Complex & Intelligent Systems (2023) 9:6645–6659
https://doi.org/10.1007/s40747-023-01083-7

ORIGINAL ARTICLE

KPRLN: deep knowledge preference-aware reinforcement learning network for recommendation

Di Wu[1] · Mingjing Tang[2,3] · Shu Zhang[1] · Ao You[1] · Wei Gao[1]

Received: 25 October 2022 / Accepted: 17 April 2023 / Published online: 26 May 2023

Abstract

User preference information plays an important role in knowledge graph-based recommender systems, which is reflected in users having different preferences for each entity–relation pair in the knowledge graph. Existing approaches have not modeled this fine-grained user preference feature well, as affecting the performance of recommender systems. In this paper, we propose a deep knowledge preference-aware reinforcement learning network (KPRLN) for the recommendation, which builds paths between user's historical interaction items in the knowledge graph, learns the preference features of each user–entity–relation and generates the weighted knowledge graph with fine-grained preference features. First, we proposed a hierarchical propagation path construction method to address the problems of the pendant entity and long path exploration in the knowledge graph. The method expands outward to form clusters centered on items and uses them to represent the starting and target states in reinforcement learning. With the iteration of clusters, we can better learn the pendant entity preference and explore farther paths. Besides, we design an attention graph convolutional network, which focuses on more influential entity–relation pairs, to aggregate user and item higher order representations that contain fine-grained preference features. Finally, extensive experiments on two real-world datasets demonstrate that our method outperforms other state-of-the-art baselines.

Keywords Knowledge graph · Recommender system · Deep reinforcement learning · Graph neural network

Introduction

With the rapid development of the Internet, recommendation systems are dedicated to enhancing the user experience in various online applications. Collaborative filtering (CF) analyzes the user's historical behavior to predict based on user–user and user–item similarities [1]. However, sparse and cold start problems plague CF-based methods. Graph-structured data (e.g., social networks, etc.) that contain the connections between entities can more accurately reflect real-world circumstances. The Knowledge Graph (KG) is a heterogeneous graph in which entities are linked by various relations, making the KG rich in semantic information [2, 3]. Therefore, KG is usually used as external information to improve the performance of recommender systems. As an example shown in Fig. 1, movie items are linked to other entities by different relations. Besides, KG can enhance the explainability of recommendation [4].

It is essential to obtain the user's interest preferences in the knowledge graph. As shown in the left of Fig. 1, a movie item links to other entities by different relations (e.g., stars, directors, writers, etc.). However, the reasons why users choose to watch movies are complex. The reason why u_1 choose this movie is that the actress in the film is Zhang Ziyi. However, for u_2, the director of the film, Ang Lee, is the reason why he likes this movie. This proves that different users have different behavioral motivations. On the right of Fig. 1, the user has tagged two movies (Forrest Gump and Crouching Tiger, Hidden Dragon). The reasons why this user likes the two movies are different. It shows that the same user will have different preferences when choosing different items. Therefore, we cannot simply calculate the user preference for the

✉ Mingjing Tang
tmj@ynnu.edu.cn

1 School of Information Science and Technology, Yunnan Normal University, Kunming, China

2 Key Laboratory of Educational Informatization for Nationalities Ministry of Education, Yunnan Normal University, Kunming, China

3 Yunnan Key Laboratory of Smart Education, Yunnan Normal University, Kunming, China

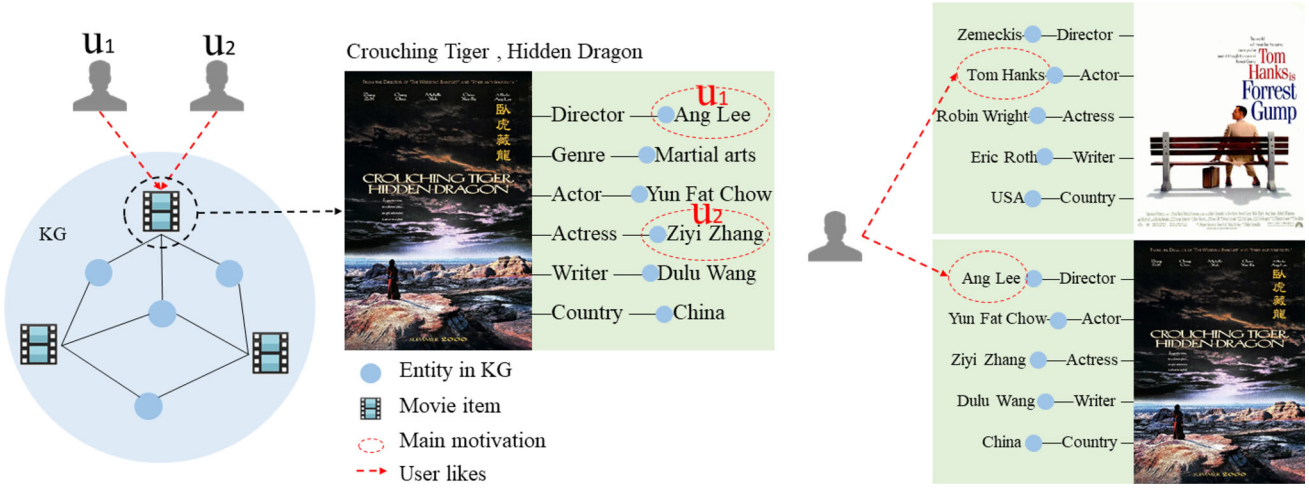


Fig. 1 An example of movie recommendation, in the left image u_1, u_2 like the same movie (Crouching Tiger, Hidden Dragon), but they like it for different reasons. u_1 is because of the movie's director, and u_2 is because of the movie's actor. In the figure on the right, the same user has different reasons for liking different movies; for Forrest Gump, it is because of the movie starring Tom Hanks, and for Crouching Tiger, Hidden Dragon because of the movie director Ang Lee

type of relationship but learn the preference features based on the user–entity–relation. Usually, the user preference can be reflected by the weight of the edge. Therefore, the personalized weighted knowledge graph of users can improve the performance of the recommender system.

Many works have shown that edge weights play an important role in the feature learning of graphs. GAT [5]computes the weights of edges based on the similarity of head and tail vertices through graph attention. KGAT [6] introduces this method into recommender systems. KGCN [7] and KGNN-LS [8] use a trainable and personalized relation scoring function to learn the weight for the relation between entities and have achieved good results. However, KGAT cannot distinguish the preference of different users for relations, and KGCN uses the score function to calculate the weight, but it cannot distinguish the user's preferences for the same relation on different entities. Personalized preference information influences the performance of recommendations [9, 10]. Currently, most KG-based methods cannot make fine-grained feature learning for each user–entity–relation. Users' preferences are fine-grained, and direct calculating preference for each user–entity–relation, not only increases the training burden of the model but also leads to model overfitting, which makes a worse performance.

In recent years, the successful application of deep reinforcement learning (RL) to graph structures has sparked great interest. Therefore, some research works combine reinforcement learning and knowledge graphs in recommender systems. However, learning fine-grained user preference features in the knowledge graph by reinforcement learning has the following challenges: (1) Data sparsity problem. The user interaction data are very sparse compared to the knowledge graph, and the distribution of user history interaction items in the knowledge graph may be discrete or aggregated. It is difficult to make a unified method exploring all user interaction items. However, we need to model each user, so it is necessary to learn personalized information about users quickly and efficiently. (2) Pendant entity in the knowledge graph. The pendant entity is the entity with just one adjacent neighbor in KG, which is not employed as the starting or target state in RL. It is not included in the path between items, so only negative feedback is available during the reinforcement learning training. However, they may contain useful information about user preferences. (3) Long path exploration, reinforcement learning tends to repeatedly explore shorter paths, because they lead to higher feedback rewards. Although we can design higher feedback rewards for long paths, this increases the training burden of the model. In addition, we believe that closer items reflect more user preference features. Therefore, it is necessary to make a well-designed explore policy in RL.

Considering the limitations of the existing methods, we propose a personalized Knowledge Preference-aware recommender systems combined with Reinforcement Learning Network (KPRLN). Specifically, we describe the learning user's preference for different entity-relation as the process of building interaction paths in the graph. By constructing the path network between the user's historical interaction items in the KG, which can be expressed as a Markov decision process [11]. Unlike previous work, to learn the user's preference features on the KG more completely, we replace the single node with a cluster of entities as the starting and target states. We iteratively extend user history interaction items along the links in the KG to their neighbors. The start-

ing and target states are represented as item-centric clusters, which reduce the complexity of the state space and action space. The cluster will add the pendant entity when expanding, and after a few iterations, the cluster will absorb the short path between items, making reinforcement learning explore more distant paths. Furthermore, we design hierarchical propagation paths to develop rewards, with each path construction making the model get more feedback. Meanwhile, we design two different reward mechanisms to make KPRLN have stable performance in recommendation tasks. Based on the expected payoff estimates for item–relation pairs, RL computes the preference for each user–entity–relation and generates the user's preference-weighted knowledge graph. In user and item representations learning, we design an attention mechanism to propagate the user's preference interests in their preference-weighted KGs, making KPRLN focus on influential entities and relations to aggregating item embedding representations. Extensive experiments on two real datasets show that our method has efficient performance.

