Efficient Interactive Recommendation via Huffman Tree-based Policy Learning

Longxiang Shi
Zhejiang University City College
Hangzhou, China
shilx@zucc.edu.cn

Zilin Zhang
Zhejiang University
Hangzhou, China
zilinzhang@zju.edu.cn

Shoujin Wang
University of Technology Sydney
Sydney, Australia
shoujin.wang@uts.edu.au

Binbin Zhou
Zhejiang University City College
Hangzhou, China
bbzhou@zucc.edu.cn

Minghui Wu*
Zhejiang University City College
Hangzhou, China
mhwu@zucc.edu.cn

Cheng Yang
Zhejiang University City College
Hangzhou, China
yangc@zucc.edu.cn

Shijian Li
Zhejiang University
Hangzhou, China
shijianli@zju.edu.cn

ABSTRACT

Interactive recommender systems (IRSs) are an essential part of our daily life, as they can suggest items to persistently satisfy our demands. Due to the interactive nature, conventional static recommendation methods such as matrix factorization, and content-based filtering are ineffective to capture the dynamic preferences of users. Recently, reinforcement learning (RL) has shown great potential in addressing the challenges in IRSs, since it can capture users' dynamic preferences and model the long-term profit of user-item interactions. However, millions of items in real-world IRSs lead to a large discrete action space in the RL setting, rendering RL-based IRSs inefficient and hindering their widespread application. Such an inefficiency issue has not been well addressed in the literature. In order to address this issue, we propose a novel Huffman Tree Policy Recommendation (HTPR) framework. Specifically, a novel policy learning network based on a newly designed Huffman tree is proposed for policy representation learning, which effectively improves the learning efficiency. Moreover, a novel parameter-sharing scheme is devised to further reduce unnecessary computations. Extensive experiments on two real-world benchmark datasets demonstrate the superiority of HTPR over the state-of-the-art IRS methods in terms of both recommendation accuracy and efficiency.

KEYWORDS

Reinforcement Learning; Deep Learning; Recommender System

ACM Reference Format:
Longxiang Shi, Zilin Zhang, Shoujin Wang, Binbin Zhou, Minghui Wu, Cheng Yang, and Shijian Li. 2023. Efficient Interactive Recommendation via Huffman Tree-based Policy Learning. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

*Corresponding author

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

1 INTRODUCTION

Interactive recommender systems (IRSs) aim to recommend preferred items to users by accurately modeling users' dynamic preferences towards items in an iterative manner. As an emerging recommendation paradigm, IRSs have been widely employed in our daily life. Due to the interactive nature of IRS, recommendation methods for IRS must be able to effectively model the iterative interactions between users and RSs so as to capture users' dynamic preferences and thus provide recommendations that can satisfy users' future needs [30]. Conventional non-interactive recommendation methods including both shallow (e.g., matrix factorization [8], content-based filtering [22]) and deep models (recurrent neural networks, convolutional neural networks, and graph neural networks [24]) can not effectively model such user-RS interactions, and thus they are not ready for building IRSs.

In recent years, reinforcement learning (RL), as a promising machine learning method, has been proven to be a powerful method for modeling the intensive interactions between different agents while considering the long-term reward of each interaction [19]. Naturally, RL has been widely applied to building IRSs so as to well model the interactions between the RS and users. [18, 29, 33]. More recently, deep learning has been incorporated into RL to form deep reinforcement learning (DRL). Particularly, due to its powerful representation capability, DRL has shown great potential to well model the intensive RS-user interactions in IRS scenarios [3]. Hence, DRL has been a mainstream method for building IRSs in recent years [26, 30, 32].

However, one key challenge for employing DRL in IRSs is the large action space problem. That is, real-world online IRSs always have to handle millions of items in an online shopping platform, leading to an extremely large action space. When facing such a large action space, conventional RL methods such as deep Q-Network (DQN) [11] and policy gradient methods [19, 20] are ineffective and inefficient in both policy learning and decision-making. Taking the DQN-based methods [29, 31] as an example, the computation involves a maximization operation over all the probable actions and

may become intractable when the size of action space is large [4]. A similar problem also exists in some policy gradient methods [13, 28]. To alleviate this issue, one popular strategy is to reduce the large action space into a considerable one. For instance, DDPG-KNN [4] has been developed to embed the discrete action space into a continuous space and then to use an approximate nearest-neighbor search method when making decisions. But this method is inefficient in the embedding process and suffers from the inconsistency problem [21]. More recently, TPGR [3] was proposed to organize the policies into a tree structure in order to improve efficiency in both policy learning and decision-making stages. However, the balanced tree structure in TPGR treats the frequent and rare items equally and thus is inefficient in learning the frequent items since they have much higher frequencies in the dataset. Due to such common long-tail issue in recommendations, the balanced tree structure is not efficient in learning and decision-making [14, 15].

