shanghai Jiao Tong University, ²Huawei Noah's Ark Lab

{zhousijin, xydai, chenhaokun, wnzhang, kren, yyu}@apex.sjtu.edu.cn, {tangruiming, hexiuqiang1}@huawei.com

ABSTRACT

Interactive recommender system (IRS) has drawn huge attention because of its flexible recommendation strategy and the consideration of optimal long-term user experiences. To deal with the dynamic user preference and optimize accumulative utilities, researchers have introduced reinforcement learning (RL) into IRS. However, RL methods share a common issue of sample efficiency, i.e., huge amount of interaction data is required to train an effective recommendation policy, which is caused by the sparse user responses and the large action space consisting of a large number of candidate items. Moreover, it is infeasible to collect much data with explorative policies in online environments, which will probably harm user experience. In this work, we investigate the potential of leveraging knowledge graph (KG) in dealing with these issues of RL methods for IRS, which provides rich side information for recommendation decision making. Instead of learning RL policies from scratch, we make use of the prior knowledge of the item correlation learned from KG to (i) guide the candidate selection for better candidate item retrieval, (ii) enrich the representation of items and user states, and (iii) propagate user preferences among the correlated items over KG to deal with the sparsity of user feedback. Comprehensive experiments have been conducted on two real-world datasets, which demonstrate the superiority of our approach with significant improvements against state-of-the-arts.

KEYWORDS

Interactive Recommender Systems, Reinforcement Learning, Knowledge Graphs, Graph Neural Networks

ACM Reference Format:

Sijin Zhou, Xinyi Dai, Haokun Chen, Weinan Zhang, Kan Ren, Ruiming Tang, Xiuqiang He and Yong Yu. 2020. Interactive Recommender System via Knowledge Graph-enhanced Reinforcement Learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'20)*, July 25–30, 2020, Virtual Event, China. ACM, NY, NY, USA, 10 pages. <https://doi.org/10.1145/3397271.3401174>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGIR '20, July 25–30, 2020, Virtual Event, China

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-8016-4/20/07...\$15.00

<https://doi.org/10.1145/3397271.3401174>

1 INTRODUCTION

With the wide use of mobile applications such as TikTok, Pandora radio and Instagram feeds, interactive recommender systems (IRS) have received much attention in recent years [29, 43]. Unlike that in traditional recommender systems [11, 19, 33], where the recommendation is treated as a one-step prediction task, the recommendation in IRS is formulated as a multi-step decision-making process. In each step, the system delivers an item to the user and may receive feedback from her, which subsequently derives the next recommendation decision in a sequential manner. The recommendation-feedback interaction is repeated until the end of this visit session of the user. The goal of the IRS is to explore users' new interests, as well as to exploit the learned preferences, to provide accurate predictions, so as to optimize the outcome of the entire recommendation sequence [42, 43].

One way to implement IRS and balance the exploration and exploitation is multi-armed bandit (MAB) methods [20, 29, 37]. In MAB-based models, the user preference is often modeled by a linear function that is continuously learned through the interactions with proper exploration-exploitation tradeoff. However, these MAB-based models pre-assume that the underlying user preference remains unchanged during the recommendation process, i.e., they do not model the dynamic transitions of user preferences [43]. The key advantage for modern IRS is to learn about the possible dynamic transitions of the user's preference and optimize the long-term utility.

Recently, some researchers have incorporated deep reinforcement learning (DRL) models into interactive recommender system [3, 13, 41, 42, 44], due to the great potential of DRL in decision making and long-term planning in dynamic environment [26]. Mahmood and Ricci [22] first proposed to use model-based techniques in RL where dynamic programming algorithms such as policy iteration are utilized. Some recent works use model-free frameworks to tackle IRS tasks, e.g., deep Q-network (DQN) [42] and deep deterministic policy gradient (DDPG) [13].

Nevertheless, employing DRL in real-world interactive recommender system is still challenging. A common method for training recommendation models is to make use of offline logged data directly, but it will suffer from the estimation bias problem [4] under the real-time interaction setting. In contrast, the ideal setting of learning the optimal recommendation policy is to train the agent online. However, due to the item-state search space for each recommendation step and the trial-and-error nature of RL algorithms, DRL methods normally face sample efficiency problem [45], i.e., learning such a policy requires a huge amount of data through interacting with real users before achieving the best policy, which may

degrade user experience and damage system profit [39]. Therefore, it is quite crucial to improve the sample efficiency of existing DRL models with only a limited amount of interaction data.

Fortunately, there is rich prior knowledge from other external sources that may contribute to dealing with the above problems, such as textual reviews, visual images or item attributes [5]. Among these, **knowledge graph (KG)**, a well-known structured knowledge base, represents various relations as the attributes of items and links items if they have common attributes, which has shown great effectiveness for representing the correlation between items [30]. The association between the items provided by KG is very suitable for the recommendation scenarios. For example, a user likes the movie *Inception*, and the information behind it may be that her favorite director is *Nolan*. With such links among actions (items) on the graph, one user-item interaction record could reveal the user's preference on multiple connected items. In addition, the information contained in the semantic space of the entire knowledge graph will also be helpful in extracting user interest during the recommendation process. Since there have been many successful works applying open-sourced KGs (such as DBpedia, NELL, and Microsoft Satori) to *traditional* recommendation systems [30, 36, 38], we believe that it is reasonably promising to leverage KG to DRL-based methods in IRS scenarios.

In this paper, we **make the first attempt to leverage KG for reinforcement learning in interactive recommender systems**, trying to address the aforementioned limitations of the existing DRL methods. We propose **KGQR (Knowledge Graph enhanced Q-learning framework for interactive Recommendation)**, a novel architecture that extends DQN. Specifically, we integrate graph learning and sequential decision making as a whole to facilitate knowledge in KG and pattern mining in IRS. On one hand, to alleviate data sparsity, the user feedback is modeled to propagate via structure information of KG, so that the user's preference can be transited among correlated items (in the KG). In this way, one interactive record can affect multiple connected items, thus the sample efficiency is improved. On the other hand, by aggregating the semantic correlations among items in KG, the item embedding and the user's preference are effectively represented, which leads to more accurate Q-value approximation and hence better recommendation performance. In addition, we also conduct a methodology to deal with action selection in such large space. Rather than enumerating the whole item set, each step the candidate set for recommendation is dynamically generated from the local graph of KG, by considering the neighborhood of the items in user's high-scored interacted items. The method of candidate selection forces the deep Q-network to fit on the samples that KG considers more useful through the structure of item correlations, hence it can make better use of limited learning samples for RL-algorithm.

To the best of our knowledge, this is the first work to introduce KG into RL-based methods to interactive recommender systems. The contributions of our work can be summarized as follows.