Our contributions are summarized as follows:

1. We propose a personalized recommendation method, KPRLN, which only uses the topological information to learn the fine-grained features of each user–entity–relation.
2. We propose a hierarchical propagation path construction method, which can explore more complex states and increases the efficiency of the model.
3. Extensive experiments on two real-world datasets demonstrate that KPRLN outperforms state-of-the-art baselines. Furthermore, it also has a good performance in reducing the effect of noise and providing explainability for the recommendation.

Related work

Knowledge-aware recommendation method

Knowledge graph assists the recommender systems through multi-dimensional dense associations and rich semantic information between entities, providing a new perspective to enhance recommender systems. Currently, KG-based recommender systems are mainly divided into three categories: embedding-based method, path-based method, and unified method. Embedding-based method (e.g., TransE [12], TransH [13], MuPR [14], DihEdral [15], etc.) learns the embeddings of entities and relations in KG, and keeps the original structural information of KG as much as possible. For example, [16, 17] unify the structural knowledge and other side information in a unified CF framework. These methods focus on learning semantic associations between knowledge graph entities but ignore the connectivity patterns of information in KG and high-order relationships between entities. Therefore, they lack the interpretability of the recommendation process. The path-based method (e.g., [4, 18, 19] etc.) regards the KG as a heterogeneous information network, constructing and extracting the latent features based on meta-path/meta-graph between users and items. These methods' performance depends on manually designing meta-paths and meta-graphs, which makes it difficult to achieve optimal performance in reality and leads to information loss when dealing with KG with complex relationships. The unified method combines the ideal of the embedding-based method and path-based method. RippleNet [20] and KGCN [7] enrich the representation of users or items by aggregating the target entity and their multi-hop neighbors. These methods mine the information in the knowledge graph more comprehensively and perform well in recommender systems. However, RippleNet and KGCN do not learn the user preferences by each user–entity–relation.

The existing methods are not modeling the fine-grained preferences of users in KG well. We combine deep reinforcement learning with knowledge graph learning users' fine-grained preference features and apply them to the recommendation.

Reinforcement learning for recommendation

Reinforcement learning is a methodology of machine learning that, through interaction with their environment, learns to maximize a numerical reward. The Markov Decision Process (MDP), proposed by Bellman, is the most common form of defining reinforcement learning. After that, Q-learning further extended the application of reinforcement learning [21]. DQN [22] introduces deep learning in reinforcement learning, which significantly increases the application scenarios of reinforcement learning. A series of applications of deep reinforcement learning methods (e.g., AlphaGo [23], autonomous driving [24], etc.) have demonstrated its powerful potential.

In recommender systems, user interactions are sequential [25]. The recommendation process can be regarded as a sequential decision process [26], which is formulated as an MDP. Reinforcement learning interacts with recommender systems by the agent, which makes it effectively learn dynamic features and improve recommendation explainability. With the development of deep learning, deep reinforcement learning in recommender systems has aroused great interest in recent years. DRN [27] proposed news recommender systems based on the deep reinforcement learning framework, and [28] made non-sequential ranking recommendations by deep reinforcement learning. Meanwhile, some recommendation methods based on other deep learning models have been successfully applied [29], and many researchers have explored the application of deep

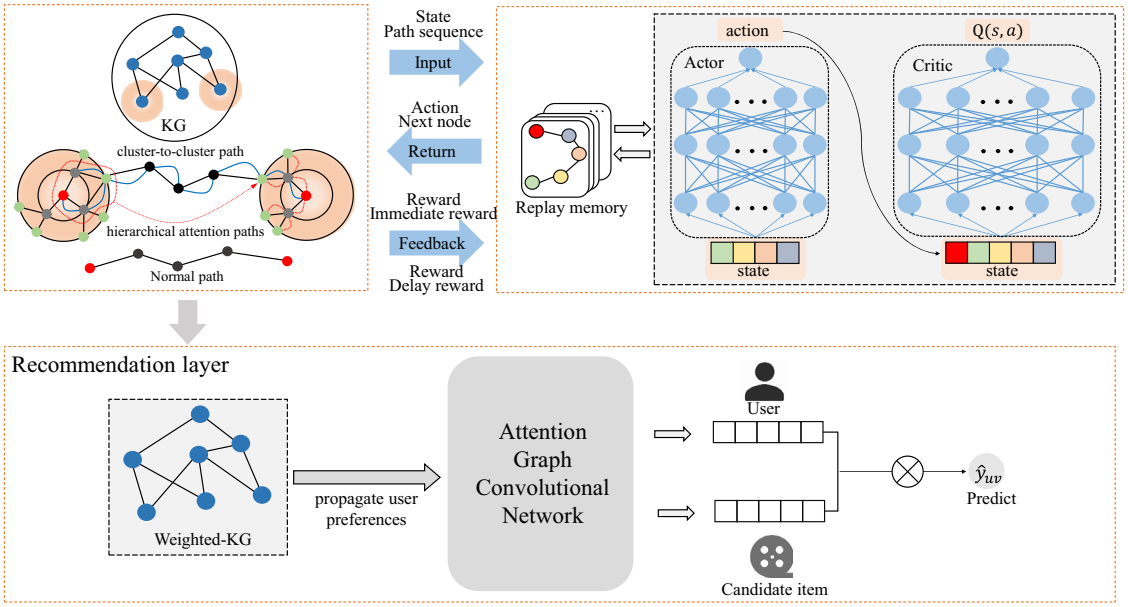


Fig. 2 Illustration of the KPRLN framework

reinforcement learning with other deep learning models in recommender systems.

In our work, we use deep reinforcement learning to enhance Graph Neural Networks (GNNs) feature representation by building preference-weighted graphs to learn the fine-grained preference features of users.

Reinforcement learning on knowledge graph

Knowledge graphs are often used as external information to improve the performance of recommender systems, and reinforcement learning increases the explainability of the recommendation due to the agent interacting with the recommender systems. Many works have explored the application of reinforcement learning in the knowledge graph. GRL [30] designed a generative adversarial net (GAN)-based reinforcement learning model for knowledge graph completion. DeepPath [31] applies reinforcement learning to knowledge graph reasoning. Specifically, DeepPath is to find reliable multi-hop paths between entity pairs in KG. [32] explored the application of RL to question–answering tasks in the KG environment, [33] was devoted to solving the problems of reinforcement learning-based path-finding methods in question–answering applications, PGPR [34] uses RL to find explainable paths between users and potential items, and [35] proposed a multimodal knowledge-aware reinforcement learning network dedicated to achieving interpretable causal reasoning procedures.

The above methods use reinforcement learning to find user–item or item–item paths in the knowledge graph. However, the challenges of reinforcement learning based on KG, such as the efficiency of model training, complete exploration in the knowledge graph, and long path exploration, have not been well solved. Therefore, in this paper, we design a novel path construction method to address these issues.

Proposed method

Aimed at the above problems and challenges, we propose a knowledge preference-aware reinforcement learning network named KPRLN, which extracts fine-grained user interest preference features in the knowledge graph. The overall framework is shown in Fig. 2. The KPRLN model is generally divided into the preference-weighted knowledge graph generation layer and the recommendation prediction layer. In the user preference-weighted knowledge graph generation layer, we construct the path network of user historical interaction items in the knowledge graph based on deep reinforcement learning. The deep reinforcement learning model explores the knowledge graph by cluster expansion and designs feedback rewards based on hierarchical propagation paths. Meanwhile, the deep reinforcement learning agent globally updates the weights of edges in the knowledge graph by the expected returns for each link. In the recommendation prediction layer, we design an attention mechanism to propagate the higher order interest of users in the knowledge graph and aggregate user and item representations for prediction. The notations and descriptions used in this paper are shown in Table 1.