To address the aforementioned significant gap in existing works, in this paper, we propose a novel Huffman-tree Policy Recommendation (HTPR) framework. To be specific, we devise a novel recommendation policy learning model based on a newly designed Huffman tree to greatly improve the learning efficiency. Huffman tree can greatly speed up the learning process by using very limited neuron nodes to represent those frequent items in the dataset during the policy learning. As a result, the computation cost of our proposed HTPR is largely reduced compared with existing policy learning methods. Subsequently, the overall learning and decision efficiency have been greatly improved. In addition, a novel parameter-sharing scheme is devised to further improve the efficiency of the learning process by reducing some unnecessary parameter load.

Our main contributions in this work are summarized below:

(1) To the best of our knowledge, this is the first work to build an IRS based on Huffman tree structure-based DRL. We design a novel Huffman-tree Policy Recommendation (HTPR) framework for building efficient IRSs.
(2) We devise a novel parameter-sharing scheme to further improve the learning efficiency by reducing unnecessary computation load.
(3) We conduct extensive experiments on two real-world benchmark datasets to demonstrate that our proposed HTPR can not only achieve superior recommendation accuracy but also is more efficient in learning and decision-making than state-of-the-art IRS methods.

2 RELATED WORKS

For most RL methods, the large discrete action space is an intractable challenge as the learning may become less effective and inefficient with the growth of the action space. To address this issue, one common approach to address this problem is simplifying the action space into a considerable one. For instance, For example, Sallans et al. [17] proposed a method that can factorize the action space as negative free energies and then adopt an ensemble method to learn the policy. Pazis et al. [16] proposed a method that embedded each action in binary format and optimized a value function associated with each bit. Bellemare et al. [1] demonstrated that pre-categorizing action spaces can improve the performance of RL method. Another approach is to decompose the large discrete action space into a smaller continuous one and then choose the corresponding actions based on similarity in the embedding space. Such methods can be found in DDPG-KNN [4] and Calca [28]. However, those methods are inefficient when embedding the actions. More recently, Chandak et al. [2] proposed a method that can learn the action representations during RL process, which can factorize the action space into a smaller one. Although this method is effective for solving tasks with large discrete action spaces, training such a model is inefficient as it involves integral calculating over the probability distribution of actions.

Designing specific neural network structures to facilitate learning the tasks in large discrete action space is another popular strategy. For example, DRN [31] and DEERS [29] adopted a refined DQN [11] network to learn the policies with large action spaces. However, learning for DQN-based solutions involves a maximum operation among the actions, whose time complexity grows linearly with the number of actions. Moreover, Zhao et al. [28] used a deconvolution layer that maps the action space into a matrix. Chen et al. [3] proposed TPGR that uses a balanced clustering algorithm to build a tree-structured policy to simplify the learning and decision process. However, TPGR adopted a balanced tree in policy representation and treated all the items equally despite the occurrence of the items, and is inefficient in learning the common items as they are frequently encountered. In practice, the overall efficiency of TPGR may be hindered by the common encountered items.

In summary, existing methods are mostly inefficient or unpractical in real-world IRS scenarios. In this paper, we utilize the Huffman tree in policy representation, which can achieve better efficiency and effectiveness over several state-of-the-art methods.

3 METHOD

In this section, we first demonstrate the problem definition to describe the recommendation process using the RL-based method and then go into details of the proposed Huffman-tree policy method.