- We propose a novel end-to-end deep reinforcement learning based framework KGQR for interactive recommendation to addresses the sparsity issue. By leveraging prior knowledge in KG in both candidate selection and the learning of user preference from sparse user feedback, KGQR can improve sample efficiency of RL-based IRS models.
- The dynamic user preference can be represented more precisely by considering the semantic correlations of items in KG, with graph neural networks.
- Extensive experiments have been conducted on two real-world datasets, demonstrating that KGQR is able to achieve better performance than state-of-the-arts with much fewer user-item interactions, which indicates high sample efficiency.

2 RELATED WORK

Traditional KG Enhanced Recommendation. Traditional KG enhanced recommendation models can be classified into three categories: path-based methods, embedding-based methods and hybrid methods. In *path-based* methods [25, 36, 40], KG is often treated as a heterogeneous information network (HIN), in which specific meta-paths/meta-graphs are manually designed to represent different patterns of connections. The performance of these methods is heavily dependent on the hand-crafted meta-paths, which are hard to design. In *embedding-based* methods, the entity embedding extracted from KG via Knowledge Graph Embedding (KGE) algorithms (like TransE [2], TransD [15], TransR [21]), is utilized to better represent items in recommendation. Zhang et al. [38] propose Collaborative Knowledge Base Embedding (CKE), to jointly learn the latent representations in collaborative filtering as well as items' semantic representations from the knowledge base, including KG, texts, and images. MKR [31] associates the embedding learning on KG with the recommendation task by cross & compress units. KSR [14] extends the GRU-based sequential recommender by integrating it with a knowledge-enhanced Key-Value Memory Network. In *hybrid* methods, researchers combine the above two categories to learn the user/item embeddings by exploiting high-order information in KG. Ripplenet [30] is a memory-network-like model that propagates users' potential preferences along with links in the KG. Inspired by the development of graph neural network [9, 17, 28], KGAT [34] applies graph attention network [28] framework in a collaborative knowledge graph to learn the user, item and entity embeddings in an end-to-end manner.

However, most of these methods are one-step prediction tasks and can not model the iterative interactions with users. Besides, they all greedily optimize an immediate user's feedback and don't take the user's long-term utility into consideration.

Reinforcement Learning in IRS. RL-based recommendation methods model the interactive recommendation process as a Markov Decision Process (MDP), which can be divided into model-based and model-free methods. As one of the *model-based* techniques, Mahmood and Ricci [22] utilize policy iteration to search for the optimal recommendation policy where an action is defined to be an item and a state is represented as n -gram of items. Policy iteration needs to go through the whole state space in each iteration, with exponential complexity to the number of items. Therefore, it is unable to handle large state and action space.

Recently, most works on RL-based recommendation prefer *model-free* techniques, including policy gradient (PG)-based, DQN-based and DDPG-based methods. PG-based methods, as well as DQN-based, treat recommending an item as an action. PG-based methods [3] learn a stochastic policy as a distribution over the whole item space and sample an item according to such distribution.

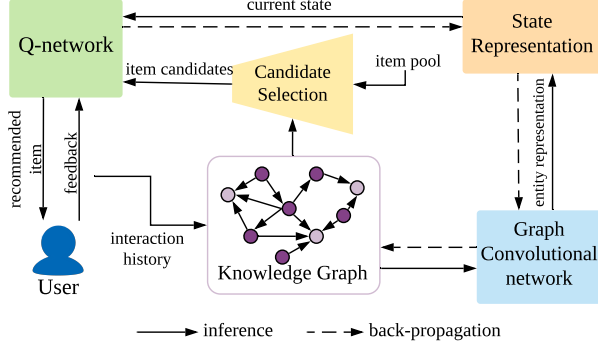


Figure 1: The overall KGQR framework. The left part illustrates the interaction process between the user and the IRS. The right part illustrates how the IRS recommends an item to the user according to past interaction history.

DQN-based methods [42, 44, 45] learn Q-value for each item and select the item with the maximum Q-value. Zheng et al. [44] combine DQN and Dueling Bandit Gradient Decent (DBGD) [8] to conduct online news recommendation. Zou et al. [45] integrate both the intent engagement (such as click and order) and long-term engagement (such as dwell time and revisit) when modeling versatile user behaviors. In DDPG-based works, an action is often defined as a continuous ranking vector. Dulac et al. [7] represent the discrete actions in a continuous vector space, pick a proto-action in a continuous hidden space according to the policy and then choose the valid item via a nearest neighbor method. In [13, 41], the policies compute the ranking score of an item by calculating the pre-defined function value (such as an inner product) of the generated action vector and the item embedding.

Nevertheless, all existing RL-based recommendation models suffer from low sample efficiency issue and need to pre-train user/item embeddings from history, which means that they cannot handle recommendation on cold-start problem well. A significant difference between our approach and the existing models is that we first propose a framework that combines the semantic and structural information of KG with the IRS to break such limitations.

3 PROBLEM FORMULATION

In feed streaming recommendation scenario, the interactive nature between the recommender system and the user is a multi-step interaction process that lasts for a period of time. At each timestep t , according to the observations on past interactions, the recommendation agent delivers an item $i_t \in \mathcal{I}$ to the user, and receives feedback (e.g., click, purchase or skip) from her. This process continues until the user leaves the recommender system. Under such circumstances, the interactive recommendation process can be formulated as a Markov Decision Process (MDP). The ultimate goal of the recommender system is to learn a recommendation policy $\pi : \mathcal{S} \rightarrow \mathcal{I}$, which maximizes the cumulative utility over the whole interactive recommendation process as

$$\pi^* = \arg \max_{\pi \in \Pi} \mathbb{E} \left[\sum_{t=0}^T r(s_t, i_t) \right]. \quad (1)$$

Table 1: Notations and descriptions.

Notations	Descriptions
\mathcal{U}, \mathcal{I}	Set of users and items in IRS.
$\mathcal{G} = (\mathcal{E}, \mathcal{R})$	Knowledge graph.
\mathcal{E}, \mathcal{R}	Set of entities and relations in \mathcal{G} .
$o_t = \{i_1, i_2, \dots, i_n\}$	Recorded user's positively interacted at timestep t .
s_t	Dense representation of user's preference at timestep t .
r_t	The user's reward at timestep t .
T	Episode length.
$e_h, h \in \mathcal{E}$	Dense representation of an entity.
$\mathcal{I}_t(\mathcal{G})$	Candidate action space at timestep t .
θ_S	Parameters of state representation network.
θ_Q, θ'_Q	Parameters of online Q-network /target Q-network.
\mathcal{D}	Replay Buffer.

Here $s_t \in \mathcal{S}$ is a representation abstracted from user's positively interacted items $o_t = \{i_1, i_2, \dots, i_n\}$ that denotes user's preference at timestep t ; $r(s_t, i_t)$ is the user's immediate feedback to the recommended item i_t at the state s_t according to some internal function $\mathcal{R} : \mathcal{S} \times \mathcal{I} \rightarrow \mathbb{R}$, abbreviated as r_t .

To achieve this goal, traditional recommendation methods usually adopt a greedy strategy and only optimize one-step reward, i.e., at each timestep t , they optimize the immediate reward r_t . Different from them, DRL algorithms take the long-term impact into consideration and explicitly model the long-run performance. They will optimize $\sum_{j=0}^{T-t} \gamma^j r_{t+j}$ at timestep t , instead. And $\gamma \in (0, 1]$ is the discount factor to control the degressively accumulated long-term rewards.