Table 1 Notations and their descriptions used in this paper

Notations	Descriptions
G	Knowledge graph
r_t	The reward at time t
s_t	The state at time t
a_t	The action at time t
G_u	Weighted knowledge graph for user u
E, e	Entities of KG and entity of entities
R, r	Relations of KG and relation of relations
E_u, e_u	Interaction entities of user u
S	Path sequence set
P	Path sequence
f_i	Feature representation of entity e_i
r_i, r_{feedback}	Immediate reward and delayed feedback reward
y_j	Expected value of action j
$\mathbb{D}$	The experience replay

Framework

We first introduce our general idea and the overall structure of our model. The knowledge graph is defined as $G = (E, R)$, where E represents the entity (node) set in the KG, and R represents the relation (edge) set in the KG. The KG is represented in the form of triples as $G = \{e, r, e'|e, e' \in E, r \in R\}$, between entity e and entity e' connected by the relation r. In our deep reinforcement learning model, we learn the user's higher order interest preferences through the user's historical interaction items. The user's historical interaction item set is represented by E_u, and for any $e_u \in E_u$, they, respectively, reflect the user's preference features and relate to each other. However, they are sparsely distributed in the knowledge graph. Therefore, we build a network of paths among them to learn their connection.

Initially, the RL agent will randomly select an e_u as the starting state $s_{t=0}$ to construct a path. We select the neighbors of the last entity in the path as the action range. When another user interaction item has been added to the path, return a positive reward and start a new walking process. Otherwise, return a negative reward and continue to walk. For each path, it is described as follows:

$$S = \begin{cases} p_1 = (e_u \xrightarrow{\text{r}^-} e_1 \xrightarrow{\text{r}^-} \cdots \xrightarrow{\text{r}^-} e_{u'}), r_1^+ \\ p_2 = (e_u \xrightarrow{\text{r}^-} e_2 \xrightarrow{\text{r}^-} \cdots \xrightarrow{\text{r}^-} e_{u'}), r_2^+ \\ \cdots, \end{cases} \tag{1}$$

where S is the path sequence-reward set. We do not add duplicate nodes in p, so p will not be a cycle or a loop.

Then, we extend the representation of user history interaction items to their neighbors. Therefore, the starting and target states represent a node cluster. As the walking process described above, the RL model constructs cluster-to-cluster paths. When the cluster-to-cluster path is found, we back-propagate to the relevant nodes in the starting cluster and link them to all the nodes in the target cluster. Return the rewards based on the hierarchical propagation paths. We formulate the number of extensions based on the size of the knowledge graph and the number of historical interaction items of the users. When the RL model is sufficiently trained, globally generate the weighted graph G_u based on local paths.

Finally, we propagate user preferences on G_u and aggregate the item embedding representation and user embedding representation by GNNs for CTR prediction and Top-K recommendation.

Reinforcement learning guides weighted graph generation

To illustrate the detailed design of our deep reinforcement learning model, we first introduce the detailed design of state, action, and reward.

State: Consists of the topology information of all entities in the current path, and s_t represents a general description of the current path sequence p at step t. We use Node2vec [36] to obtain the entities embedding representations in the knowledge graph as the inputs of state representations in deep reinforcement learning. The embedding representation of entity e_i is f_i, for $p = (e_1, e_2, \ldots, e_t)$, the s_t is represented as follows:

$$s_t = [f_1; f_2; \ldots; f_t], \tag{2}$$

where [;] represents vector concatenation.

We utilize pooling to simplify state input to enhance the efficiency of the reinforcement learning model. Consider that in path p, the last node determines the action range. Therefore, we pool the path except for the last node. The $s_t = [f_1; f_2; \ldots; f_t]$ is pooled as

$$s_t = [\text{max} - \text{pooling}\{f_1; f_2; \ldots f_{t-1}\}; f_t], \tag{3}$$

where max-pooling$\{\cdot\}$ is the pooled representation of the p after removing the last node. And it is concatenated with the embedding of the last node.

Action: The next node to join path p. Define a_t to be an action at time t, a_t is the embedding representation of the entity which adds to p. The action set (neighbors of the last node) removes the nodes already present in p to make sure that p is a real path. The RL agent selects action based on the expected reward of a_t according to $Q(s_t, a_t)$, and updates state s_t to $s_{(t+1)}$. $Q(s_t, a_t)$ is the return reward value predicted by Q-network for action a_t. We will introduce the design of the Q-network later.

Reward: The reward is used for feedback to guide deep reinforcement learning model training. In our model, it is designed into two parts: immediate reward for model training and delayed feedback reward for balancing the immediate reward. We define the immediate reward as r_i, which is obtained by constructing the path network. And the delayed feedback reward is defined as r_{feedback}, determined by the current weighted knowledge graph.

1. **Immediate rewards**: Our model expects to build the path network among the user historical interaction items. Therefore, when another user historical interaction item adds to p, return a positive reward, otherwise return a negative reward

$$r_i = \begin{cases} |d| & if\ e \in E_u \\ -\zeta|d| & \text{otherwise} \end{cases}, \tag{4}$$

where d is a constant and ζ is a balancing hyperparameter.
KPRLN extend user history interaction items along the links in the knowledge graph to their neighbors. When finding a cluster-to-cluster path, the model will backpropagate to the relevant nodes in the starting cluster. Find all potential paths within the starting cluster based on the extended hops. And link these potential paths with the cluster-to-cluster path. In the target cluster, these paths spread outward around the user interaction item center. The hierarchical propagation path reward is designed as follows:

$$r_i^h = \frac{1}{2^h} r_i, \tag{5}$$

where h represents the number of hops. In KPRLN, the reward is halved for each additional hop compared to the original immediate reward.
2. **Delayed feedback reward**: We hope that the weighted knowledge graph can work well in the recommender systems. Therefore, we designed a delayed feedback reward based on the recommendation task. We divide the whole training process into multiple epochs. We sample users' historical interactions in each epoch and make predictions in the current weighted knowledge graph. According to the predicted performance, the r_{feedback} is defined as

$$r_{\text{feedback}} = \mathcal{Z}(\text{scores}(G_u)) r_i \beta, \tag{6}$$

where $scores(\cdot)$ is the user weighted graph model estimate, which is calculated base on the recommended task performance, $\mathcal{Z}(\cdot)$ is a normalization function, and β is a balance hyperparameter.

The design of the Q-network is shown in Fig. 3. Input the current path state s and the next action a to the Q-network. After applying two ReLU layers, the output $Q(s, a)$ represents the expected value of the action a in the s state, as follows:

$$Q(s_t, a_t) = f_\theta([s_t; a_t]), \tag{7}$$

where $f_\theta(\cdot)$ is the deep neural network shown in Fig. 3.

Experience replay enables the Q-network to update parameters with recent experience stored in the replay memory, thus stabilizing the training process. However, it may lead to overestimating and local optimum, as great q-value paths are found repeatedly. Therefore, we use DDQN [37] as our RL framework. Our model first finds the action corresponding to the maximum q-value. Then, calculate the target q-value of the action in the target network. Finally, decoupling the choice of target Q-value action and the calculation of the target Q-value to eliminate the problem of overestimation

$$y_j^Q = r + \gamma Q(s_{j+1}, \text{argmax}_{a'} Q(s_{j+1}, a, \theta), \theta'), \tag{8}$$

where γ is the discount factor, θ is the parameter of the original network, and θ' is the parameter of the target network. Backpropagation updates the parameters in the Q-network by the mean squared loss function

$$L_{\text{RL}}(\theta) = \frac{1}{|\mathbb{D}|} \sum_{(s,a,r,s') \sim \mathbb{D}} [(y_j^Q - Q(s, a_j, \theta))^2], \tag{9}$$

where $|\mathbb{D}|$ represents the number of samples collected in the experience replay pool.

Preference knowledge-aware recommendation

In the deep reinforcement learning layer, the preference-weighted KG G_u is generated for each user based on their historical interactions. We propagate users' interests on G_u to get high-order preference representations of items and users.

Item representation

First, we propagate user preferences along the relations in G_u. To learn more semantic information in the knowledge graph and consider the size of the knowledge graph, we designed an attention graph convolutional network based on [38].