3.1 Problem Definition

In this paper, we study the interactive recommendation task in which the recommendation agent delivers an item to users and receives feedback from users' decisions. During the interaction, the user (aka the environment in RL setting) sequentially selects items recommended by the recommender. The goal of the recommender agent is to maximize its cumulative reward. Under these circumstances, the interactive recommendation process can be modeled as a Markov Decision Process (MDP) with the key components as $< S, A, P, R, \gamma >$ which can be defined as follows:

- **State** S: A state $s \in S$ is defined as the recent interactions between a user and recommender system. The consecutive interaction sequence length depends on the reality and the pre-defined boundary conditions.
- **Action** A: An action $a \in A$ is an item that the recommender system suggests for the user.
- **Transition probability function** P: $P(s'|s, a)$ is the transition function that indicates the new state s', when the recommender agent suggests item a at the state of s.

- **Reward function** R: $R(s, a)$ is a function that figures out the immediate reward the recommender agent receives from the user's feedback after the agent suggests item a at the state of s.
- **Discount factor** γ: $\gamma \in [0, 1]$ defines the discount factor that measures the value of present reward and future reward.

Each episode in the MDP denotes a recommendation process and gets the discounted sum of rewards as the return. For an episode with length of T, the cumulative reward of the t-th step is: $G_t = \sum_{i=t}^{T}(\gamma^{i-t}R(s_t, a_t))$. The goal is to find a recommendation policy $\pi : S \rightarrow A$ that maximizes the cumulative reward for the whole interactive recommendation process:

$$\pi = \arg\max_{a \in A} \mathbb{E}[G_t] \tag{1}$$

3.2 Efficient Recommendation Policy Framework Based on Huffman Tree

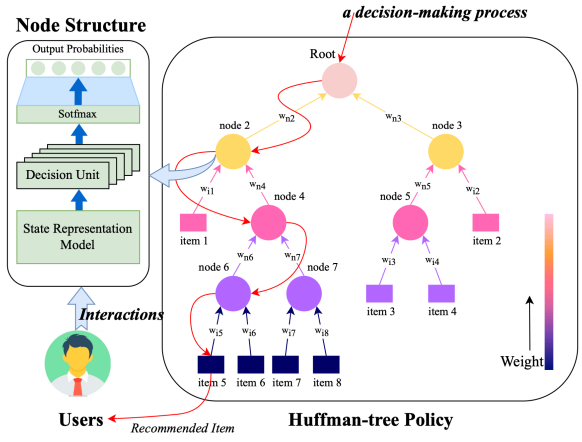

Figure 1: Huffman-tree policy network for IRS

3.2.1 Improving Efficiency in Forward Pass with Huffman Tree. As mentioned above, in interactive recommendation settings, one major concern for employing DRL is the large-scale action space, which can lead to an ineffective and inefficient performance in learning and decision-making. Generally, policy gradient methods are more popular in dealing with large discrete action spaces than value-based RL methods, as most value-based RL methods involve maximization over all the actions in both learning and utilization. For policy gradient methods, the forward pass of the policy networks requires the calculation of probabilities for the available items in action spaces. The output probabilities in the policy network are often implemented as Softmax layers. Since the forward pass of Softmax layer needs explicit normalization over the actions, organizing the output actions with a tree-structured policy network can substantially improve the efficiency of forward pass. Such works can be found in hierarchical Softmax [10] and TPGR [3]. However, existing works that adopted balanced tree policy networks for interactive recommendation need to compute the cumulative product by traversing the tree from root to leaf for a certain item when

Algorithm 1 Huffman Tree Policy Structure Build Algorithm

Input: Input item list $< i_1, i_2, ..., i_m >$ with the corresponding occurrence frequency list $< f_1, f_2, ..., f_m >$, number of the branch of Huffman tree c.

Initialize: A nodes set whose elements are the node with weight f_j the jth item: $Set = \{Node(f_j, i_j) | j \in [1, m]\}$.

while $length(Set) >= c$ **do**
  Sort Set by the node's weight to be a non-decreasing sequence.
  Create a node $node_{new}$ with weights $\sum_{j=1}^{c} f_j$.
  for j from 1 to c **do**
    Assign the jth child of $node_{new}$ to be the jth node of Set.
    Remove the jth node from Set.
  end for
  Add $node_{new}$ into Set.
end while
if Set contains more than one node **then**
  Create a node $node_{new}$ with the sum of weights in Set.
  for j from 1 to $length(Set)$ **do**
    Assign the jth child of $node_{new}$ to be the jth node of Set.
    Remove the jth node from Set.
  end for
end if
return The set that contains the root node of Huffman tree $Set = \{node_{new}\}$.