In general, we can use Q-value to evaluate the value of an action (i.e., recommending an item) taken at a given state, defined as

$$Q^\pi(s_t, i_t) = \mathbb{E} \left[\sum_{j=0}^{T-t} \gamma^j r_{t+j} \right] = \mathbb{E} \left[r_t + \sum_{j=1}^{T-t} \gamma^j r_{t+j} \right], \quad (2)$$

which is a weighted sum of the expected reward of all future steps starting from the current state and following the policy π to take actions. Then following the optimal Bellman equation [1], the optimal Q^* , having the maximum expected reward achievable is:

$$Q^*(s_t, i_t) = \mathbb{E}_{s_{t+1}} [r_t + \gamma \max_{i_{t+1}} Q^*(s_{t+1}, i_{t+1}) | s_t, i_t]. \quad (3)$$

Since the state and action spaces are usually enormous, we normally estimate the Q-value of each state-action (s_t, i_t) pair via a θ_Q -parameterized deep neural network, i.e., $Q_{\theta_Q}^*(s_t, i_t) \approx Q^*(s_t, i_t)$.

As mentioned in Section 1, learning this Q-function from scratch requires numerous interactions with real users due to the low data efficiency problem that is common in the RL algorithm. However, unlike basic RL algorithms, in RS scenarios, KG can provide complementary and distinguishable information for each item by their latent knowledge-level connection in the graph. Thus, with the prior knowledge of the environment and actions, the Q-function can be learned more efficiently,

$$Q_{\theta_Q}^*(s_t, i_t; \mathcal{G}) = Q_{\theta_Q}^*(s_t(\mathcal{G}), i_t(\mathcal{G})). \quad (4)$$

Here \mathcal{G} is the knowledge graph comprised of subject-property-object triples facts, e.g., triple $(Nolan, DirectorOf, Inception)$, which denotes *Nolan* is the director of *Inception*. And it is often present as $(head, relation, tail)$, $head \in \mathcal{E}$, $relation \in \mathcal{R}$, $tail \in \mathcal{E}$ and \mathcal{E}, \mathcal{R} denote the set of entities and relationships in \mathcal{G} , respectively.

Usually, an item $i \in \mathcal{I}$ can be linked to an entity $e \in \mathcal{E}$ in the knowledge graph, e.g., the movie *Godfather* from MovieLens dataset has a corresponding entity entry in DBpedia. We will introduce how we design the knowledge enhanced DRL framework for IRS in the following sections and the key notations used in this paper are summarized in Table 1.

4 KGQR METHODOLOGY

The overview of our proposed framework is shown in Figure 1. Generally, our KGQR model contains **four main components: graph convolution module, state representation module, candidate selection module and Q-learning network module**. In the interactive recommendation process, at each timestep t , the IRS sequentially recommends items i_t to users, and correspondingly updates its subsequent recommendation strategy based on user’s feedback r_t . At the specific time during one recommendation session, according to the interaction history o_t combined with the knowledge graph \mathcal{G} , the IRS models the user’s preference s_t via graph convolution module and state representation module. The details of these two representation learning modules will be discussed in Section 4.1. Then the IRS calculates the highest-scored item in the candidate set through Q-network and recommends it to the user. We will introduce the candidate selection module and deep Q-network module in Section 4.2 and Section 4.3, respectively.

4.1 KG Enhanced State Representation

In IRS scenario, it is impossible to get user’s state s_t directly, and what we can directly observe is the recorded user-system interaction history o_t . As state is one of the key part in MDP, the design of state representation module is critical to study the optimal recommendation strategy.

4.1.1 Graph convolutional embedding layer. Generally, the state representation in IRS is abstracted from the user’s clicked¹ items, since the positive items represent the key information about what the user prefers [42]. Given the user’s history, we first convert the clicked item set $\{i_t\}$ into embedding vectors $\mathbf{i}_t \in \mathbb{R}^d$, where d is the dimension of the embeddings. Since we have already linked items with entities in KG, we can take advantage of the semantic and correlation information among items in KG for better item embedding $\mathbf{i}_t(\mathcal{G})$.

In order to distill structural and semantic knowledge in the graph into a low-dimensional dense node representation, different approaches of graph embedding methods can be applied. In addition to harvesting the semantic information, we incline to explicitly link these items so that one data can affect more items. Thus, a **graph convolutional network (GCN) [17] is used in our work to recursively propagate embeddings along the connectivity of items and learn the embeddings of all entities $\{\mathbf{e}_h \in \mathbb{R}^d\}_{h \in \mathcal{E}}$ on the graph \mathcal{G} .**

The computation of the node’s representation in a single graph convolutional embedding layer is a two-step procedure: aggregation and integration. These two procedures can naturally be extended to multiple hops, and we use the notation k to identify k -th hop. In each layer, first, we aggregate the representations of the neighboring

nodes of a given node h :

$$\mathbf{e}_{N(h)}^{k-1} = \frac{1}{|N(h)|} \sum_{t \in N(h)} \mathbf{e}_t^{k-1}, \quad (5)$$

where $N(h) = N(head) = \{tail \mid (head, relation, tail) \in \mathcal{G}\}$ is the set of neighboring nodes of h . Notice that, here we consider the classic *Average* aggregator for example, other aggregator like *concat* aggregator [9], *neighbor aggregator* or *attention mechanism (GAT)* [28] can also be implemented.

Second, we integrate the neighborhood representation with h ’s representation as

$$\mathbf{e}_h^k = \sigma(\mathbf{W}_k \mathbf{e}_{N(h)}^{k-1} + \mathbf{B}_k \mathbf{e}_h^{k-1}), \quad (6)$$

where \mathbf{W}_k and \mathbf{B}_k are trainable parameters for k -hop neighborhood aggregator and σ is the activation function implemented as $\text{ReLU}(x) = \max(0, x)$. In Equation 6, we assume the neighborhood representation and the target entity representation are integrated via a multi-layer perceptron. After k -hop graph convolutional embedding layer, each clicked item is then converted into $\mathbf{i}_t(\mathcal{G}) = \mathbf{e}_{i_t}^k$.

