Automatic Meta-Path Discovery for Effective Graph-Based Recommendation

Wentao Ning
The University of Hong Kong
Southern University of Science and Technology
Hong Kong and Shenzhen, China
wtning@cs.hku.hk

Reynold Cheng*
The University of Hong Kong
Hong Kong, China
cckcheng@cs.hku.hk

Jiajun Shen
TCL Research
Hong Kong, China
sjj@tcl.com

Nur Al Hasan Haldar
The University of Western Australia
Perth, Australia
nur.haldar@uwa.edu.au

Ben Kao
The University of Hong Kong
Hong Kong, China
kao@cs.hku.hk

Xiao Yan
Southern University of Science and Technology
Shenzhen, China
yanx@sustech.edu.cn

Nan Huo
The University of Hong Kong
Hong Kong, China
huonan@connect.hku.hk

Wai Kit Lam
TCL Research
Hong Kong, China
tian23.li@tcl.com

Bo Tang†
Southern University of Science and Technology
Shenzhen, China
tangb3@sustech.edu.cn

ABSTRACT

Heterogeneous Information Networks (HINs) are labeled graphs that depict relationships among different types of entities (e.g., users, movies and directors). For HINs, meta-path-based recommenders (MPRs) utilize meta-paths (i.e., abstract paths consisting of node and link types) to predict user preference, and have attracted a lot of attention due to their explainability and performance. We observe that the performance of MPRs is highly sensitive to the meta-paths they use, but existing works manually select the meta-paths from many possible ones. Thus, to discover effective meta-paths automatically, we propose the Reinforcement learning-based Meta-path Selection (RMS) framework. Specifically, we define a vector encoding for meta-paths and design a policy network to extend meta-paths. The policy network is trained based on the results of downstream recommendation tasks and an early stopping approximation strategy is proposed to speed up training. RMS is a general model, and it can work with all existing MPRs. We also propose a new MPR called RMS-HRec, which uses an attention mechanism to aggregate information from the meta-paths. We conduct extensive experiments on real datasets. Compared with the manually selected meta-paths, the meta-paths identified by RMS consistently improve recommendation quality. Moreover, RMS-HRec outperforms state-of-the-art recommender systems by an average of 7% in hit ratio. The codes and datasets are available on https://github.com/Stevenn9981/RMS-HRec.

CCS CONCEPTS

• Information systems → Recommender systems; Collaborative filtering.

KEYWORDS

reinforcement learning, meta-path, graph neural network.

1 INTRODUCTION

A Heterogeneous Information Network (HIN) [15, 17, 39, 48] consists of multiple types of entities and links (or relations), and Figure 1 illustrates an example of movie HINs. Due to its ability to model the complex interactions among different entities, HINs are widely used in fields such as natural language processing [38, 50], community detection [6, 35], and bioinformatics [9, 48]. HINs have also been used to support recommender systems. HINs provide
Figure 1: An example of a movie HIN. It contains four node types (i.e., user, movie, actor, and director) and three relationship types (i.e., watched, acted, directed) among the nodes.

more information than user-item interaction logs commonly utilized by recommenders, and have attracted plenty of attention recently [1, 4, 19].

Meta-path, which consists of a sequence of relations between different node types [32], is often utilized to extract information from HINs for recommendation. For example, in Figure 1, Movie → Actor → Movie (abbreviated as MAM) is a meta-path and can be adopted to find movies having the same actor. A few meta-path-based recommenders (MPRs) have been proposed in recent years [11, 19, 45]. For example, HERec [30] collects random walk samples according to meta-paths to train user and item embeddings. MCRec [13] utilizes a convolutional layer to train meta-path representations based on their sampled path instances. MEIRec [5] utilizes meta-paths to select related node neighbors and fuses information from them to obtain node embeddings. These MPRs achieved better performance compared with existing recommenders, and are favored as they can also provide the explainability [7, 8, 16, 33] to the recommendation results. For example, MCRec [13] learns the weights of meta-paths for each user-item pair, if the weight of meta-path UAM (refer to Figure 1) is the largest among all the meta-paths for a user-item pair in the recommendation results, the reason that the specific movie is recommended to the user is more likely because he has ever seen many movies starring an actor who also stars in this movie.

Existing MPRs require a set of meta-paths as input. Our experiments reveal that the performance of MPRs is highly sensitive to the choice of meta-paths. For example, on the Yelp dataset (Section 5), the use of different meta-path sets can result in a significant performance difference in recommendation. For example, Table 1 shows that meta-path set 2 yields a recommendation quality gain of 17.7% over meta-paths set 1.

Although using the right set of meta-paths is crucial to the performance of MPRs, the issues of finding an effective meta-path set for recommendation have not been well addressed in the literature. Existing MPR works often claim that domain experts’ knowledge can be used to obtain meta-paths. The fact is that selecting correct meta-paths for recommendation is very challenging. First, due to the complexity of an HIN, the search for the optimal meta-paths is highly expensive – the complexity of finding \( m \) length-1 meta-paths for \( n \) node types is exponential (i.e., \( O(n^m) \)). Second, our experiments show that for a given HIN, the best meta-path set varies across different MPRs; a meta-path set that is the best for an MPR may not work well for another MPR. Third, the structures of the meta-paths required by different MPRs can vary. While some recommendation models need meta-paths that start with a user type and end with an item type [5, 13], others need meta-paths that both start and end with a user or item type [9, 30]. These observations call for better methods for discovering meta-paths automatically.