performing forward pass. Denoting the calculation time for each node in forward pass as t_{node}, as the layer for each leaf-node is equal to the tree depth d in the balanced tree network, the average calculation time (ACT) for each item is dt_{node}. Unfortunately, in real-world recommendation systems, the distribution of items is imbalanced as the common items involve most of the ratings and the unpopular items hold only a few ratings, so-called the long-tail recommendation effect [14]. Denoting the frequency of each item is $f_1, f_2, ..., f_m$, where m is the total number of items, the ACT for balanced tree policy network can be written as:

$$ACT = \frac{dt_{node}}{m} \sum_{i=1}^{m} f_i \tag{2}$$

From this equation, we can infer that the long-tail effect itself may bring a challenge in efficiency for a balanced tree policy network, as it treats all items equally and arranges each item in the bottom level of the tree. If we arrange the corresponding items of each leaf in different levels, then the ACT can be written as:

$$ACT = \frac{t_{node}}{m} \sum_{i=1}^{m} f_i d_i \tag{3}$$

Where d_i denotes the depth of the ith item.

Fortunately, the above equation can be formalized as a minimum weighted path problem, which can be solved by the well-known Huffman tree [7]. Huffman tree organizes the leaf note of each item to different layers according to their occurrence. In the Huffman tree, the common items are assigned to the low layers while the rare items are assigned to the high layers. Through the Huffman

tree architecture, the efficiency of the forward pass can be optimized for learning and utilizing a recommendation policy with policy gradients. Hence, we adopt the Huffman tree for policy network representation. The probabilities of frequently encountered items are represented with fewer neurons and vice versa. In this way, learning and decision efficiency can be improved. Our experiments show that these settings can improve performance as well as efficiency.

Specifically, we adopt n-ary Huffman tree to reduce the total number of non-leaf nodes, which are regarded as the decision unit in the Huffman tree policy. Experiments show that a larger number of non-leaf nodes leads to higher learning time (refer to the experiment part). The building algorithm of the n-ary Huffman tree is depicted in Algorithm 1. We can first estimate the occurrence of all available recommended items. Based on the occurrence the Huffman tree can be obtained by Algorithm 1. Then we can organize the tree nodes (i.e., neural networks) as the Huffman tree structure.

3.2.2 Improving Efficiency in Backpropagation with Parameter-Sharing. The overall framework of the proposed Huffman policy network is illustrated in Figure 1. In the Huffman-tree policy network, each non-leaf node of the Huffman tree policy receives the user historical item list as input and outputs the probabilities of the branch. Each non-leaf node is a decision network and contains two parts: the state representation model and a decision unit. The state representation model receives the user's historical item list as input and outputs the state representation vector that models the user's behavior, and the decision unit receives the state vector and outputs a probability of the corresponding branch. For each leaf node in the Huffman tree policy network, the recommendation probability of each item is computed by traversing the Huffman tree from the root to the leaf.

During the learning of the Huffman policy network, we obtain a cumulative product of the tree node network, which contains a series of parameters. The backpropagation operation in such complicated networks is time-consuming as it contains a large number of parameters with the increase of nodes. Hence, to further improve the learning efficiency, we adopt the parameter-sharing strategy, which is widely used in neural networks to reduce unnecessary computations. Specifically, we divide the decision node in the Huffman tree into two parts: the state representation model and the decision node. The state representation model receives the user historical item list as input and outputs the state representation vector, and the decision unit receives the state vector and outputs a probability of the corresponding branch. As the state representation model may be complicated neural networks (i.e, convolutional neural networks and recurrent neural networks), all the nodes in the Huffman tree policy network share the same parameter setting. In addition, to improve the ability of the Huffman tree policy network, the decision unit of each node holds an unshared parameter setting. Through this parameter-sharing strategy, we can obtain a balance of efficiency and performance.

Finally, based on the above settings, the Huffman tree policy network is shown in Figure 1.