4.1.2 Behavior aggregation layer. Since the interactive recommendation is a sequential decision-making process, at each step, the model requires the current observation of the user as input, and provides a recommended item i_t as output. It is natural to use autoregressive models such as recurrent neural networks (RNN) to represent the state based on the observation-action sequence [10, 23]. Thus, we use an RNN with a gated recurrent unit (GRU) as the network cell [6] to aggregate user’s historical behaviors and distill user’s state $s_t(\mathcal{G})$. The update function of a GRU cell is defined as

$$\begin{aligned} \mathbf{z}_t &= \sigma_g(\mathbf{W}_z \mathbf{i}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z), \\ \mathbf{r}_t &= \sigma_g(\mathbf{W}_r \mathbf{i}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r), \\ \hat{\mathbf{h}}_t &= \sigma_h(\mathbf{W}_h \mathbf{i}_t + \mathbf{U}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{b}_h), \\ \mathbf{h}_t &= (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \hat{\mathbf{h}}_t, \end{aligned} \quad (7)$$

where \mathbf{i}_t denotes the input vector, \mathbf{z}_t and \mathbf{r}_t denote the update gate and reset gate vector respectively, \circ is the elementwise product operator. The update function of hidden state \mathbf{h}_t is a linear interpolation of previous hidden state \mathbf{h}_{t-1} and a new candidate hidden state $\hat{\mathbf{h}}_t$. The hidden state \mathbf{h}_t is the representation of current user state, which is then fed into the Q-network, i.e.,

$$s_t(\mathcal{G}) = \mathbf{h}_t. \quad (8)$$

For simplicity, the set of the whole network parameters for computing $s_t(\mathcal{G})$, including parameters of graph convolutional layer and parameters of GRU cell, is denoted as θ_S .

In Figure 2(a), we illustrate the knowledge enhanced state representation module elaborated above. The upper part is the recurrent neural network that takes clicked item’s embedding at each timestep as the input vector, and outputs the hidden state of the current step as the state representation. The item embeddings, which are the input to GRU, are learned by performing graph convolutional network in KG, as shown in the bottom part.

4.2 Neighbor-based Candidate Selection

Generally, the clicked items have some inherent semantic characteristics, e.g., similar genre movies [30]. Since users are usually

¹Without loss of generality, we take “click” behavior as user positive feedback as the running example.

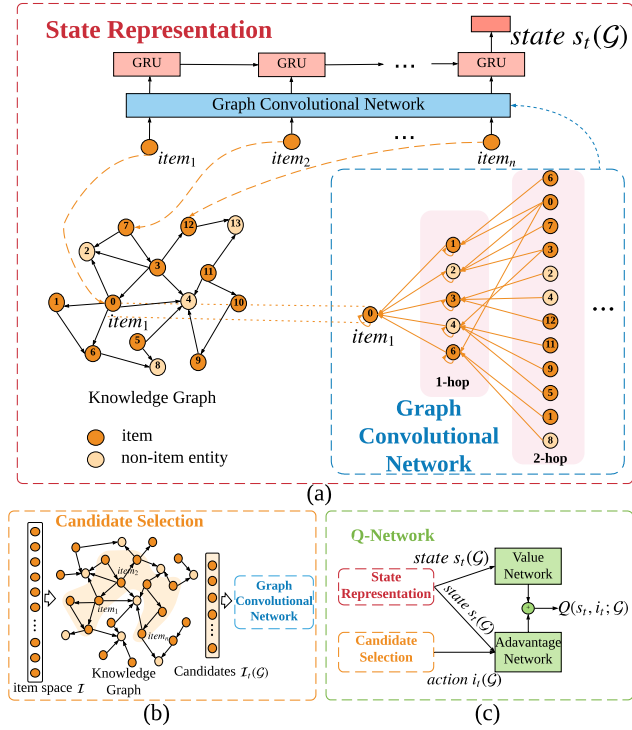


Figure 2: The neural architecture of KGQR. (a) The knowledge-enhanced state representation module maintains user’s preferences with a recurrent neural network and a graph neural network; (b) The candidate selection module dynamically reduces large action space according to user’s positive feedback; (c) The Q-value network predicts the Q-value with a value network and an advantage network.

not likely to be interested in all items, we can focus on selecting the potential candidates for restricted retrieval based on this semantic information in KG. Specifically, we utilize KG to filter some irrelevant items (i.e., actions) and dynamically obtain the potential candidates. The restricted retrieval will focus the data samples on the area that is more useful, as suggested in the structure of item correlation. Thus, these potential candidates would not only reduce the large searching space, but also improve the sample efficiency of policy learning.

More specifically, we perform a sampling strategy based on the k -hop neighborhood in KG. In each timestep t , the user’s historical interacted items serve as the seed set $\mathcal{E}_t^0 = \{i_1, i_2, \dots, i_n\}$. The k -hop neighborhood set starting from the seed entities is denoted as

$$\mathcal{E}_t^k = \left\{ \text{tail} \mid (\text{head}, \text{relation}, \text{tail}) \in \mathcal{G} \text{ and } \text{head} \in \mathcal{E}_t^{l-1} \right\}, \quad (9)$$

$$l = 1, 2, \dots, k.$$

Then, the candidate action set for the current user state is defined as

$$I_t(\mathcal{G}) = \left\{ \text{item} \mid \text{item} \in \bigcup_{l=1}^k \mathcal{E}_t^l \text{ and } \text{item} \in I \right\}, \quad (10)$$

with a user-defined cardinality. The shallow part in "candidate selection" in Figure 2(b) denotes the selected actions with the information from KG. Then all candidate items get their embedding through the graph convolutional layers.

4.3 Learning Deep Q-Network

After modeling the user’s state $s_t(\mathcal{G})$ and obtaining candidate sets $I_t(\mathcal{G})$, we need to design Q-network to combine this information and improve the recommendation policy for the interactive recommendation process. Here we implement a deep Q-network (DQN) with dueling-Q [35] and double-Q [27] techniques to model the expected long-term user satisfaction from the current user state as well as to learn the optimal strategy.

4.3.1 Deep Q-network. We adopt dueling technique to reduce the approximation variance and stabilize training [35]. That is, using two networks to compute the value function $V(i_t(\mathcal{G}))$ and advantage functions $A(s_t(\mathcal{G}), i_t(\mathcal{G}))$ respectively, and it is shown in Figure 2. Then the Q-value can be computed as,

$$Q(s_t(\mathcal{G}), i_t(\mathcal{G}); \theta_V, \theta_A) = V(i_t(\mathcal{G}); \theta_V) + A(s_t(\mathcal{G}), i_t(\mathcal{G}); \theta_A). \quad (11)$$

Here the approximation of value function and advantage function are accomplished by multi-layer perceptrons. θ_V and θ_A is the parameter of value function and advantage function respectively and we denote $\theta_Q = \{\theta_V, \theta_A\}$.