To address these issues, we propose a model-independent Reinforcement learning (RL)-based Meta-path Selection (RMS) framework to figure out the meaningful meta-paths for recommendation and it is the first framework that can be directly applied to existing MPRs. We train a policy network (agent) to automatically generate a relation to extend the current meta-path set to take the place of the human intervention in the designing of meta-path sets and use the encoding of the current meta-path set as input. We optimize the meta-path selection policy network by exploiting the performance gain of the newly generated meta-path set from the downstream recommender. In addition, to incorporate the meta-path selection in the recommendation tasks, we also propose a new MPR dubbed HRec and obtain an advanced algorithm RMS-HRec by integrating HRec with RMS. HRec generates user/item meta-path subgraphs via the generated meta-paths and then obtains the embeddings via a graph attention network. Then another attention mechanism is utilized to fuse the user/item embeddings from each meta-path.

Specifically, the above-mentioned procedures are non-trivial due to the following challenges. 1) It is arduous to get the performance gain of each meta-path set from a large number of possible meta-path sets because it will result in a considerable time consumption to train the RL model. 2) The huge training time and memory consumption of HRec make direct training impossible on the real-world graphs. To deal with these challenges, inspired by the hyperparameter optimization techniques [3, 23, 31], we propose an early stopping approximation technique, namely testing meta-path sets on lightly trained models, to ease the training time burden (Challenge 1, details in Section 3.3). Additionally, we also propose some strategies to control the density of generated meta-path subgraphs to avoid large GPU memory usage for more effective training of HRec (Challenge 2, details in Section 4). Our experiments show the meta-path set selected by RMS can improve accuracy by more than 20% in some cases compared to other strategies, and RMS can find some meta-paths that are effective but not obvious. Also, RMS-HRec performs better than other state-of-the-art recommenders within a comparable training overhead.

The main contributions of this work are as below:

- We propose a model-independent meta-path selection framework RMS, which can be plugged into any meta-path-based recommendation model, and an early stopping strategy to make the training more efficient.
- Equipped with RMS, we develop a new meta-path-based recommender RMS-HRec and design training strategies to make the training process more efficient and scalable.
We conduct extensive experiments to evaluate RMS and RMS-HRec. Experimental results demonstrate the performance of a recommender is very sensitive to the input meta-paths and our proposed models can discover important meta-paths that are easily neglected during manual designing.

The rest of the paper is organized as follows. Section 2 gives some basic definitions. Section 3 elaborates on the core components of our RMS model in detail. Section 4 introduces our proposed recommender HRec. Section 5 proves the effectiveness of our methods by extensive experiments. Section 6 describes the related work. Section 7 draws a conclusion.

2 PRELIMINARY
In this section, we introduce some necessary definitions [32, 44].

Definition 2.1. (Heterogeneous Information Network) A Heterogeneous Information Network (HIN) is a graph, denoted as $G = (V, E, N, R)$, with multiple types of nodes and edges. $V$ represents the node set, which is associated with a node type mapping function $\zeta : V \rightarrow N$, where $N$ is the node type set. $E$ denotes the edge set, which is also associated with an edge type mapping function $\psi : E \rightarrow R$, where $R$ denotes the relation type set.

Example: Figure 1 shows an example of HIN, which consists of four types of nodes (i.e., $N = \{\text{user}, \text{movie}, \text{actor}, \text{director}\}$) and three types of relations/edges (i.e., $R = \{\text{watch/watched}, \text{act/acted}, \text{direct/directed}\}$).

Definition 2.2. (Meta-path) Given an HIN $G$, a meta-path $M$ is a sequence of node types in the form of $n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_l$ (abbreviated as $n_1 n_2 \cdots n_l$), where $n_i \in N$ denotes the node type.

Example: User $\rightarrow$ Movie $\rightarrow$ Actor $\rightarrow$ Movie (abbreviated as $U/M/A/M$) is a meta-path in Figure 1.

Definition 2.3. (Meta-path Instances) Given an HIN $G$ and a meta-path $M$, a meta-path instance $I^M_i$ is defined as a node sequence in $G$ following the meta-path $M$.

Example: Suppose meta-path $M = UMAM$ and $G$ is the HIN shown in Figure 1, then one meta-path instance $I^M_1$ in $G$ can be $U_1M_1A_2M_3$.

Definition 2.4. (Meta-path Neighbors) Given an HIN $G$ and a meta-path $M$, the meta-path neighbors of node $n$ are a set of nodes $N^M_n$ in $G$ which is connected to $n$ via the meta-path $M$.

Example: As shown in Figure 1, suppose meta-path $M = MAM$, then node $M_j$’s meta-path neighbors $N^M_{M_j} = \{M_2, M_3\}$.

3 RMS FOR META-PATH SELECTION
In this section, we first discuss the overview workflow of RMS, then we elaborate on the core components of our RMS model in detail. In the end, we illustrate how we train the model and describe our proposed strategy, which is used to speed up the training process.