As shown in Fig. 4, we sample the neighbors of the item sequentially according to the values of the edge weights in G_u and aggregate the multi-hop neighbors of items based on this. Then, aggregate item representation based on the attention graph convolutional network

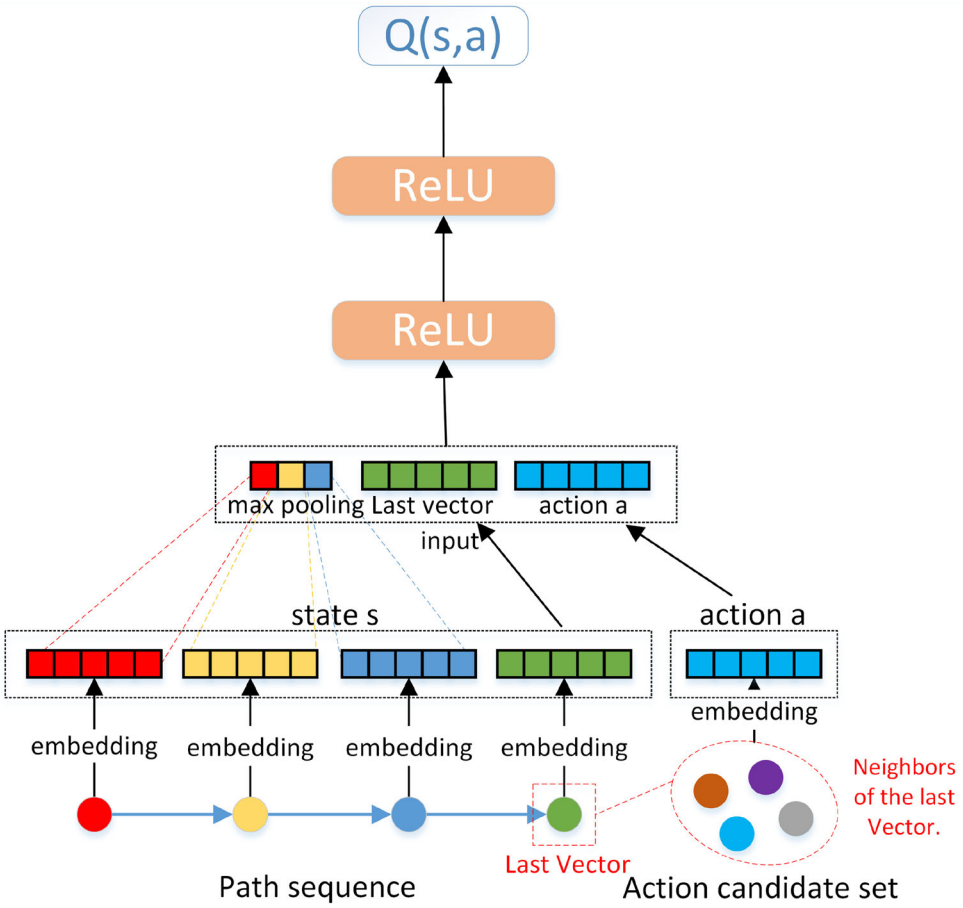


Fig. 3 The structure of the Q-network in our framework. The state s is the embedding of the vertices of the path sequence, and the action a is the neighbor of the last vertex. After pooling, they are used as the input to the Q-network

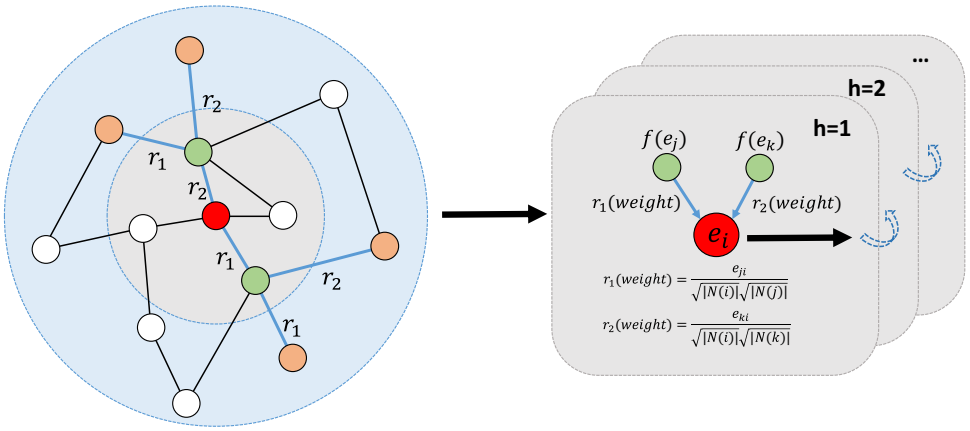


Fig. 4 Item feature aggregation in user weight graph

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j \in N(i)}^{n} \frac{e_{ji}}{c_{ji}} h_j^{(l)} W^{(l)}), \tag{10}$$

where n represents the number of samples, and l is the number of layers in graph convolution, which represents the number of propagation hops. $N(i)$ represents the neighbors of the node i, c_{ji} is the square-root product of node degrees (i.e., $c_{ji} = \sqrt{|N(i)|}\sqrt{|N(j)|}$), $\sigma(\cdot)$ is an activation function, and e_{ji} is the scalar weight from node j to node i.

User representation

Considering that the item embedding already contains the user's preference features, we associate the users with their historical interaction items.

Specifically, we build the user–item bipartite graph and aggregate the features of user interaction items to get user embedding representation, which can be described as

$$\mathbf{u} = f_{\text{agg}}(\mathbf{i}_u), \tag{11}$$

where $\mathbf{i}_u$ represents the interaction item embedding of user u, which is aggregated in G_u. And, $f_{\text{agg}}(\cdot)$ is a function for aggregating the user embedding representation.

Learning algorithm

We predict the interact probability between the user and the item based on user embedding u and item embedding v

$$\hat{y}_{uv} = \mathcal{F}(u, v). \tag{12}$$

In our recommender system, we iterate over all possible user–item pairs by negative sampling strategy. The loss is calculated as

$$L_{\text{RS}} = \sum_{u,v} \mathcal{J}(y_{uv}, \hat{y}_{uv}) - \sum \sigma(b^{(l)} + G_u * W^{(l)}) + \lambda ||\Theta||_2^2, \tag{13}$$

where $\mathcal{J}(\cdot)$ is the cross-entropy loss, the second term is the item aggregation loss, $||\Theta||_2^2$ is the L2-regularization loss function, and λ is the balance hyperparameter.

The process of KPRLN is described as Algorithm 1. It mainly consists of two parts: (1) generate the user preference-weighted knowledge graph; (2) aggregating users' higher order interest preferences under the GNNS framework.

Algorithm 1 KPRLN algorithm

Require: Knowledge graph $\mathcal{G}$, interaction matrix Y;
Ensure: Prediction function $\mathcal{F}(u, v|\Theta, Y, \mathcal{G})$
Initialize all parameters
while KPRLN not converge **do**
  for user in Users **do**
    put user interaction items in RL model
    while number in training iteration **do**
      Get state s_t
      Select a_t according to the policy network
      Interact with RL agent
      Get immediate reward r_i
      if $|step\ t| = |L|$ **then**
        Reconstruct weight graph
        Get $r_{feedback}$
      end if
      Get reward r_t,state s_{t+1},path p_t
      Store transition(s_{t+1}, a_t, r) in $\mathbb{D}$
      Update value and policy network
      Update state $s_t = s_{t+1}$
      Update the parameters by gradient descent
    end while
    Generate user weight graph $\mathcal{G}_u$
  end for
  Aggregate items embedding and users embedding
  Calculate predicted probability $\hat{y}_{uv}$
  Update parameters of $\mathcal{F}$
end while
Return $\mathcal{F}(u, v|\Theta, Y, \mathcal{G})$

Table 2 Detailed statistics of the three datasets

	Movielens-1 M	Last.FM
Users	6036	1872
Interactions	753,772	42,346
Items	2347	3846
Relations	12	60
Entities	6729	9366
KG triples	20,195	15,518

Experiments

In this section, we show the performance of KPRLN. We evaluate our model on two real-world scenarios: Movielens-1 M and Last.FM, and compare it with state-of-the-art methods. First, we introduce the experimental setup, including datasets and baselines. Second, compare with other baselines and model variants under the same scenario. Then, we discuss the impact of hyperparameters on model performance. Finally, we show a case on the movie dataset, demonstrating that KPRLN can provide reasonable explanations for users' preferences on recommendations.

Datasets

We use datasets based on real scenarios as follows:

1. Movielens-1 M[1] is a widely used movie dataset. It is smaller than Movielens-20 M and contains about 1 million ratings.
2. Last.FM[2] is a widely used music dataset that contains data from Last.FM. Information from over 2000 users of the online music system.

Since these datasets are explicitly fed back, we convert them to implicit feedback by setting a rating threshold, marking all entries larger than the threshold as 1, indicating that the user is satisfied, and sampling unsatisfactory ones marked as 0 for each user matching set. And we removed users who did not include positive implicit feedback.

The Movielens-1 M includes 6036 users and 753,772 interactions, and the knowledge graph contains 2347 items, 6729 entities, and 20,195 triples. The Last.FM includes 1872 users and 42,346 interactions, and the knowledge graph contains 3846 items, 9366 entities, and 15,518 triples. The basic statistics of the two datasets are shown in Table 2.

The knowledge graph of Last.FM is published by [7], and the knowledge graph of Movielens-1 M is published by [39].

[1] https://grouplens.org/datasets/movielens/1m/.

[2] https://grouplens.org/datasets/hetrec-2011/.