4.3.2 Model training. With the proposed framework, we can train the parameters of the model through trial-and-error process. During the interactive recommendation process, at timestep t , the recommender agent gets the user’s state s_t from the observations o_t about her, and recommended an item i_t via an ϵ -greedy policy (i.e., with probability $1 - \epsilon$ choosing the item in the candidate with the max Q-value, with probability ϵ choosing a random item). Then the agent receives the reward r_t from the user’s feedback and stores the experience (o_t, i_t, r_t, o_{t+1}) in the replay buffer \mathcal{D} . From \mathcal{D} , we sample mini-batch of experiences, and minimize the mean-square loss function to improve the Q-network, defined as

$$L(\theta_Q) = \mathbb{E}_{(o_t, i_t, r_t, o_{t+1}) \sim \mathcal{D}} [(y_t - Q(s_t, i_t; \theta_Q))^2]. \quad (12)$$

Here y_t is the target value based on the optimal Q^* . According to Equation (3), y_t is defined as

$$y_t = r_t + \gamma \max_{i_{t+1} \in I_{t+1}(\mathcal{G})} Q(s_{t+1}, i_{t+1}; \theta_Q). \quad (13)$$

To alleviate the overestimation problem in original DQN, we also utilize a target network Q' along with the online network Q (i.e., the double DQN architecture [27]). The online network back-propagates and updates its weights at each training step. The target network is a duplicate of the online network and updates its parameters with training delay. The target value of the online network update is then changed to

$$y_t = r_t + \gamma Q'(s_{t+1}, \arg \max_{i_{t+1} \in I_{t+1}(\mathcal{G})} Q(s_{t+1}, i_{t+1}; \theta_Q); \theta'_Q), \quad (14)$$

where θ'_Q denotes the parameter of the target network, and θ'_Q updates according to soft assign as

$$\theta'_Q = \tau \theta_Q + (1 - \tau) \theta'_Q, \quad (15)$$

where the interpolation parameter τ is also called update frequency.

Algorithm 1: Training KGQR

Input: $\mathcal{D}; \tau; \epsilon;$
Output: $\theta_S; \theta_Q; \{e_h\}_{h \in \mathcal{E}};$

- 1 Initialize all parameters: $\theta_S, \theta_Q, \{e_h\}_{h \in \mathcal{E}}, \theta'_Q \leftarrow \theta_Q;$
- 2 **repeat**
- 3 **for** $u \in \mathcal{U}$ **do**
- 4 Initialize the clicked history $\mathbf{x} \leftarrow \{\};$
- 5 **for** $t \in \{0, 1, \dots, T\}$ **do**
- 6 **for** $\mathbf{x} = o_t = \{i_1, i_2, \dots, i_n\}$, get $\{i_1, \dots, i_n\}$ via Eq.(5), Eq.(6);
- 7 Get s_t via Eq.(7), Eq.(8);
- 8 Recommend i_t by ϵ -greedy w.r.t Q-value in Eq.(11);
- 9 Receive reward r_t ;
- 10 **if** $r_t > 0$ **then**
- 11 Append i_t to \mathbf{x} ;
- 12 **end**
- 13 Set $o_{t+1} \leftarrow \mathbf{x}$;
- 14 Get $I_{t+1}(\mathcal{G})$ via Eq.(9), Eq.(10);
- 15 Store $(o_t, i_t, r_t, o_{t+1}, I_{t+1}(\mathcal{G}))$ to buffer \mathcal{D} ;
- 16 **end**
- 17 Sample mini-batch of tuples
 $(o_t, i_t, r_t, o_{t+1}, I_{t+1}(\mathcal{G})) \sim \text{Unif}(\mathcal{D});$
- 18 Get s_t, s_{t+1} from o_t, o_{t+1} via Eq.(5), Eq.(6) and Eq.(7);
- 19 Construct target values y_t via Eq. (14);
- 20 Update $\theta_S, \theta_Q, \{e_h\}_{h \in \mathcal{E}}$ via SGD w.r.t the loss function Eq.(12);
- 21 Update θ'_Q via Eq. (15);
- 22 **end**
- 23 **until** *covered*;

To summarize, the training procedure of our KGQR is presented in Algorithm 1. Note that this paper mainly focuses on the way of incorporating KG into DRL methods for IRS. Thus we study the most typical DQN model as a running example. Our method can be seamlessly incorporated into other DRL models such as policy gradient (PG) [3], DDPG [13] etc.

5 EXPERIMENT

We conduct experiments on two real-world datasets to evaluate our proposed KGQR framework. We aim to study the following research questions (RQs):

- **RQ1:** How does KGQR perform as compared with state-of-the-art interactive recommendation methods?
- **RQ2:** Does KGQR improve sample efficiency?
- **RQ3:** How do different components (i.e., KG-enhanced state representation, GCN-based task-specific representation learning, neighbor-based candidate selection) affect the performance of KGQR?

5.1 Experimental Settings

5.1.1 Datasets. We adopt two real-world benchmark datasets for evaluation and describe them as below.

Book-Crossing² is a book rating dataset from Book-Crossing community. The ratings are ranging from 0 to 10. This dataset is linked with Microsoft Satori and the sub-KG is released by [30].

Table 2: Statistics of the datasets.

		Book-crossing	Movielens-20M
User-Item Interaction	#User	17,860	16,525
	#Linked Items	14,910	16,426
	#Interactions	139,746	6,711,013
Knowledge Graph	#Entities	77,903	101,769
	#Relation Types	25	32
	#Triples	151,500	489,758

Movielens-20M³ is a benchmark dataset, which consists of 20 million ratings from users to movies in MovieLens website. The ratings are ranging from 1 to 5. It is also linked with Microsoft Satori and the sub-KG is released by [32].

For Book-Crossing dataset, we follow the processing of [30] to convert original ratings into two categories, 1 for high ratings, 0 for others. For MovieLens-20M dataset, we keep the users with at least 200 interactions. The statistics information of these two datasets is presented in Table 2.

We choose these two typical datasets since our work focuses on incorporating KG into RL-based models for IRS. The experiments on more datasets with rich domain information such as news or images will be left as future work.

5.1.2 Simulator. Due to the interactive nature of our problem, online experiments where the recommender system interacts with users and learns the policy according to the users' feedback directly would be ideal. However, as mentioned in Section 1, the trial-and-error strategy for training policy in an online fashion would degrade the user's experience, as well as the system profit. Thus, the community has formed a protocol [3, 7, 13, 29, 41] to build up an environment simulator based on offline datasets for evaluation.

Following the experiment protocol in [3], our mimic environment simulator takes into account the instinctive feedback as well as the sequence nature of user behavior. We perform matrix factorization to train the 20-dimensional embeddings of the users and items. Then we normalize the ratings of each dataset into the range $[-1, 1]$, and use them as users' instinctive feedback. Then we combine a sequential reward with the instinctive reward. For instance, if the recommender system recommends an item i_j to a user u_i at timestep t , the final reward function comes as

$$R(s_t, i_t) = r_{ij} + \eta(c_p - c_n), \quad (16)$$

where r_{ij} is the predicted rating given by the simulator, c_p and c_n means the consecutive positive and negative counts representing the sequential pattern, and η is a trade-off between instinctive feedback and sequential nature. In our experiment, η is chosen from $\{0.0, 0.1, 0.2\}$, following the empirical experience in [3].

For each dataset, we randomly divide the users into two parts: 80% of the users are used for training the parameters of our model, and the other 20% are used for testing the model performance. Due to train/test dataset splitting style, the users in our test set never exist in the training set. That is to say, the experiment is a cold-start scenario, which means there is no user click history at the beginning.