3.1 Overview
We propose an RL-based framework RMS to select meta-paths for the MPRs. The overview of RMS is shown in Figure 2. The RL agent is a policy network that aims to investigate high-quality meta-paths. It takes the encoding of the current meta-path set as input and predicts the next relation to extend the current meta-path set. The reward will be the performance gain of the downstream recommender after using the new meta-path set and used to update the agent. Note that we also propose an early stopping approximation technique that makes the training process much more efficient.

3.2 Main Components of RMS
Reinforcement Learning (RL) follows a Markov Decision Process (MDP) [34] formulation, which contains a set of states $S$, a set of actions $A$, a decision policy $P$, and a reward function $R$. To maximize the cumulative rewards, the agent learns to consider actions based on the current state in the environment derived from an HIN. We design the key components of RMS and formalize it with the tuple $(S, A, P, R)$. The procedure is formulated as below:

State ($S$): It is important to store the structural information of current meta-paths to guarantee an agent can take an effective action based on the current state. We assign an ID to each relation type on an HIN (from 1 to $n$, where $n$ is the number of relation types). We encode each meta-path with a vector of length $n$, and each entity is the number of the corresponding relation type in this meta-path. It can make sure all the encodings have the same length. For instance, if there are 6 types of relations on an HIN$^1$, we will assign the relation type IDs to 1...6. If the ID representation of a meta-path $\phi$ is $[2, 6, 6, 4]$, then its encoding $E_\phi$ will be $(0, 1, 0, 1, 0, 2)$.

The state $s_i$ at step $i$ is represented by the embedding of the meta-path set $\Phi_i$ at step $i$. To embed a meta-path set, we add up all of the encodings of meta-paths in the meta-path set and apply an L2 normalization as below:

$$s_i = \text{Normalize} \left( \sum_{\phi \in \Phi_i} E_\phi \right),$$

$^1$Note that relations with opposite head and tail nodes should be treated as two different relations, e.g., $U \rightarrow M$ and $M \rightarrow U$ are two different relation types.
where $\Phi_i$ denotes the meta-path set at step $i$, $E_\phi$ represents the encoding of meta-path $\phi$. This encoding method can reflect the weights of each relation in a meta-path set effectively.

Note that different algorithms may require different meta-path forms. Some models may need the meta-paths starting and ending with a user or item node type. Then, we use $\{User-Item-User\}$ or $\{Item-User-Item\}$ as the initial meta-path set. If the algorithms require the meta-paths that strictly start with a user type node and end with an item type node, then we set $\{User-Item\}$ as the initial meta-path set. The initial meta-path set contains only one meta-path that directly connects users/items and reflects the user-item interactions since it is crucial for recommendation.

**Action ($A$):** The action space $A_a$ for a state $s_i$ is all the relations $(r_1, r_2, \ldots)$ that appear on an HIN, with a special action $STOP(r_0)$. At each step, the policy network will predict an action to extend the current meta-path set to yield a higher reward. Here, if the policy network selects the action $STOP$ or the current step exceeds the maximal step limit $I$, the meta-paths will not be extended.

If the predicted action is a relation on the HIN, we concatenate it with a complementary relation to make it a symmetric meta-path. This is done to ensure that the extended meta-paths will also start and end with a user/item. Then we try to extend all the current meta-paths with this symmetric meta-path and also insert this symmetric meta-path into the current meta-path set. Note that the previous meta-paths will also be copied and preserved in the meta-path set and will not be removed to make sure important information will not be overlooked.

For instance, if we want a meta-path set that starts and ends with a user type node, in a movie HIN (see Figure 1), the current set is $\{U-M-U\}$ and the predicted action (relation) is $M-A$. Then we will auto-complete it with the relation $A-M$ and it will become $M-A-M$. Then we traverse each meta-path in the current meta-path set to find the first positions where we can insert this relation into each meta-path. The meta-path set at the next step will be $\{U-M-U, U-M-A-M-U\}$. However, $M-A-M$ does not start with a user type, so it will not be added to the meta-path set. This strategy ensures the generated meta-paths meet the requirements of the recommender.

**Policy ($P$):** The decision policy $P$, is a state transition function that searches for the effective action based on the current state. The goal of $P$ is to train a policy that tries to maximize the discounted cumulative reward $R = \sum_{i=0}^I y^{I-i} r_i$, where $I$ is the maximal step limit and $y$ is a real number between 0 and 1. $y$ is acted as a discount factor to make the far-future reward less important than a near-future reward.

**Reward ($R$):** Reward ($R$) evaluates the decision made by the policy network to help the agent for better performance. Here, if the agent chooses $STOP$, the reward will be 0, if the agent chooses any relation which will not change the meta-path set, to punish this situation, the reward will be -1. Otherwise, we define the reward as the improvement of the performance. The formulation of reward can be represented as follows:

$$R(s_i, a_i) = N(s_i, a_i) - N(s_{i-1}, a_{i-1}),$$

where $N(s_i, a_i)$ is the performance of recommendation task at step $i$. Here, we adopt the NDCG@10 as the performance metric, but other metrics could also be used. After the policy network predicts the relation, we extend the current meta-path set and put them as the input of the downstream recommender. Then we train the recommender slightly using our proposed early stopping approximation technique (details in Section 3.3) and evaluate the model on the validation set to get the performance, then the reward is calculated to update the policy network.