Table 3 Hyper-parameters' setting

	d	η	N	H	λ	Batch size
Movielens-1 M	64	1×10^{-3}	4	2	1×10^{-5}	1024
Last.FM	16	5×10^{-4}	8	1	1×10^{-4}	256

Baselines

We use the following state-of-the-art baselines for comparison with KPRLN.

1. LibFM [40] is a feature-based factorization model in CTR scenarios. We concatenate user ID and item ID as input for LibFM.
2. PER [18] connections between users and items are captured by extracting meta-path-based features in heterogeneous networks. We use the properties of items as features to build the meta-path between the user and the item.
3. CKE [16] based on the embedding method, which combines collaborative filtering (CF) with structural information, textual information, and visual information in a unified recommendation framework. In this paper, CF is used in conjunction with the structural knowledge module to implement CKE.
4. RippleNet [20] is a method of obtaining links in the knowledge graph in the form of water wave diffusion. Expand users' potential interests through multiple links. In the recommender system, users' interests can be more comprehensively reflected.
5. KGCN [7] is an end-to-end framework that effectively captures inter-item correlations by mining relevant attributes on the knowledge graph. Calculate the scores of users and relations, and use the links on the item to propagate the user's potential interest on the knowledge graph.
6. HAGERec [41] emphasizes the importance of characterizing semantic information of relations, which explores users' potential preferences from the high-order connectivity structure of the heterogeneous knowledge graph, combining graph convolutional networks for explainable recommendation.

Experiments setup

The hyperparameter statistics of our experiments are shown in Table 3. The hyperparameters are as follows: d represents the embedding dimension, H represents the number of item propagation hops, N represents the number of aggregation domains, λ represents the L2-regularization weight, and η represents the learning rate.

The training, evaluation, and test sets ratio for each dataset is 8:1:1. Each experiment was repeated three times, and the average performance was reported. We evaluate model performance using the following two experimental scenarios: (1) CTR prediction. We use the model to predict click probabilities for items in the test set. We use ACC (Accuracy), AUC (Area Under Curve), and F1 to evaluate the performance of CTR prediction. (2) Top-K recommendation. We select the K items with the highest predicted click probability for users in the test set and then select Precision@K and Recall@K to evaluate the recommended set. We use the Adam algorithm to optimize all training parameters. The code for KPRLN is implemented under Python 3.7, Tensorflow 1.14.0, and Numpy 1.21.5.

Results

Performance comparisons with baselines

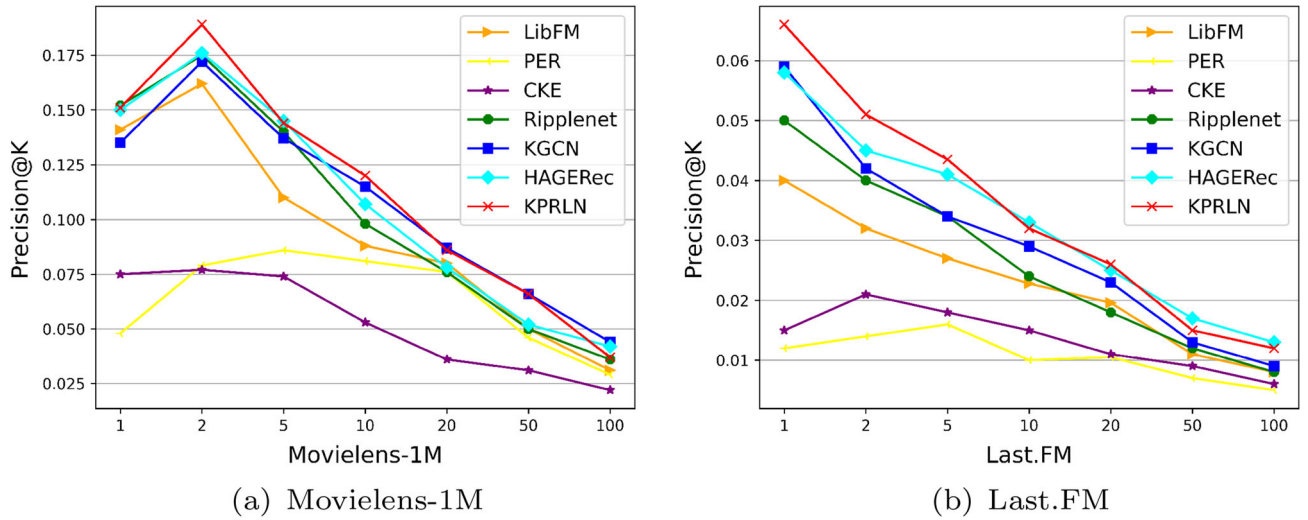
We present the results of CTR prediction and top-K recommendation of KPRLN and other baselines in Table 4 and Figs. 5 and 6, respectively, and draw the following conclusions:

1. In general, KPRLN has the best performance on the recommendation scenarios of the two datasets. As shown in Table 4, in Movielens-1 M, the average improvement in AUC, ACC, and F1 is 7.5%, 6.55%, and 6.37%, respectively. In the Last.FM, the average improvement in AUC, ACC, and F1 is 6.8%, 6.22%, and 5.77%. Furthermore, KPRLN also performs well in Precision@K, Recall@K, as shown in Figs. 5 and 6, demonstrating the efficacy of KPRLN in learning users' high-order interest preferences.
2. PER does not perform well. Because the meta-path we designed is difficult to achieve optimally in movie and music recommendation scenarios, we need a lot of expertise to design meta-paths. This makes it difficult for PER to be optimal in results. Compared with other baselines, CKE performs relatively poorly, which may be because the learning of image features and text features is introduced into the original CKE model, while only the knowledge structure features are in the process of our construction.
3. Ripple and KGCN are unified methods that integrate the semantic representation of entities and relations and the connectivity information base on GNN. However, none of them are well designed to learn the user fine-grained preference interest for each user–item–relation triple. Therefore, they do not perform as well as KPRLN.
4. HAGERec performs the best in all baselines, which uses the attention mechanism to filter aggregated neighbors and designs an interaction signals unit to make GCN characterize more passed information from the network

Table 4 Performance comparisons with baselines

Dataset	Methods	AUC	Impr	ACC	Impr	F1	Impr
Movielens-1 M	LibFM	0.892	4.3%	0.812	4.5%	0.819	4.0%
	PER	0.710	22.5%	0.664	19.3%	0.673	18.6%
	CKE	0.801	13.4%	0.742	11.5%	0.742	11.7%
	Ripplenet	0.921	1.4%	0.844	1.3%	0.848	1.1%
	KGCN	0.913	2.2%	0.840	1.7%	0.843	1.6%
	HAGERec	0.923	1.2%	0.847	1.0%	0.847	1.2%
	KPRLN	**0.935**	–	**0.857**	–	**0.859**	–
Last.FM	LibFM	0.769	5.5%	0.711	4.2%	0.710	3.7%
	PER	0.633	19.1%	0.596	15.7%	0.596	15.1%
	CKE	0.744	8.0%	0.673	8.0%	0.673	7.4%
	Ripplenet	0.780	4.4%	0.691	6.2%	0.702	4.5%
	KGCN	0.796	2.8%	0.731	2.2%	0.721	2.6%
	HAGERec	0.814	1.0%	0.743	1.0%	0.734	1.3%
	KPRLN	**0.824**	–	**0.753**	–	**0.747**	–

Best results are bolded



(a) Movielens-1M (b) Last.FM

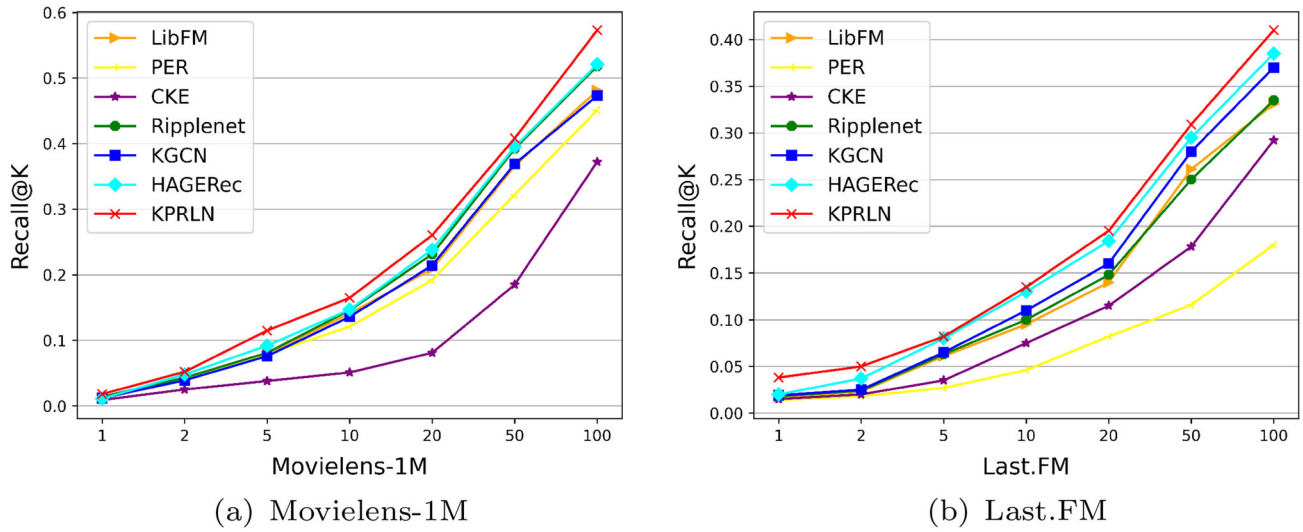
Fig. 5 The results of Precision@K in top-K recommendation

of a central entity. It demonstrates that effectively distinguishing users' preferences for items can improve the performance of the recommender system.