²<http://www2.informatik.uni-freiburg.de/~cziegler/BX/>

³<https://grouplens.org/datasets/movielens/>

To handle cold-start problem, the recommender system collects the most popular items among the training users, and recommends a popular item to a test user at step t_0 . Then, according to the user’s feedback, the recommender system recommends items to the user interactively. Besides, we remove the recommended items from the candidate set to avoid repeated recommendation in one episode. The episode length $T = 32$ for all the two datasets in our experiment.

5.1.3 Evaluation Metrics. Three evaluation metrics are used.

Average Reward. As an IRS aims to maximize the reward of the whole episode, a straightforward evaluation measure is the average reward over each interaction of test users.

$$\text{Reward} = \frac{1}{\#users \times T} \sum_{users} \sum_{t=1}^T \gamma^t R(s_t, i_t) \quad (17)$$

We also check for the precision and recall during T timesteps of the interactions, which are widely used metrics in traditional recommendation tasks.

Average Cumulative Precision@T.

$$\text{Precision@T} = \frac{1}{\#users \times T} \sum_{users} \sum_{t=1}^T \theta_{hit} \quad (18)$$

Average Cumulative Recall@T.

$$\text{Recall@T} = \frac{1}{\#users} \sum_{users} \sum_{t=1}^T \frac{\theta_{hit}}{\#preferences} \quad (19)$$

We define $\theta_{hit} = 1$ if the instinctive feedback of the recommended item given by the simulator is higher than the predefined threshold, which is 0.5 in Book-Crossing and 3.5 in Movielens-20M. And we define $\#preferences$ is the total number of the positive instinctive feedback among all items, i.e., number of items with $r_{ij} > \text{boundary}$ based on the simulator.

Significance test. The Wilcoxon signed-rank test has been performed to evaluate whether the difference between KGQR and the other baselines is significant.

5.1.4 Baselines. We compare KGQR with 7 representative baseline methods in the IRS scenario, where GreedySVD and GRU4Rec are traditional recommendation methods, LinearUCB and HLinearUCB are based on multi-armed bandits, DDPG^{KNN}, DDPGR, DQNR are DRL-based methods.

GreedySVD is a well-known collaborative filtering methods via singular value decomposition [18]. In interactive scenarios, we train the model after each interaction with users and recommend an item with the predicted highest rating to one user.

GRU4Rec is a representative RNN-based sequential recommendation algorithm [12] to predict what the user will click at the next timestep based on the browsing histories.

LinearUCB is a multi-armed bandit algorithm [20] which selects items according to the estimated upper confidence bound (UCB) of the potential reward based on contextual information about the users and items.

HLinearUCB is a contextual bandit algorithm combined with extra hidden features [29].

DDPG^{KNN} is a DDPG-based method [7] which represents the discrete actions into a continuous vector space. The actor selects

a proto-action in the continuous space and then chooses the item with the max Q-value from the candidate items selected via K -nearest-neighbor (KNN) according to the proto-action. In this approach, a larger K value boosts the performance but also brings computational cost, indicating the existence of a trade-off between performance and efficiency. In our experiment, the K value is set to $\{1, 0.1N, N\}$, where N is the total number of items.

DDPGR is a DDPG-based method [41] where the actor learns a ranking vector. The vector is utilized to compute the ranking score of each item, by performing product operation between this vector and item embedding. Then the item with the highest ranking score is recommended to the user.

DQNR is a DQN-based method [44] where the recommender system learns a Q function to estimate the Q-value of all the actions at a given state. The method then recommends the item with highest Q-value at the current state.

Note that, traditional knowledge enhanced recommendation methods like CKE [38], RippleNet [30], KGAT [34] and etc. are not suitable for the online interactive recommendation as tested in this paper. Because our recommendation process is an online sequential recommendation in the case of cold-start setting, this means there is no data about the test user at the beginning. We model user preferences in real-time through the user interaction process and provide recommendations under the current situation. These traditional models could not handle this cold-start problem; thus, we do not compare our proposed model with them.

5.1.5 Parameter Settings. In KGQR, we set the maximal hop number $k = 2$ for both datasets. We have tried larger hops, and find that the model with larger hops brings exponential growth of computational cost with only limited performance improvement. The dimension of item embedding is fixed to 50 for all the models. For baseline methods, the item embedding is pre-trained by matrix factorization with the training set. For KGQR, we pre-train the embedding of KG by TransE [2], and then embedding will be updated while learning the deep Q-network. Besides, other parameters are randomly initialized with uniform distribution. The policy network in all the RL-based methods takes two fully-connected layers with activation function as ReLU. The hyper-parameters of all models are chosen by grid search, including learning rate, L_2 norm regularization, discount factor γ and etc. All the trainable parameters are optimized by Adam optimizer [16] in an end-to-end manner. We use PyTorch [24] to implement the pipelines and train networks with an NVIDIA GTX 1080Ti GPU. We repeat the experiments 5 times by changing the random seed for KGQR and all the baselines.

5.2 Overall Performance (RQ1)

Table 3 reports the performance comparison results. We have the following observations: (i) KGQR consistently obtains the best performance across all environment settings on both datasets. For instance, compared to RL-based recommendation methods like DQN-based (i.e., DQNR) and DDPG-based (e.g., DDPG^{KNN}, DDPGR), KGQR improves over the strongest baselines in terms of Reward by 3.2% and 5.3% in Book-Crossing and MovieLens-20M, respectively. For traditional evaluation metrics, KGQR improves Precision@32 by 0.5% and 1.9% in the two datasets, respectively. This demonstrates that the leverage of prior knowledge in KG significantly

Table 3: Overall Performance Comparison.