### 3.3 Model Training and Optimization

**Early Stopping Approximation.** For each meta-path set, if we train the recommender to convergence and then test the model to calculate rewards, the whole training time of the RL agent will be extremely long. Fortunately, inspired by the hyperparameter optimization techniques [3, 23, 31], we find the performance of a model after training several epochs is highly correlated to the performance when the model converges, which means we can compare the quality of different meta-path sets on the lightly trained models. When we train the RL agent, we only lightly train the recommender and then evaluate the quality of generated meta-path sets. This strategy reduces the training time by more than 100 times. However, we admit the correlation between lightly trained model performance and final model performance is not impeccable, it does have an accuracy penalty and the selected meta-paths are not necessarily optimal, but our experiments show that it is still very effective and can find high-quality meta-paths.

**Training and Inference.** The whole procedure of training can be reviewed in Figure 2. The main steps are: i) get the current state based on the current meta-path set; ii) generate the next action (relation) based on the current state; iii) extend the current meta-path set and get the next state; iv) put the new meta-path set into the recommender and get the reward to update the RL agent. During the inference process, we start with the initial meta-path set and let the fully-trained RL agent generate relations and extend this set until the RL agent outputs the $STOP$ action or the current step exceeds the maximal step limit $I$.

**Optimization.** In our problem, States ($S$) are in a continuous space, and Actions ($A$) are in a discrete space. Therefore, we follow the steps of Deep Q-Network [26] (DQN), a typical RL algorithm, although it can be substituted with other RL algorithms. Suppose that we have a policy network $Q$ which is used to evaluate an action on a state: $S \times A \rightarrow \mathbb{R}$. We can directly construct a policy $\pi$ that maximizes the reward:

$$\pi(s) = \arg\max_a Q(s, a).$$

The Bellman equation [26] is used to update $Q$ as follows,

$$Q(s_i, a_i) = R(s_i, a_i) + yQ(s_{i+1}, \pi(s_{i+1})),
$$

where $s_{i+1}$ is the next state and $a_{i+1}$ is the next action. In our algorithm, we utilize multi-layer perceptron (MLP) as the policy network $Q$ since it can approximate any function.

While training the agent, we first calculate the temporal difference error $\delta$ as below:

$$\delta = Q(s, a) - (r + \gamma \max_a Q(s', a)).$$
In order to minimize the error $\delta$, we use Huber loss [18] function as shown below:

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,r,x) \in B} \mathcal{L}(\delta),$$

where $\mathcal{L}(\delta) = \begin{cases} \frac{1}{2} \delta^2 & |\delta| \leq 1 \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases}$ and $B$ is randomly sampled history data.

4 OUR RECOMMENDER

Figure 3 shows the architecture of our recommender HRec, containing two attention layers that learn the importance of different node neighbors and the different meta-paths, respectively. Here, we require a set of user meta-paths and a set of item meta-paths as input. User (Item) meta-paths means the meta-paths that start and end with a user (item) type node.

First, we will generate a meta-path reachable subgraph for each meta-path. The user (item) meta-path subgraphs will contain only user (item) type nodes, and the nodes are connected if they are meta-path neighbors. Then we perform a GAT [36] layer to each subgraph to get the inter-meta-path information. The GAT [36] layer can learn to differentiate the importance of different meta-path neighbors and aggregate the weighted information from them. Then the aggregated embeddings will be put into the next attention layer.

Since different meta-paths may play a different role in a recommendation task, we also adopt an attention mechanism to learn to assign the weight for different meta-paths and fuse the embeddings from each meta-path to get better node representations. Suppose that we have $X$ meta-paths $\{\phi_1, \ldots, \phi_X\}$. After GAT layers, we get $X$ groups of node embeddings denoted as $\{H_1, \ldots, H_X\}$. To get the importance of each meta-path, we transform the embeddings via a multi-layer perceptron (MLP) and multiply with a query vector $q^T$. Then get the average of all the meta-path-specific node embeddings.

The equation is shown as follows:

$$w^{\phi_x} = \frac{1}{|V_x|} \sum_{i \in V_x} \tilde{q}_i^{T} \cdot \sigma(\text{MLP}(h_i^{\phi_x})), \quad h_i^{\phi_x} \in H_x.$$  

Here, $\tilde{q}^{\phi_x}$ denotes the normalized importance of meta-path $\phi_x$. $\sigma$ is the activation function and $V_x$ is the node set of meta-path $\phi_x$.