Ablation study

We conduct ablation experiments on the KPRLN to analyze the effect of different components. To demonstrate the improvement of the performance of the recommendation system by the generated weighted knowledge graph, we compare the weighted knowledge graph generated by KPRLN with the unweighted knowledge graph (average aggregate neighbors), and the result is shown in Table 5. Furthermore, we verified the impact of the hierarchical propagation paths in the deep reinforcement learning training on the model's performance. The results are shown in Figs. 7, and 8, and the following conclusions are drawn:

1. As shown in Table 5, KPRLN performs better than the average aggregate method, which proves that user preference information can improve the performance of the recommender systems, and KPRLN can effectively learn user preferences.
2. As shown in Fig. 7, we train each dataset 10,000 times to ensure model convergence and use the comprehensive indicators of AUC, ACC, and F1 to determine the performance of the recommender systems. We find that the performance of KPRLN is proportional to the number of training sessions of the model, and the model is stable in the late training period.



(a) Movielens-1M (b) Last.FM

Fig. 6 The results of Recall@K in top-K recommendation

Table 5 User weight and none weight

	Movielens-1 M			Last.FM		
	AUC	ACC	F1	AUC	ACC	F1
Average	0.907	0.830	0.835	0.798	0.723	0.712
KPRLN	**0.935**	**0.857**	**0.859**	**0.824**	**0.753**	**0.747**

Best results are bolded

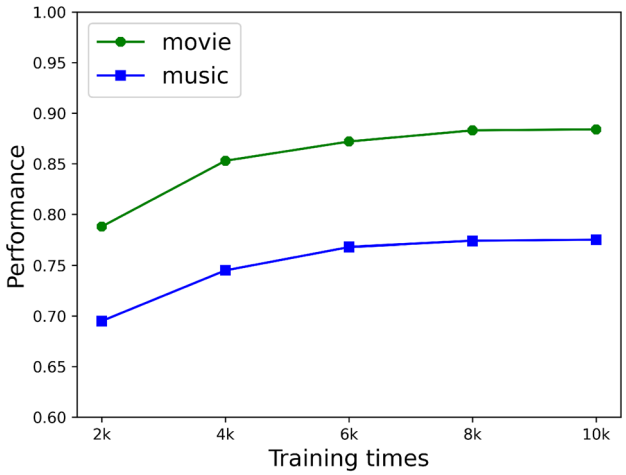


Fig. 7 Performance in different training times

3. $KPRLN\_att^{-}$ is a variant of the KPRLN that removes the hierarchical propagation paths in deep reinforcement learning. As shown in Fig. 8, in recommender systems, KPRLN not only has better performance than $KPRLN\_att^{-}$ but is also more efficient than $KPRLN\_att^{-}$.

Research on noise resistance

KPRLN generated the users' weighted knowledge graphs, so that we could remove small weight edges in the knowledge graph. We have removed 5%, 10%, and 20% of edges (as noises) in ascending order of edge weight value and compared the recommendation performance. As shown in Fig. 9, the following conclusions are drawn:

1. In Movielens-1 M, KPRLN gets the best performance in the original knowledge graph, and the performance gradually decreases as the remove ratio increases. In Last.FM, KPRLN achieves the best performance in the 5% removed knowledge graph, which proves that the edges we removed are useless noise.
2. In Last.FM, the performance of removing 20% is better than removing 10%. The reason may be that the KG of Last.FM is more sparse than the KG of Movielens-1 M and the KG of Last.FM has 60 relation types which are more than Movielens-1 M. Therefore, the KG of Last.FM may contain more useless information, which makes removing 20% edges can make the model focus better on useful information than removing 10% edges.

Performance comparisons with hyperparameters

We compare the effect of item aggregation domain number N and item propagation hop number H on the model.

(1) The performance on different N.

Knowledge graph has rich semantic information that enriches the representation of items. We emphasize that user preferences for entity–relation pairs affect the performance of the recommender system and propagate users' preferences on

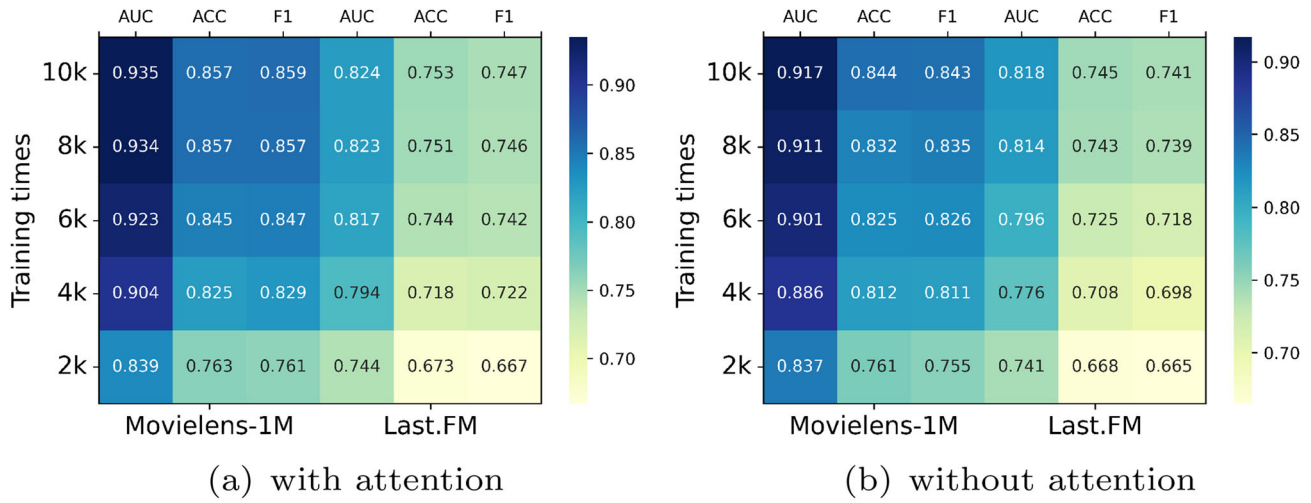


Fig. 8 The impact of attention mechanism on the model

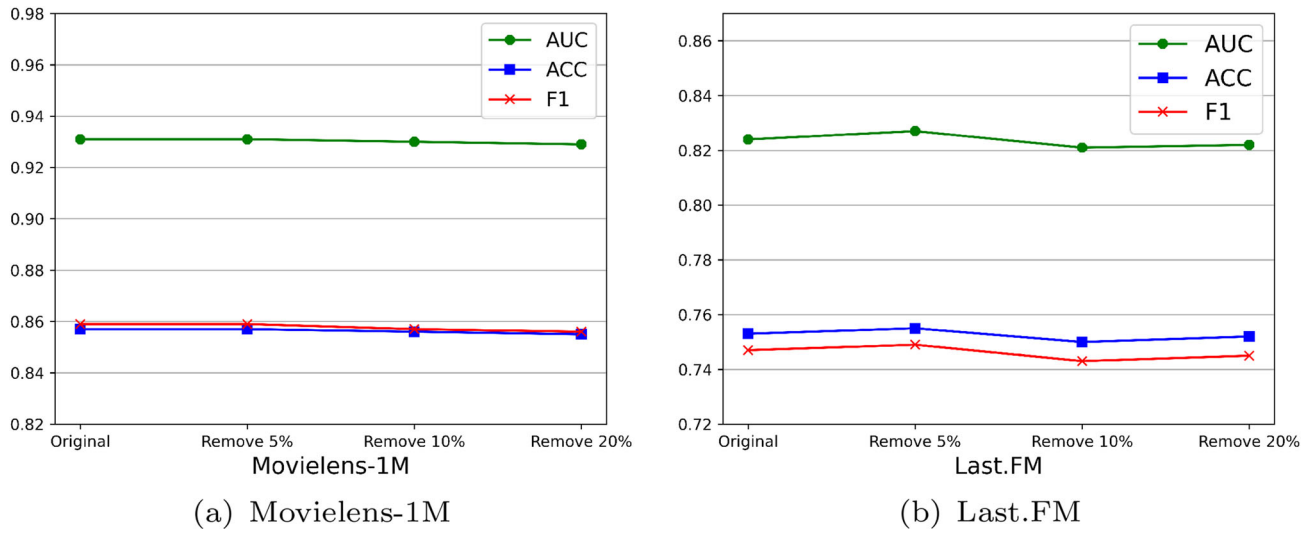


Fig. 9 The results of noise experiment

the knowledge graph based on the attention mechanism. The hyperparameter N represents the number of neighbors we sampled. Therefore, we need to discuss the impact of N on KPRLN.