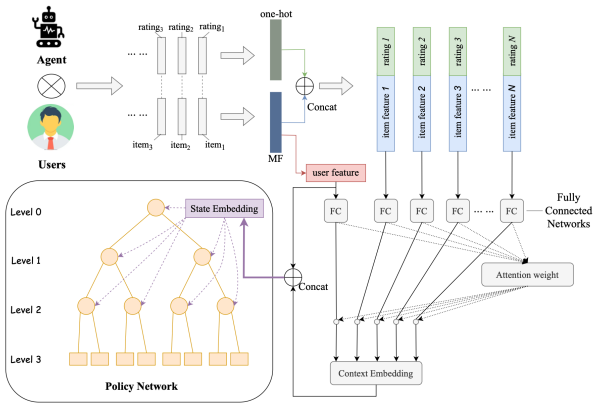

Figure 2: State representation model

3.3 State Representation Model

In this part, we present the state representation model, as shown in Figure 2. In this figure, we assume that the recommender agent is performing tth recommendation. The N previous user selected items along with their feedbacks (ratings) are used to encode the state. The user and item features are obtained through matrix factorization on the historical user-rating matrix. Each rating is mapped to a one-hot vector. We then adopt an attention-based state representation model inspired by ATEM [25] to model the state from user historical interactions. Generally, attention-based models are more efficient compared to RNN-based models in modeling the sequential data [23, 24]. Different from ATEM, we add user features in the input layer to distinguish different users. The input features, including user features and item features, are processed by a fully-connected layer to obtain the feature embeddings. Then we use the attention-based model to capture the contributions of each item feature along with the user features. Denoting h_i as the ith preprocessed item feature, h_{user} as the preprocessed user feature, the attentive context embedding is calculated as follows:

$$\begin{aligned} e &= \sum_{i=1}^{N} (\alpha_{ti} h_i) + \alpha_{user} h_{user}, \\ s.t. &\sum_{i=1}^{N} (\alpha_{ti}) + \alpha_{user} = 1 \end{aligned} \tag{4}$$

Where α_{ti} is the integration weight of the ith item embedding vector w.r.t the tth target item, α_{user} is the integration weight for a specific user.

The attention weights are calculated through a Softmax layer as below:

$$\begin{aligned} \alpha_{ti} &= \frac{\exp(f(h_i))}{\sum_{i=1}^{N} \exp(f(h_i^F)) + \exp(f(h_{user}))} \\ \alpha_{user} &= \frac{\exp(f(h_{user}))}{\sum_{i=1}^{N} \exp(f(h_i^F)) + \exp(f(h_{user}))} \\ f(h_i) &= ReLU(W^{\alpha} h_j^T + b^{\alpha}) \; \forall h_j \in \{h_i, h_{user}\} \end{aligned} \tag{5}$$

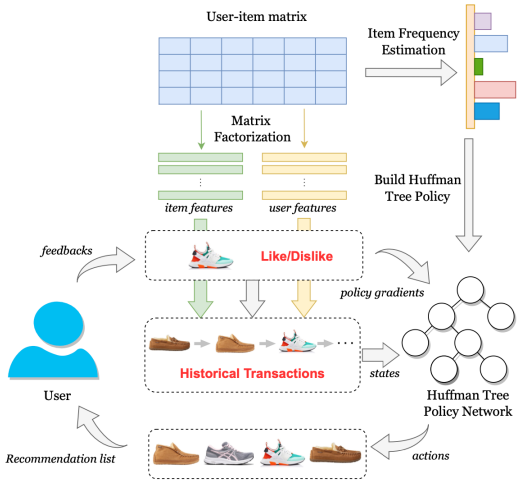

Figure 3: The learning process of Huffman tree policy network

where W^{α} and b^{α} are the weight and bias parameters for the attention layer respectively.

In the output layer of the state representation model, we concatenate the input user feature and the attentive context embedding as the state embedding vector.

3.4 Learning Process

The learning process of the Huffman tree policy network is depicted in Figure 3. We first use the historical user-item matrix to extract the features for each item and user. Specifically, we adopt Funk SVD [5] to decompose the user-item matrix into an item matrix and a user matrix. The item matrix and user matrix are used as item features and user features, respectively. Additionally, the item occurrence frequency can also be estimated. Based on the item distribution the Huffman tree policy network can be built using Algorithm 1, with each leaf node corresponding to a certain recommended item. Recommendations are provided to the user by mapping the output action of the policy to the items. During the interaction between the user and the recommender agent, the Huffman tree policy can be trained via the policy gradient method.