Lastly, we aggregate the embeddings from each meta-path to get the final node embeddings $H$ according to their weights by:

$$H = \sum_{x=1}^{X} \tilde{q}^{\phi_x} \cdot H_x.$$  

Recommendation. After we get the neighbor-aggregated embeddings for users and items, we calculate the inner product of them to predict the scores that how much an item matches a user:

$$\hat{y}(u, i) = h_u^T h_i.$$  

To train our recommendation model, we adopt BPR loss [28], which makes the scores of observed user-item interactions larger than the scores of unobserved interactions:

$$\mathcal{L}_{\text{rec}} = \sum_{(u,i,j) \in O} -\ln \sigma(\hat{y}(u, i) - \hat{y}(u, j)),$$

where $O = \{(u,i,j) \mid (u,i) \in R^+, (u,j) \in R^-\}$ denotes the training set. $R^+$ means the positive (interacted) user-item pairs and $R^-$ denotes the negative (un-interacted) user-item pairs, $\sigma(\cdot)$ represents the sigmoid function. For the training steps, we adopt the embeddings learned by Matrix Factorization (MF) [28] as the initial embeddings of user and item nodes.

Training strategies. As mentioned above, HRec leverages a graph neural network to learn the representations of users and items. Nevertheless, graphs in recommender systems are often large-scale, incurring huge memory consumption and computation burden during message passing for graph neural networks, which makes it impossible to be applied to recommender systems directly when the graph is huge. Therefore, we design two strategies to improve the efficiency and effectiveness of the model.

1. In the real world, a graph is often scale-free, and some nodes on an HIN may have too many meta-path neighbors, resulting in a huge time and memory cost. Therefore, we employ the Neighbor Sampling (NS) technique, which will randomly sample a subset of meta-path neighbors to perform message passing each time. This technique allows us to avoid running out of memory and save a lot of time.
We conduct experiments to answer these research questions.

- **RQ1**: How does RMS perform compared to other meta-path selection strategies?
- **RQ2**: How does the RMS-explored meta-path set perform compared to other meta-path sets?
- **RQ3**: How does RMS-HRec perform compared to other state-of-the-art recommendation models?

### 5.1 Experiment Settings

#### 5.1.1 Datasets

We conduct our experiments on two commonly used public datasets Yelp\(^5\) and Douban Movie\(^6\), and a private in-

|       | Yelp   | Douban Movie | TCL   |
|-------|--------|--------------|-------|
| #Nodes| 31,092 | 37,595       | 122,284|
| #Edges| 976,276| 3,429,882    | 1,084,154|
| #Relations| 10     | 12           | 16    |

#### Table 2: Statistics of the datasets

(2) Meta-path is an effective tool to select semantic-related nodes. However, if a node is connected to too many nodes in the graph via a meta-path, then this meta-path should be ineffective because it cannot distinguish which nodes are highly related to a particular node. Therefore, for each meta-path, we will limit the density of the generated subgraph. We will discard the meta-path whose generated subgraph’s density is larger than a given threshold \(t (0 < t < 1)\). This strategy further helps us filter out low-quality meta-paths.

**Integrated with RMS.** According to the discussion above, HRec needs a set of user meta-paths and a set of item meta-paths as input. To cooperate with RMS, we train two RL agents to get the user meta-path set and item meta-path set separately. When we train the agent for the user meta-path set, we fix the item meta-path set as the initial set and vice versa. The RL agent will be updated by the feedback from the recommender.

### 5.1.2 Experiments Methods and Metrics

To evaluate the performance of recommendation, we adopt the leave-one-out evaluation\[^9, 10\]. For each user, we will regard the interacted items as positive items and the remaining as negative items. We randomly select 499 negative samples for each positive item and rank the positive one among the total 500 items. Further, we adopt two common metrics\[^10, 41\]: Hit Ratio at Rank \(k \) (HR@\(k\)) and Normalized Discounted Cumulative Gain at Rank \(k \) (NDCG@\(k\)) to evaluate our proposed models.

#### 5.1.3 Baseline Meta-path Selection Strategies

Since there exists no method that can derive different meta-paths for different MPRs, to demonstrate the effectiveness of RMS integration on MPRs, we design two baseline meta-path selection strategies based on random and greedy selection. We also select a meta-path discovery method in graph mining area.

- **Random**: We iteratively select a random meta-path set and perform training on the recommender using the selected meta-path set as input. We stop searching when the time limit is reached and select the meta-path set that achieves the best validation performance.

- **Greedy**: We initialize a meta-path set and add it to the meta-path set. Then train the recommender with the current meta-path set. Until the time limit is reached, we consider the final meta-path set as the current optimal meta-path set.

- **MPDRL\[^37\]**: This is a meta-path discovery method based on the user-given node pairs in graph mining area which is proposed to gain insights from data.

We run all of the meta-path selection strategies within the same time limit to ensure fairness in the meta-path selection process. Note that we adopt the early stopping approximation strategy on the recommender when we train the RL agent, so the RL agent can explore sufficient possible meta-path sets. After the RL agent is well trained on the training set, we use the found meta-path set and train the recommender until the model converges. Finally, we test the recommender on the test dataset.

#### 5.1.4 Baseline Recommenders

**Meta-path-based Recommenders.** To demonstrate the easy adaptation of RMS to MPRs, we first apply RMS to our recommender HRec and further apply it to two other representative state-of-the-art MPRs including HERec\[^30\] and MCRec\[^13\]. They need different forms of meta-paths as input.

- **HERec\[^30\]**: HERec needs two kinds of meta-path sets. One is the meta-paths start and end with a user type node and the other is the meta-paths start and end with an item type node.