- The results are shown in Fig. 10. KPRLN shows the best performance at $N = 4$ in Movielens-1 M and achieves the best result in Last.FM when $N = 8$. This is because if N is too small, it does not contain enough neighbor information, and if N is too large, the model performance is susceptible to noise. It should be noted that the number of neighbors for some items may be less than N, in which case we select all neighboring entities.

(2) The performance on different H.
The number of neighbor propagation hops H is also very critical, and the size of the propagation hops determines the range through entity information. Therefore, it is important to ensure the appropriate number of propagation hops.

- The results are shown in Fig. 11. In Movielens-1 M, KPRLN achieves the best performance when $H = 2$, but in Last.FM, the best performance is obtained when $H = 1$. The number of entities aggregated to the item increases exponentially with H, which makes H more sensitive than N. In Movielens-1 M, we can get more information in the longer relation chain, while Last.FM is relatively sparse, so too large H brings more noise to

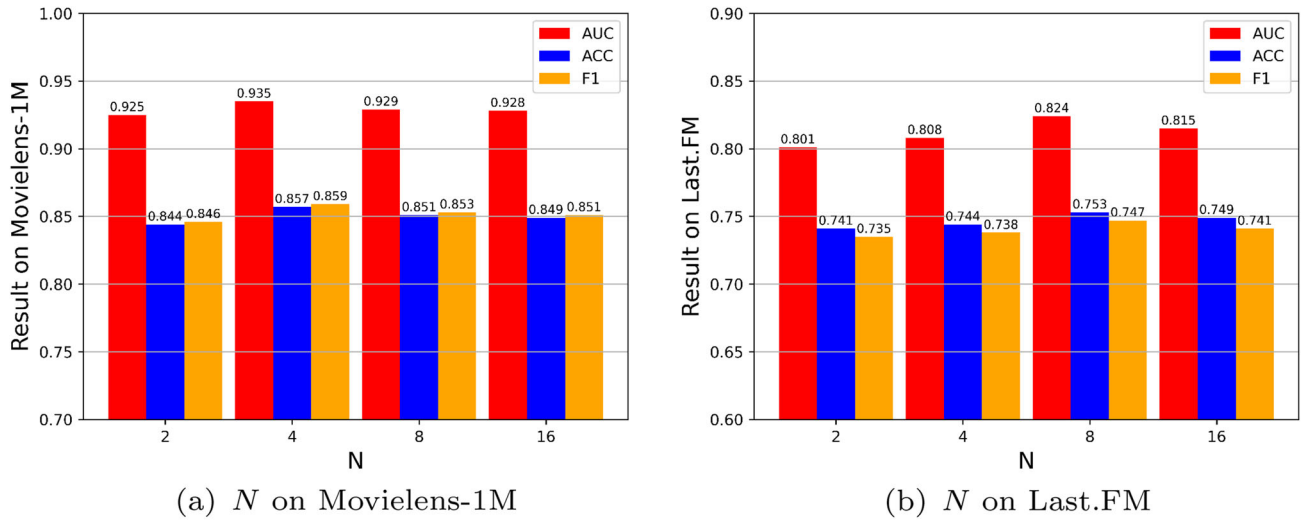


Fig. 10 The results of hyperparameters N on datasets

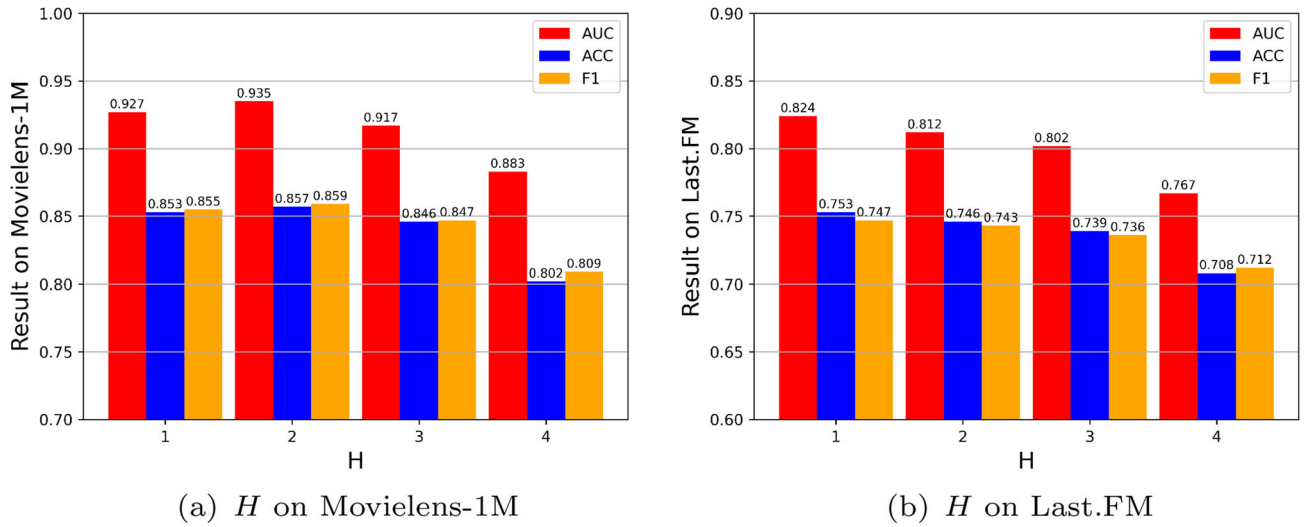


Fig. 11 The results of hyperparameters H on datasets

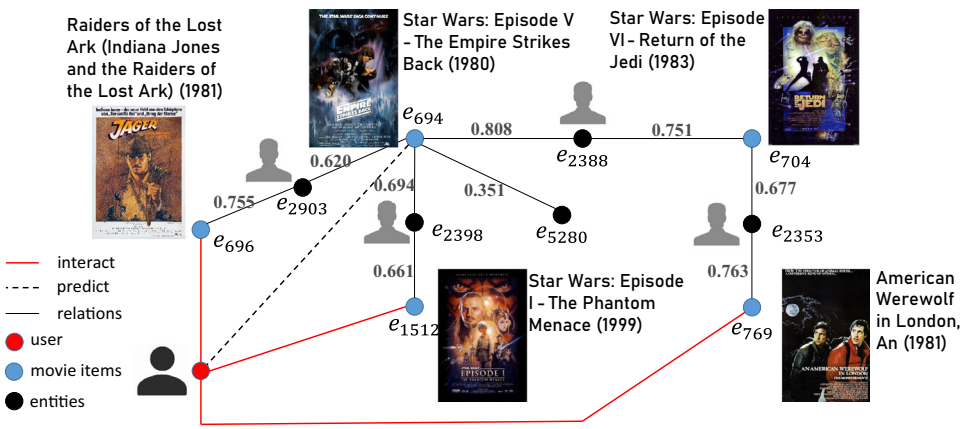


Fig. 12 The local structure of the preference-weighted knowledge graph, which shows an example of movie recommendation. Establish the reasonable paths between users and predicted items based on the weight values, and explain the recommendations

the model. In addition, the performance of KPRLN is more stable in Movielens-1 M than in Last.FM.

Case study

We select a real example from Movielens-1 M to intuitively demonstrate the effectiveness of KPRLN. We randomly select a user–item pair from the test dataset, and the item (e_{694}) is treated as a target item that would be recommended for the user. Then, KPRLN generates the user preference weight knowledge graph based on the user's interaction items. The movies used for model training are Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981), Star Wars: Episode I—The Phantom Menace (1999), and American Werewolf in London, An (1981). The movie used for prediction is Star Wars: Episode V-The Empire Strikes Back (1980). As shown in Fig. 12, the weights of the edges in the graph represent the user's preference, and the edges which in the path between the interaction items can get higher weights. Therefore, the model can learn more useful information when aggregating the representation of e_{694}. The edge weight between entity e_{5280} and predicted item e_{694} is relatively low, because e_{5280} is not associated with the user's historical interaction items.