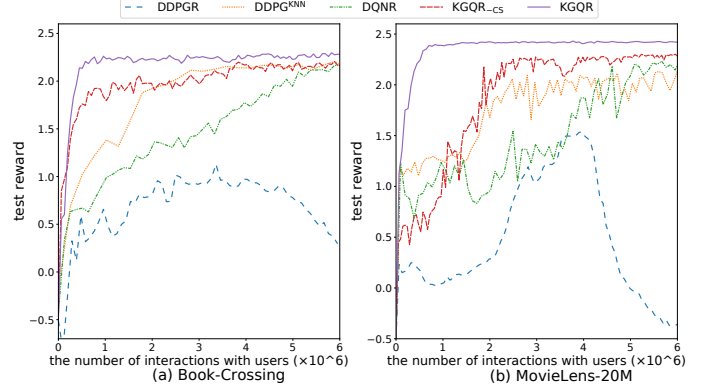
Dataset	Method	$\eta = 0$			$\eta = 0.1$			$\eta = 0.2$		
		Reward	Precision@32	Recall@32	Reward	Precision@32	Recall@32	Reward	Precision@32	Recall@32
Book-Crossing	Greedy SVD	-0.0890	0.3947	0.0031	-0.1637	0.4052	0.0032	-0.2268	0.4133	0.0033
	GRU4Rec	0.5162	0.8611	0.0070	1.3427	0.8595	0.0070	2.1797	0.8625	0.0070
	LinearUCB	-0.0885	0.3956	0.0032	-0.1640	0.4049	0.0032	-0.2268	0.4133	0.0033
	HLinearUCB	-0.1346	0.3819	0.0031	-0.3566	0.3841	0.0031	-0.6064	0.3713	0.0031
	DDPG	0.5521	0.9115	0.0074	1.1412	0.8800	0.0072	2.2057	0.9270	0.0076
	DDPG ^{KNN} (K=1)	0.3159	0.7302	0.0059	0.7312	0.7990	0.0065	0.8409	0.7472	0.0061
	DDPG ^{KNN} (K=0.1N)	0.7312	0.9907	0.0080	2.0750	0.9813	0.0080	3.3288	0.9758	0.0079
	DDPG ^{KNN} (K=N)	0.7639	0.9927	0.0081	2.2729	0.9942	0.0081	3.7179	0.9915	0.0081
MovieLens-20M	DQNR	0.7634	0.9936	0.0081	2.2262	0.9907	0.0080	3.6118	0.9881	0.0080
	KGQR	0.8307*	0.9945*	0.0081	2.3451*	0.9971*	0.0081	3.7661*	0.9966*	0.0081*
	Greedy SVD	0.4320	0.6569	0.0049	0.6915	0.6199	0.0048	0.9042	0.5932	0.0046
	GRU4Rec	0.7822	0.8382	0.0074	1.5267	0.8253	0.0072	2.3500	0.8316	0.0073
	LinearUCB	0.2307	0.3790	0.0029	0.6147	0.5821	0.0046	0.8017	0.5614	0.0044
	HLinearUCB	0.0995	0.3852	0.0029	0.0172	0.3841	0.0028	0.2265	0.3774	0.0027
	DDPG	0.2979	0.4917	0.0034	1.4952	0.7626	0.0055	2.3003	0.6977	0.0045
	DDPG ^{KNN} (K=1)	0.5755	0.7293	0.0059	1.0854	0.7165	0.0058	1.6912	0.7371	0.0061
	DDPG ^{KNN} (K=0.1N)	0.6694	0.8167	0.0070	1.1578	0.8165	0.0069	2.2212	0.8215	0.0068
	DDPG ^{KNN} (K=N)	0.8071	0.9606	0.0082	2.1544	0.9446	0.0081	3.6071	0.9533	0.0082
	DQNR	0.8863	0.9680	0.0086	2.3025	0.9667	0.0081	3.4036	0.9089	0.0071
	KGQR	0.9213*	0.9726*	0.0086*	2.4242*	0.9722*	0.0083*	3.7695*	0.9713*	0.0084*

* indicates statistically significant improvements (measured by Wilcoxon signed-rank test at $p < 0.05$) over all baselines.

Table 4: Sample Efficiency Comparison: number of interactions to achieve reward 0.5,1.0,1.5,2.0 for each dataset.

Model	Book-Crossing				MovieLens-20M			
	0.5	1.0	1.5	2.0	0.5	1.0	1.5	2.0
DDPG	0.46M	2.49M	-	-	2.28M	2.62M	3.82M	-
DDPG ^{KNN}	0.20M	0.49M	1.46M	2.44M	0.04M	0.07M	1.76M	2.42M
DQNR	0.20M	1.06M	3.09M	4.83M	0.06M	0.08M	2.47M	4.27M
KGQR	0.06M	0.17M	0.24M	0.40M	0.04M	0.06M	0.16M	0.33M

improves the recommendation performance. (ii) In most conditions, non-RL methods including conventional methods and MAB-based methods, perform worse than the RL-based methods. Two reasons stand for the significant performance gap. On the one hand, except GRU4Rec, the capacity of other non-RL methods are limited in modeling user preference without considering sequential information. On the other hand, they all focus on the immediate item reward and do not take the present value of the overall performance of the whole sequence into the current decision, which makes them perform even worse in environments that give future rewards more (e.g., $\eta = 0.1$, $\eta = 0.2$). (iii) Among the RL-based baselines, we can observe that DQNR and DDPG^{KNN} ($K = N$) achieves much better performance than the other DDPG based methods. When $K = N$ (N is the total number of items), DDPG^{KNN} can be seen as a greedy policy that always picks the item with max Q-value. We also notice that the training process of DDPG based methods is not stable, e.g., their training curves sometimes experience a sudden drop. This may be accounted for that the continuous proto-action picked by the actor is inconsistent with the final action that the critic is learned with. Such inconsistency between actor and critic may result in inferior performance.


Figure 3: Learning curves of KGQR and DRL-based baseline models.

5.3 Sample Efficiency (RQ2)

One motivation of exploiting KG is to improve sample efficiency in RL-based recommendation, i.e., to reduce the amount of interaction data needed to achieve the same performance. In Figure 3 and Table 4, we analyze the number of interactions needed for each DRL-based model to achieve the same performance with the environment of $\eta = 0.1$ in Eq.(16). As can be observed, our proposed KGQR can achieve the same performance as the other RL-based methods using the least number of interactions. More specifically, to achieve a test reward of 2.0, KGQR only needs 17.3% and 13.6% of interactions compared to the second efficient baseline (i.e., DDPG^{KNN}) in the two datasets. This result empirically validates that sample efficiency is improved by utilizing the semantic and correlation information of items in KG. The detailed analysis of different components that contributes to improve sample efficiency is proposed in Section 5.4.2.

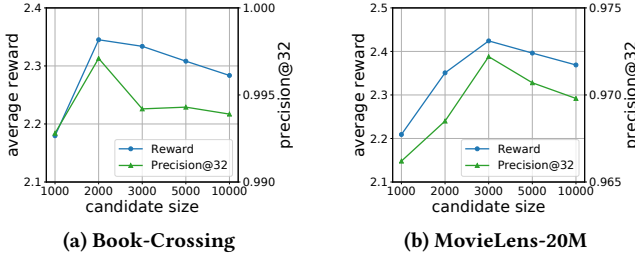
Table 5: Comparison of Different KGQR Variants.

	KGQR-KG	KGQR-GCN-CS	KGQR-CS	KGQR
KGemb*	×	✓	✓	✓
GCNprop*	×	×	✓	✓
CS*	×	×	×	✓

* KGemb denotes KG enhanced item representation; GCNprop denotes GCN propagation in state representation; CS denotes the neighbor-based candidate selection.

Table 6: Ablation Study of KGQR.

Model	Book-Crossing		Movielens-20M	
	Reward	Precision@32	Reward	Precision@32
KGQR-KG	2.2262	0.9907	2.3025	0.9667
KGQR-GCN-CS	2.2181	0.9819	2.2402	0.9621
KGQR-CS	2.2836	0.9939	2.3689	0.9698
KGQR	2.3451	0.9971	2.4242	0.9722

**Figure 4: Influence of Candidate Size.**

5.4 Analysis (RQ3)

In this section, we further analyze the effectiveness of different components in the proposed framework. In KGQR, there are three components utilizing KG that may affect the performance of KGQR: KG enhanced item representation, GCN propagation in state representation (Section 4.1) and neighbor-based candidate selection (Section 4.2). To study the effectiveness of each such component, we evaluate the performance of four different KGQR variants, namely KGQR-KG (i.e., DQNR), KGQR-CS, KGQR-GCN-CS and KGQR. The relationship between KGQR variants and different components is presented in Table 5. In the ablation study, we consider the environment of $\eta = 0.1$ in Eq.(16), and Table 6 shows the performance of these four variants.