- **MCRec\[^13\]**: MCRec needs only one kind of meta-path set. The meta-paths should start with a user type node and end with an item type node.

**Non-meta-path-based Recommenders.** In addition to the aforementioned baseline models, we present recommendation performance achieved by other state-of-the-art models that are not based on the meta-paths, e.g., Matrix Factorization (MF)-based models (BPR\[^28\], NCF\[^12\]), regularization-based models (CKE\[^48\], CFKG\[^49\]) and GNN-based models (GEMS\[^10\], KGAT\[^41\]).

#### 5.1.5 Implementation details

We implement all the models using Pytorch and DGL\[^5\]. For the recommendation part of RMS-HRec, we fix the embedding size to 64, the hidden layer size to 32, batch size to 90000. For the DQN of RMS, we adopt the implementation in\[^47\] and conduct a 3-layer MLP with \((32, 64, 32)\) hidden units for Q function. The learning rate and memory buffer size are set to 0.001 and 10000, respectively. We also set the maximal step limit to 4 for RMS-HRec, 3 for HERec and 5 for MCRec. We run the baseline recommenders using the default settings in their released code and original reports.
5.2 Meta-path Selection Method Study (RQ1)

Effectiveness of RMS. We compare RMS with self-design baselines over three MPRs. Table 3 shows the experimental results, and some observations are given as follows:

- On all the datasets and models, RMS constantly outperforms random and greedy strategies on all metrics. This demonstrates that our method can find more effective meta-paths.
- For MCRec, the performance improvement is not as much as in HRec. We argue that the performance gain highly depends on the recommendation algorithms. Some models do not leverage meta-paths effectively so that their performance is not highly sensitive to the selected meta-paths.

5.2 Meta-path Selection Method Study (RQ1)

Effectiveness of RMS. We compare RMS with self-design baselines over three MPRs. Table 3 shows the experimental results, and some observations are given as follows:

- On all the datasets and models, RMS constantly outperforms random and greedy strategies on all metrics. This demonstrates that our method can find more effective meta-paths.
- For MCRec, the performance improvement is not as much as in HRec. We argue that the performance gain highly depends on the recommendation algorithms. Some models do not leverage meta-paths effectively so that their performance is not highly sensitive to the selected meta-paths.

Table 3: Effect of different meta-path selection strategies

| Algorithm | Strategy      | Yelp HR@3 | Yelp NDCG@10 | Douban HR@3 | Douban NDCG@10 | TCL HR@3 | TCL NDCG@10 |
|-----------|---------------|------------|---------------|-------------|----------------|----------|--------------|
| HRec      | RMS           | **0.1484** | **0.1740**    | **0.2131**  | **0.2400**     | **0.3079** | **0.3230**   |
|           | Greedy        | 0.1381     | 0.1633        | 0.1800      | 0.2076         | 0.2895   | 0.3058       |
|           | Random        | 0.1181     | 0.1449        | 0.1946      | 0.2221         | 0.2875   | 0.3035       |
|           | MPDRL         | 0.1344     | 0.1589        | 0.1757      | 0.2041         | 0.2815   | 0.2993       |
| HERec     | RMS           | **0.0979** | **0.1215**    | **0.1613**  | **0.1984**     | **0.2891** | **0.3082**   |
|           | Greedy        | 0.0918     | 0.1166        | 0.1577      | 0.1960         | 0.2738   | 0.2920       |
|           | Random        | 0.0922     | 0.1167        | 0.1575      | 0.1945         | 0.2709   | 0.2878       |
|           | MPDRL         | 0.0915     | 0.1160        | 0.1570      | 0.1928         | 0.2713   | 0.2855       |
| MCRec     | RMS           | **0.1317** | **0.1540**    | **0.1961**  | **0.2236**     | **0.1734** | **0.2075**   |
|           | Greedy        | 0.1234     | 0.1504        | 0.1952      | 0.2204         | 0.1733   | 0.2070       |
|           | Random        | 0.1229     | 0.1451        | 0.1928      | 0.2193         | 0.1732   | 0.2066       |
|           | MPDRL         | 0.1223     | 0.1430        | 0.1919      | 0.2178         | 0.1694   | 0.2045       |