Conclusions and future work

This paper proposes a knowledge graph recommender system based on deep reinforcement learning (KPRLN). In the deep reinforcement learning model, we design hierarchical propagation paths to establish associations between users' historical interaction items and learn the features of users' preferences for entities and relations of KG. At the same time, coordinated by different reward mechanisms, the preference-weighted KG is generated for each user. Then, more influential neighbors are sampled based on an attention mechanism to propagate users' preferences on the KG, aggregating to get embedding representations of items and users. Our method is not to learn users' preferences for various relations at a macro-level but to learn in detail about the user and specific entity–relation–entity combinations. And demonstrate excellent performance on widely used real-world datasets, achieving significant progress compared to several state-of-the-art baselines.

Our future work intends to evaluate the effectiveness of our model on more real-world data.

Acknowledgements This work is supported by the National Natural Science Foundation of China (Grant No. 62266054), and Science and Technology Program of Yunnan Province (Grant No. 202101AT070 095).

Author Contributions DW: conceptualization, methodology, software, formal analysis, investigation, visualization, writing-original draft. MT: conceptualization, funding acquisition, writing-review and editing. SZ: writing-review and editing. AY: data curation, validation. WG: investigation, supervision.

Data Availability Data available on request from the authors.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Wang S, Hu L, Wang Y, He X, Sheng QZ, Orgun MA, Cao L, Ricci F, Yu PS (2021) Graph learning based recommender systems: a review. arXiv:2105.06339
2. Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: a survey of approaches and applications. IEEE Trans Knowl Data Eng 29:2724–2743
3. Liu J, Duan L (2021) A survey on knowledge graph-based recommender systems. In: 2021 IEEE 5th advanced information technology, electronic and automation control conference (IAEAC), Chongqing, vol 5. pp 2450–2453
4. Wang X, Wang D, Xu C, He X, Cao Y, Chua T-S (2019) Explainable reasoning over knowledge graphs for recommendation. In: Proceedings of the AAAI conference on artificial intelligence, Hawaii, vol 33. pp 5329–5336
5. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: International conference on learning representations. https://arxiv.org/abs/1710.10903
6. Wang X, He X, Cao Y, Liu M, Chua T-S (2019) Kgat: knowledge graph attention network for recommendation. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining, Anchorage. pp 950–958. https://doi.org/10.1145/3292500.3330989
7. Wang H, Zhao M, Xie X, Li W, Guo M (2019) Knowledge graph convolutional networks for recommender systems. In: The world wide web conference, San Francisco. pp 3307–3313. https://doi.org/10.1145/3308558.3313417
8. Wang H, Zhang F, Zhang M, Leskovec J, Zhao M, Li W, Wang Z (2019) Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining, Anchorage. pp 68–977

9. Huai Z, Tao J, Che F, Yang G, Zhang D (2021) Knowledge graph enhanced recommender system. arXiv preprint arXiv:2112.09425
10. Hui B, Zhang L, Zhou X, Wen X, Nian Y (2022) Personalized recommendation system based on knowledge embedding and historical behavior. Appl Intell 52:954–966
11. Afsar MM, Crump T, Far B (2022) Reinforcement learning based recommender systems: a survey. ACM Comput Surv. https://doi.org/10.1145/3543846
12. Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O (2013) Translating embeddings for modeling multi-relational data. Adv Neural Inf Process Syst 2:2787–2795. https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf
13. Wang Z, Zhang J, Feng J, Chen Z (2014) Knowledge graph embedding by translating on hyperplanes. In: Proceedings of the AAAI conference on artificial intelligence, Québec City, vol 28
14. Balazevic I, Allen C, Hospedales T (2019) Multi-relational poincaré graph embeddings. Adv Neural Inf Process Syst 4460–4470
15. Xu C, Li R (2019) Relation embedding with dihedral group in knowledge graph. In: Proceedings of the 57th annual meeting of the association for computational linguistics, Florence. pp 263–272
16. Zhang F, Yuan NJ, Lian D, Xie X, Ma W-Y (2016) Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. KDD'16, San Francisco. pp 353–362
17. Wang H, Zhang F, Xie X, Guo M (2018) Dkn: Deep knowledge aware network for news recommendation. In: Proceedings of the 2018 World Wide Web Conference. WWW '18, Lyon. pp 1835–1844
18. Yu X, Ren X, Sun Y, Gu Q, Sturt B, Khandelwal U, Norick B, Han J (2014) Personalized entity recommendation: A heterogeneous information network approach. In: Proceedings of the 7th ACM international conference on web search and data mining, New York. pp 283–292
19. Zhao H, Yao Q, Li J, Song Y, Lee DL (2017) Meta-graph based recommendation fusion over heterogeneous information networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, Halifax. pp 635–644
20. Wang H, Zhang F, Wang J, Zhao M, Li W, Xie X, Guo M (2018) Ripplenet: propagating user preferences on the knowledge graph for recommender systems. In: Proceedings of the 27th ACM international conference on information and knowledge management, Torino. pp 417–426
21. Wang X, Wang S, Liang X, Zhao D, Huang J, Xu X, Dai B, Miao Q (2022) Deep reinforcement learning: a survey. IEEE Trans Neural Netw Learn Syst 1–15
22. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236
23. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489. https://doi.org/10.1038/nature16961
24. Codevilla F, Müller M, López A, Koltun V, Dosovitskiy A (2018) End-to-end driving via conditional imitation learning. In: 2018 IEEE international conference on robotics and automation (ICRA), Brisbane. pp 4693–4700
25. Shani G, Heckerman D, Brafman RI (2005) An mdp-based recommender system. J Mach Learn Res 6:1265-1295
26. Hu B, Shi C, Liu J (2017) Playlist recommendation based on reinforcement learning. In: Intelligence Science I: Second IFIP TC 12 International Conference (ICIS), Shanghai. pp 172–182
27. Zheng G, Zhang F, Zheng Z, Xiang Y, Yuan NJ, Xie X, Li Z (2018) Drn: a deep reinforcement learning framework for news recommendation. In: Proceedings of the 2018 world wide web conference, Lyon. pp 167–176
28. Zhao X, Xia L, Zhang L, Ding Z, Yin D, Tang J (2018) Deep reinforcement learning for page-wise recommendations. In: Proceedings of the 12th ACM Conference on Recommender Systems, Vancouver. pp 95–103
29. Karimi M, Jannach D, Jugovac M (2018) News recommender systems: survey and roads ahead. Inf Process Manag 54(6):1203–1227. https://doi.org/10.1016/j.ipm.2018.04.008
30. Wang Q, Ji Y, Hao Y, Cao J (2020) Grl: knowledge graph completion with gan-based reinforcement learning. Knowl Based Syst 209:106421
31. Xiong W, Hoang T, Wang WY (2017) Deeppath: a reinforcement learning method for knowledge graph reasoning. arXiv preprint arXiv:1707.06690
32. Das R, Dhuliawala S, Zaheer M, Vilnis L, Durugkar I, Krishnamurthy A, Smola A, McCallum A (2018) Go for a walk and arrive at the answer: reasoning over paths in knowledge bases using reinforcement learning. In: International conference on learning representations. https://arxiv.org/abs/1711.05851
33. Lin XV, Socher R, Xiong C (2018) Multi-hop knowledge graph reasoning with reward shaping. In: EMNLP. https://arxiv.org/abs/1711.05851
34. Xian Y, Fu Z, Muthukrishnan S, De Melo G, Zhang Y (2019) Reinforcement knowledge graph reasoning for explainable recommendation. In: Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval, Paris. pp 285–294
35. Tao S, Qiu R, Ping Y, Ma H (2021) Multi-modal knowledge-aware reinforcement learning network for explainable recommendation. Knowl Based Syst 227:107217
36. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco. pp 855–864
37. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence, Phoenix, vol 30
38. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. Adv Neural Inf Process Syst 1025–1035
39. Wang H, Zhang F, Zhao M, Li W, Xie X, Guo M (2019) Multi-task feature learning for knowledge graph enhanced recommendation. In: The world wide web conference, WWW '19. San Francisco. pp 2000–2010
40. Rendle S (2012) Factorization machines with libfm. ACM Trans Intell Syst Technol (TIST) 3(3):1–22. https://doi.org/10.1145/2168752.2168771
41. Yang Z, Dong S (2020) Hagerec: hierarchical attention graph convolutional network incorporating knowledge graph for explainable recommendation. Knowl Based Syst 204:106194. https://doi.org/10.1016/j.knosys.2020.106194