The policy network can be trained by any policy gradient method. In this work, we adopt REINCFORCE [27] algorithm as the learning method of our n-ary Huffman-tree model. Denoting the whole policy network as π_θ, the purpose of our algorithm is to maximize the expectation of discounted cumulative rewards. The loss function is defined as:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r_t] \tag{6}$$

According to the policy gradient theorem, each step to update the parameters θ is based on the gradient of the loss function which can be written as:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)] \tag{7}$$

Algorithm 2 Learning the Huffman tree policy network

Input: learning rate $\eta > 0$, discount factor γ, episode length n.
Initialize: Policy parameter θ.

repeat
  Sample an episode $\{(s_1, a_1, r_1), ..., (s_T, a_T, r_T)\}$ from historical user data or online data.
  for $k = 1$ to n **do**
    $Q^{\pi_\theta}(s_k, s_k) = \sum_{i=k}^{n} \gamma^{i-k} r_i$
    Calculate $\pi_\theta(a_t|s_t)$ based on the Huffman tree policy network.
    $\theta \leftarrow \theta + \eta Q^{\pi_\theta}(s_k, s_k)\nabla_\theta \log \pi_\theta(a|s)$
  end for
until Coverage
Return: π_θ

Table 1: Summary Statistics of Datasets.

Dataset	Instant Video	Baby
#Users	122,609	175,826
#Items	8,229	8,256
#Ratings	145,983	228,861
#Users in training set	98,084	140,660
#Users in testing set	24,522	35,166

where $\pi(a|s)$is an action policy giving the probability of taking action a at state s, and $Q^{\pi_\theta(s,a)}$ is the action-value function denoting the expected cumulative reward of action a at state s. During learning steps, $Q^{\pi_\theta(s,a)}$ can be estimated by sampling trajectories from historical user data under policy π_θ. Detailed information of the learning algorithm is shown in Algorithm 2.

4 EXPERIMENT

In this part, we evaluate the proposed HTPR on 2 benchmark datasets. We first describe the setup of experiments and then evaluate the performance and efficiency of the proposed method. We intend to answer the following research questions (RQ) through experiments:

- **RQ1:** How does HTPR perform compared with the state-of-the-art IRSs in terms of recommendation accuracy?
- **RQ2:** Does HTPR improve the efficiency in learning and decision for IRSs?
- **RQ3:** How do the different hyper-parameters affect the performance of HTPR?

4.1 Experimental Setting

4.1.1 Datasets. We evaluate the proposed HTPR on 2 real-world benchmark datasets: Instant Video and Baby, which are commonly used for evaluating the performance of IRS. The two datasets contain product reviews collected from Amazon [6, 9]. The users' ratings of those datasets ranged from 0 to 5. We use a quarter of each dataset for evaluation. Specifically, we use 80% of the data for training and the left 20% for testing. Five-fold cross-validation is adopted in the evaluation. The statistics of the two datasets are listed in Table 1. Due to the interactive nature of IRSs, an ideal

Table 2: Recommendation accuracy comparison with baselines, *the improvement is significant at $p < 0.05$.

Dataset	Metric	Popularity	SVD	DDPG-KNN(k=1)	DDPG-KNN(k=0.1M)	DQN-R	TPGR	HTPR	Improv.
Instant Video	HR@10	0.00004	0.00119	0.00336	0.01807	0.05376	0.10801	**0.16573***	34.6%
	MRR@10	0.00001	0.00041	0.00165	0.00679	0.02117	0.03970	**0.10232***	61.2%
	HR@30	0.00016	0.00507	0.00713	0.05137	0.12150	0.18900	**0.25362**	25.3%
	MRR@30	0.00002	0.00075	0.00190	0.00884	0.02497	0.04425	**0.10807***	59.1%
Baby	HR@10	0	0.00054	0.00019	0.00216	0.05405	0.06201	**0.06790***	8.67%
	MRR@10	0	0.00086	0.00003	0.00057	0.01791	0.02841	**0.03047***	6.76%
	HR@30	0.00003	0.01871	0.00019	0.00381	0.11064	0.12194	**0.13050***	6.57%
	MRR@30	0	0.00220	0.00003	0.00066	0.02142	0.03178	**0.03453***	7.96%

Table 3: Efficiency comparison for decision making (in seconds), *the improvement is significant at $p < 0.05$