Table 4: Effect of meta-path on RMS-HRec

| Meta-path set | Yelp HR@3 | Yelp NDCG@10 | Douban HR@3 | Douban NDCG@10 | TCL HR@3 | TCL NDCG@10 |
|---------------|-----------|---------------|-------------|----------------|----------|--------------|
| RMS           | **0.1484** | **0.1740**    | **0.2131**  | **0.2400**     | **0.3079** | **0.3230**   |
| - UBUU        | 0.1441    | 0.1696        |             |                |          |              |
| - BCaB        | 0.1425    | 0.167         |             |                |          |              |
| - BCiB        | 0.1399    | 0.1657        |             |                |          |              |
| - BUB         | 0.1352    | 0.1597        |             |                |          |              |
| + BUBCaB      | 0.1375    | 0.1636        |             |                |          |              |
| + UBCiBU      | 0.1340    | 0.1602        |             |                |          |              |
| RMS           | **0.2131** | **0.2400**    |             |                |          |              |
| - MAM         | 0.2127    | 0.2381        |             |                |          |              |
| - MDMAM       | 0.2104    | 0.2375        |             |                |          |              |
| - MAMAM       | 0.2103    | 0.2384        |             |                |          |              |
| - MUM         | 0.2091    | 0.2360        |             |                |          |              |
| - MUMDM       | 0.2083    | 0.2349        |             |                |          |              |
| - MDM         | 0.2076    | 0.2357        |             |                |          |              |
| - UMDM        | 0.2075    | 0.2345        |             |                |          |              |
| - UMDMU       | 0.2027    | 0.2308        |             |                |          |              |
| + UMAMU       | 0.2096    | 0.2358        |             |                |          |              |
| + UMUU        | 0.2064    | 0.2333        |             |                |          |              |
| RMS           | **0.3079** | **0.3230**    |             |                |          |              |
| - UM          | 0.3057    | 0.3211        |             |                |          |              |
| - MDM         | 0.3055    | 0.3213        |             |                |          |              |
| - MLM         | 0.3018    | 0.3185        |             |                |          |              |
| - MLMDM       | 0.3012    | 0.3169        |             |                |          |              |
| - UMDM        | 0.2841    | 0.3005        |             |                |          |              |
| - MUM         | 0.2800    | 0.2965        |             |                |          |              |
| - MUMDM       | 0.2713    | 0.2884        |             |                |          |              |
| + MTM         | 0.3060    | 0.3212        |             |                |          |              |
| + MDMMD       | 0.3040    | 0.3200        |             |                |          |              |

Figure 5: NDCG@10 of each meta-path selection strategy over different time limits on HRec model

We also test the performance of each meta-path selection strategy over different time limit. Due to the space limitation, we only show
we design an experiment on Yelp dataset. We manually designed four meta-path sets: (1) Set1: \{UU, BCiB\} (2) Set2: \{UU, UBU, BCiB\} (3) Set3: \{UBU, BUB\} (4) Set4: \{UBU, BUB, BCiB, BCaB\}.

We test each meta-path set on HRec on the validation set. As shown in Figure 7, we found that even though we slightly train the model (for one or a few epochs), the effectiveness of Set4 is the best, which is similar when the model converges. Though, we admit that this strategy may not guarantee the selected meta-paths are optimal, it greatly reduces the training time by more than 100 times and can also get decent results.

5.3 Meta-path Sensitivity Analysis (RQ2)

To illustrate the influence of meta-paths on recommendation performance and to demonstrate the effectiveness of RMS, we run RMS-HRec on all datasets.

Adding/removing one meta-path. To test the meta-path sensitivity of RMS-HRec, we adopt a leave-one-out principle to check whether the performance changes if we remove/add one meta-path from/to the RMS-found meta-path set. Table 4 shows the performance of RMS-HRec, where '-' means remove a meta-path, '+' means add a meta-path. The findings on sensitivity analysis are listed below:

- When we add/remove any meta-path to/from the meta-path set found by RMS, the performance will decrease, which proves that our approach can figure out the high-quality meta-paths.
- Performance deteriorates differently when we remove different meta-paths, demonstrating that different meta-paths have different impacts on performance. When we add another meta-path, the performance also drops, indicating that using as many meta-paths as possible does not necessarily lead to performance gains.

Adding/removing more meta-paths. To explore the quality of other meta-path set. We also try to add/remove more meta-paths and test the performance. We just show some representative results in Table 5 due to the space limit. It is the performance of three handcrafted meta-path sets compared with the RMS-found set. These results show if we remove/add more meta-paths, the performance may be even worse. Compared with the manually created meta-path sets, RMS can also consistently improve the recommendation quality. Even compared with the meta-path set used in other paper (i.e., Yelp dataset S2 [30]), we can also get around 4.5% performance gain in terms of HR@3.

5.4 Recommendation Performance (RQ3)

Recommendation Effectiveness. Table 6 shows the overall performance of the algorithms in all datasets. For HERec and MCRRec algorithm, instead of manually choosing a meta-path set, we use the best meta-path set found by RMS. The major observations are summarized as follows:

- Our method outperforms all the baselines for all metrics on all datasets. This shows that the choice of meta-paths plays an important role in recommendation on HINs and our method can effectively leverage the semantic and structural information of the HINs and perform recommendation.
- RMS-HRec outperforms two MF-based methods (BPR, NCF). This is because HINs have more entities besides users and items and contain more semantic information. RMS-HRec also has a better performance compared to two GNN-based methods (GEMS, KGAT) and two meta-path-based methods.