5.4.1 Recommendation performance. Effect of KG enhanced item representation. In KGQR-KG, the item embeddings are pretrained by MF model from the historical interaction data, while in KGQR-GCN-CS, they are retrieved from KG, pretrained with TransE. Therefore, the marginal difference between KGQR-KG and KGQR-GCN-CS performance indicates that the information in KG has almost equal contribution with the historical interaction data, which suggests the superiority of applying KG for cold-start scenario (i.e., no historical interaction data exists).

Effect of GCN propagation in state representation. Comparing the performance of KGQR-GCN-CS with KGQR-CS in Table 6, the improvement in KGQR-GCN-CS indicates that the signal from RL-based recommendation guides the update of KG embedding so that the items in KG can be represented more suitably for the current specific recommendation task.

Effect of neighbor-based candidate selection. The comparison between the performance of KGQR-CS and KGQR validates the effectiveness of neighbor-based candidate selection, for candidate selection module can leverage the local structure of interacted items in KG to filter irrelevant items and such restricted retrieval improves the final recommendation performance.

To study the influence of candidate size, we vary the candidate size in the range of {1000, 2000, 3000, 5000, 10000} and present the recommendation performance in Figure 4. We observe that recommendation performance first grows as candidate size increases, since a small size of candidate limits possible choices of the recommendation algorithm. However, further increasing of candidate size degrades the performance, since the neighbor-based candidate selection filters some irrelevant items in advance. Such irrelevant items have very limited chance to be recommended and to collect feedback which makes them not be able to learn well by the recommendation algorithm and eventually gains a negative effect to the performance.

5.4.2 Sample efficiency. Effect of KG-enhanced state representation. Comparing the number of interactions of KGQR-KG with that of KGQR w.r.t same test average reward in Figure 3, we notice that in both two environments the utilizing of KG and task-specific representation learning improves the sample efficiency. This observation demonstrates our motivation that the propagation of user preference through the correlated items via GCN is helpful in dealing with sample efficiency issues.

Effect of neighbor-based candidate selection. Besides the performance improvements, the candidate selection significantly improves the sample efficiency, as shown in Figure 3 (comparing the purple line and red line).

6 CONCLUSION

In this work, we proposed a knowledge graph enhanced Q-learning framework (KGQR) for the interactive recommendation. To the best of our knowledge, it is the first work leveraging KG in RL-based interactive recommender systems, which to a large extent addresses the sample complexity issue and significantly improves the performance. Moreover, we directly narrow down the action space by utilizing the structure information of knowledge graphs to effectively address the large action space issue. The model propagates user preference among the correlated items in the graph, to deal with the extremely sparse user feedback problem in IRS. All these designs improve sample efficiency, which is a common issue in previous works. The comprehensive experiments with a carefully-designed simulation environment based on two real-world datasets demonstrate that our model can lead to significantly better performance with higher sample efficiency compared to state-of-the-arts.

For future work, we plan to investigate KGQR on news and image recommendation tasks with other DRL frameworks such as PG and DDPG. We are also scheduling the process of deploying KGQR onto an online commercial recommender system. Further, we are interested in inducing a more complex sequential model to represent the dynamics of user preferences, e.g., taking user's propensity to different relations that the click history shows into consideration.

ACKNOWLEDGEMENT

The corresponding author Weinan Zhang thanks the support of "New Generation of AI 2030" Major Project 2018AAA0100900 and NSFC (61702327, 61772333, 61632017, 81771937). The work is also sponsored by Huawei Innovation Research Program.

REFERENCES

- [1] Richard Bellman. 1952. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America* 38, 8 (1952), 716.
- [2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS'13*. 2787–2795.
- [3] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *AAAI'19*, Vol. 33. 3312–3320.
- [4] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *WSDM'19*. ACM, 456–464.
- [5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishith Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 7–10.
- [6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
- [7] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
- [8] Artem Grotov and Maarten de Rijke. 2016. Online learning to rank for information retrieval: SIGIR 2016 Tutorial. In *SIGIR'16*. ACM, 1215–1218.
- [9] Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS'17*. 1024–1034.
- [10] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.
- [11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WebConf'17*. International World Wide Web Conferences Steering Committee, 173–182.
- [12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. *ICLR'16*.
- [13] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *SIGKDD'18*. ACM, 368–377.
- [14] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *SIGIR'18*. ACM, 505–514.
- [15] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *IJCNLP'15*. 687–696.
- [16] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR'15*.
- [17] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
- [18] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD'08*. ACM, 426–434.
- [19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
- [20] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WebConf'10*. ACM, 661–670.
- [21] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI'15*.
- [22] Tariq Mahmood and Francesco Ricci. 2007. Learning and adaptivity in interactive recommender systems. In *Proceedings of the ninth international conference on Electronic commerce*. ACM, 75–84.
- [23] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *EMNLP&AAI'15*.
- [24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS'19*. 8024–8035.
- [25] Chuan Shi, Zhiqiang Zhang, Ping Luo, Philip S Yu, Yading Yue, and Bin Wu. 2015. Semantic path based personalized recommendation on weighted heterogeneous information networks. In *CIKM'15*. ACM, 453–462.
- [26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
- [27] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *AAAI'16*.
- [28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
- [29] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2017. Factorization bandits for interactive recommendation. In *AAAI'17*.
- [30] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM'18*. ACM, 417–426.
- [31] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In *WebConf'19*. ACM, 2000–2010.
- [32] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *WebConf'19*. ACM, 3307–3313.
- [33] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR'06*. ACM, 501–508.
- [34] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. *SIGKDD'19*.
- [35] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2016. Dueling network architectures for deep reinforcement learning. *ICML'16*.
- [36] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM'14*. ACM, 283–292.
- [37] Chunqiu Zeng, Qing Wang, Shekoofeh Mokhtari, and Tao Li. 2016. Online context-aware recommendation with time varying multi-armed bandit. In *SIGKDD'16*. ACM, 2025–2034.
- [38] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *SIGKDD'16*. ACM, 353–362.
- [39] Weinan Zhang, Ulrich Paquet, and Katja Hofmann. 2016. Collective noise contrastive estimation for policy transfer learning. In *AAAI'16*.
- [40] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *SIGKDD'17*. ACM, 635–644.
- [41] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *ACM RecSys'18*. ACM, 95–103.
- [42] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *SIGKDD'18*. ACM, 1040–1048.
- [43] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. 2013. Interactive collaborative filtering. In *CIKM'13*. ACM, 1411–1420.
- [44] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *WebConf'18*. International World Wide Web Conferences Steering Committee, 167–176.
- [45] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. *arXiv preprint arXiv:1902.05570* (2019).