Method	Average learning time per step		Average time per- decision	
	Instant Video	Baby	Instant Video	Baby
DDPG-KNN(k=1)	0.56430	1.42474	0.09974	0.01657
DDPG-KNN(k=0.1M)	0.88891	1.72415	1.20832	1.72455
DQN-R	0.64808	1.75619	0.10880	0.08240
TPGR	0.42570	0.60594	0.05370	0.06397
HTPR	**0.28766***	**0.33975***	**0.03158***	**0.03448***
Improv.	32.4%	43.9%	41.2%	46.1%

way for conducting experiments is to directly interact with users. Unfortunately, online experiments are too expensive and vulnerable to commercial risks for IRS [3, 32]. Similar to some existing works, we use an offline environment simulator that is based on offline datasets to conduct experiments. For each timestep, the environment simulator provides the historical items and ratings of a user and releases a reward after the recommender agent suggests items. The reward function of the environment simulator is set to the normalized ratings, which linearly normalize the ratings into $[-1, 1]$:

$$R(s, a) = -1 + 2 * \frac{r_{ij}}{5} \tag{8}$$

where r_{ij} is the rating for user i for item j.

4.1.2 Evaluation Metrics. We report 2 commonly used evaluation metrics in the experiments:

- **Hit Ratio (HR)@K:**HR measures the fraction of items that user favors in the recommendation list and is calculated as below:

$$HR@K = \frac{1}{\#\ Users * K} \sum_{\#\ Users} \sum_{i=1}^{K} \theta_{hit} \tag{9}$$

  Here we define $\theta_{hit} = 1$ if the item user selects and favors ($r_{ij} > 3.5$) is in the top-K suggested items.
- **Mean Reciprocal Rank (MRR)@K:** MRR@K measures the average reciprocal rank of the first relevant item. Denoting k_i as the rank of the first relevant recommendation item, MRR@K is calculated as below:

$$MRR@K = \frac{1}{\#\ Users * K} \sum_{\#\ Users} \sum_{i=1}^{K} \frac{1}{k_i} \tag{10}$$

4.1.3 Baseline Methods. We compare the proposed HTPR with several state-of-the-art IRS methods from different types. The details are listed below:

- **Popularity:** It ranks the top k frequent items according to their popularity measured by the number of ratings, which is a simple but widely adopted baseline method.
- **SVD:** It suggests recommendations based on singular value decomposition (SVD). For IRS setting, the model is trained after each interaction with users and gives recommendations with the predicted highest rating.
- **DDPG-KNN:** It's a DDPG-based method that maps the discrete action space to a continuous one, then selects K nearest items in the continuous space with the max Q-value obtained by the critic network [4]. In our experiment, K value is set to $\{1, 0.1m\}$.
- **DQN-R:** It's a DQN-based method that adopts a refined DQN to evaluate the Q-values of the items and chooses the item with the max Q-value[31].
- **TPGR:** It adopts a tree-structured policy and uses the policy gradient method to optimize the tree-structured policy [3]. This is the state-of-the-art IRS approach and is similar to our proposed method.

Table 4: Efficiency evaluation of HTPR on different branch sizes

Dataset	Metric	16	32	64	128	256	512	m
Instant Video	HR@30	0.20463	0.20325	0.22270	**0.23673**	0.22872	0.23624	0.21453
	Average learning time per step	0.4031	0.35103	0.27738	**0.25329**	0.27081	0.26938	0.70964
	Average time per decision	0.3926	0.03904	0.02539	**0.02496**	0.02859	0.02646	0.04758
Baby	HR@30	0.13084	0.13071	0.12816	**0.14561**	0.14114	0.13788	0.1404
	Average learning time per step	0.40809	0.34899	0.33975	0.2475	0.2411	**0.21839**	1.14037
	Average time per decision	0.03055	0.03218	0.03488	0.02289	**0.02126**	0.02164	0.04894

Table 5: Efficiency evaluation of HTPR on different hidden sizes

Dataset	Metric	8	16	32	64	128
Instant Video	HR@30	0.20296	0.2227	0.23225	0.25362	0.2854
	Average learning time per step	0.27562	0.27738	0.27722	0.28438	0.29946
	Average time per decision	0.02432	0.02539	0.02539	0.02902	0.03563
Baby	HR@30	0.12782	0.12816	0.13117	0.1305	0.13032
	Average learning time per step	0.31829	0.33975	0.33717	0.34982	0.34445
	Average time per decision	0.02519	0.03448	0.03256	0.03792	0.02672

The experimental details including settings of both baseline methods and our method are presented in the appendix part at the end of this paper. Our implementations can be found at GitHub [1].