For all datasets, S1 is the initial meta-path set that contain only two simplest meta-paths. S2 is a medium-size set and S3 is a large set. Here, Yelp S2 is also the handcrafting meta-path set used in [30]. For the other sets, we just randomly add or delete meta-paths from RMS-found set. We omit the specific meta-paths due to the space limit.
Table 6: Overall Performance comparison of RMS-HRec

|               | Yelp       | Douban Movie | TCL        |
|---------------|------------|--------------|------------|
|               | HR@1 | HR@3 | NDCG@10 | HR@1 | HR@3 | NDCG@10 | HR@1 | HR@3 | NDCG@10 |
| RMS-HRec      | 0.0648 | 0.1484 | 0.1740 | 0.0997 | 0.2131 | 0.2400 | 0.1654 | 0.3079 | 0.3230 |
| BPR           | 0.0388 | 0.1025 | 0.1301 | 0.0529 | 0.1421 | 0.1768 | 0.1211 | 0.2704 | 0.2886 |
| NCF           | 0.0514 | 0.1251 | 0.1522 | 0.0622 | 0.1605 | 0.1974 | 0.1279 | 0.2513 | 0.2673 |
| CFKG          | 0.0572 | 0.1246 | 0.1477 | 0.0637 | 0.1650 | 0.2033 | 0.1602 | 0.2984 | 0.3145 |
| CKE           | 0.0454 | 0.1092 | 0.1360 | 0.0612 | 0.1581 | 0.1947 | 0.1390 | 0.2818 | 0.3023 |
| HERec         | 0.0548 | 0.1317 | 0.1540 | 0.0928 | 0.1961 | 0.2256 | 0.1484 | 0.2860 | 0.3055 |
| MCRec         | 0.0389 | 0.0979 | 0.1215 | 0.0594 | 0.1613 | 0.1984 | 0.0602 | 0.1708 | 0.2056 |
| GEMS          | 0.0100 | 0.0294 | 0.0408 | 0.0262 | 0.0591 | 0.0740 | 0.0344 | 0.0701 | 0.0923 |
| KGAT          | 0.0415 | 0.1151 | 0.1388 | 0.0644 | 0.1669 | 0.2042 | 0.1454 | 0.2990 | 0.3162 |

Figure 8: Training efficiency of each recommender

(HERec, MCRec), this shows using meta-paths and combining with GNNs is a better way to leverage HINs.

Model Efficiency. We report the training efficiency of RMS-HRec and all the baseline recommenders in Figure 8. Our method needs to search meta-paths first, which will take some time. However, we find the training efficiency of our method is similar as or faster than other GNN-based methods (GEMS, KGAT) and not much slower than other methods. Therefore, we think the trade-off between accuracy and efficiency is reasonable.

6 RELATED WORK

Meta-path-based recommenders. Recently, Meta-paths are wildly used in a large volume of HIN-based recommendation models [4, 42, 44]. The MPRs can be divided into two categories according to the requirement of the input meta-path form. The first category [5, 13] uses the meta-paths which start with the user type and end with the item type. This kind of algorithms can model the high-order connectivity and complex interactions between users and items. However, they rely on explicit path reachability and may not be reliable when path connections are sparse or noisy. The second category [9, 30] needs two kinds of meta-paths: 1) the meta-paths that start and end with a user type, and 2) the meta-paths that start and end with an item type. They can fully mine latent structure features of users and items and are more resistant to sparse and noisy data, but may include too much information that is not relevant to the recommendation. However, all of these methods suppose that meta-paths are already given, which is not realistic in real applications.

Meta-path discovery on HINs. Some works [25, 29, 37] attempt to discover meaningful meta-paths on HINs. GEMS [10] adopts a genetic algorithm to find effective meta-structures (i.e., abstract graphs consisting of node and edge types) for recommendation and performs GCN [21] layers to fuse information from the found meta-structures. However, these meta-structures can only be used in their recommender since MPRs cannot handle meta-structures. Also, it is very time-consuming because of the colossal search space. RL-HGNN [51] proposes a reinforcement learning (RL)-based method to find a meta-path for each node to learn its effective representations. However, they focus on producing powerful node embedding for each node using a found meta-path and enhancing the GNN model, rather than generating a set of format-specific meta-paths to support meta-path-based recommendation algorithms. Besides, it is not easy to adapt RL-HGNN to our scenario since all of their RL components are designed for obtaining better node embeddings, and our designs are completely different from theirs. In addition, it is not clear how their method can be adapted to generate meta-paths with specific forms. According to the above discussion, even though some works are designed to find meta-paths, none of them can be directly adapted to benefit existing meta-path-based recommenders.

7 CONCLUSION

In this paper, we propose a reinforcement learning-based meta-path selection framework RMS to automatically find an effective meta-path set for recommendation. This allows recommender systems to better leverage information conveyed in meta-paths when making recommendations. Our experiments show that choices of meta-paths significantly affect the performance of meta-path-based recommenders and our proposed framework can effectively find high-quality meta-paths for any meta-path-based recommender. In addition, we design a new meta-path-based recommender, along with special training strategies, which better explore the potential of meta-paths and can achieve state-of-the-art performance.

ACKNOWLEDGEMENTS

Reynold Cheng, Wentao Ning, and Nan Huo were supported by the HKU-TCL Joint Research Center for Artificial Intelligence (Project no. 200009430), the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18), the University of Hong Kong (Projects 104005858 and 10400599), and the Guangdong–Hong Kong–Macau Joint Laboratory Program 2020 (Project No: 2020B1212030009). Ben Kao was supported by HKU-TCL Joint Research Center for AI and Innovation and Technology Fund (Grant ITS/234/20).