4.2 Recommendation Accuracy Evaluation (RQ1)

We first investigate the recommendation performance of HTPR against some state-of-the-art baselines w.r.t the aforementioned evaluation metrics. For the environment simulator, we fixed the length of each episode to 32. The hyper-parameters of all methods are tuned by grid search. In addition, to make a fair comparison our TPGR implemented with the state representation model learned online. The experiment results are shown in Table 2. In this table, the best results in each row are highlighted in bold.

From Table 2 We can infer that the proposed HTPR performs clearly better than the baseline methods on the two Amazon datasets using the aforementioned metrics. The conventional methods such as popularity and matrix factorization obtain bad performance under the three datasets. DDPG-KNN method performs worse in $K = 1$, and improves at $K = 0.1M$ in both Instant Video and Baby datasets, here M denotes the total number of items. Specifically, the proposed HTPR consistently outperforms the best-performing baselines from 6.57% to 34.6% in hit rate. Compared with TPGR, HTPR adopts simpler neural network functions to represent the probabilities of popular items while complicated neural network functions are used to represent the probabilities of uncommon items. Through this mechanism, HTPR is more effective for training the IRS than the balanced tree policy network.

4.3 Efficiency Evaluation (RQ2)

In this part, we evaluate the learning and decision efficiency of the HTPR compared with some RL-based recommendation methods. We also fixed the length of each episode in the environment simulator. We run each method on an AMD Ryzen 3600 6-core CPU with NVIDIA GeForce RTX2060 GPU. For learning efficiency, we report the average learning time per step (seconds) for HTPR against three RL-based recommendation methods: DDPG-KNN, DQN-R and TPGR. For decision efficiency, we report the average time per decision (seconds) of the same four methods.

The results are shown in Table 3. HTPR consumes much less time for both learning and decision-making compared with the 3 RL baselines. The overall efficiency of HTPR consistently and significantly outperforms the best-performing baseline TPGR from 32.4% to 46.1% with an average improvement of 41.9% in learning and decision. For DDPG-KNN, as the value of K affects the efficiency a lot, we investigate the two settings of K: $\{1, 0.1m\}$. The KNN method is very inefficient as K goes big. The proposed HTPR achieves better efficiency in both learning and decision-making against state-of-the-art methods.

4.4 Parameter Sensitivity Analysis (RQ3)

In this part, we evaluate the effect of two hyper-parameters: the branch of the node for Huffman tree policy network, and the hidden size for each node. For the branch of each node, it directly determines the total number of decision nodes of the Huffman tree. Given an n-ary Huffman tree with m nodes, and the branch is set to c, then based on the n-ary Huffman tree build algorithm, the number of decision nodes is $m \mod c - 1$, here $c > 2$. We report the performance (HR@30) and efficiency of 7 different branch sizes: $\{16, 32, 64, 128, 256, 512, m\}$. In this experiment, we fixed the hidden size of the neural networks as 64. The experimental results are shown in Tabel 4. As the branch size increases, the performance of HTPR improves at first (The two datasets perform best at $c = 128$). Then the performance begins to fall down when $c > 128$. The learning time and efficiency time follow a similar pattern to performance. At $c = 128$ and the Instant Video dataset achieves the best learning and decision efficiency, while the Baby dataset obtains the best

[1] https://github.com/shilx001/HuffmanTreePolicy

efficiency at $c = 512$. Specifically, the Huffman tree degenerates to a single node when c equals the item size. In such settings, the learning time and decision time become inefficient. Consequently, when deploying the HPGR into practice, a suitable branch size is worth tuning.

We also report the performance of different hidden sizes for each decision node in HTPR. In this experiment, we fix the branch size as 64. The result is shown in Table 5. For the Amazon instant video dataset, the performance improves as the hidden size increases. However, the efficiency is downgraded as the larger hidden size consumes more computation. In Amazon baby dataset, the performance improves when the hidden size is smaller than 32, but holds on a similar level when the hidden size is larger than 32. It means that the hidden size set to 32 is enough to tackle the IRS tasks. Therefore, in practice, we need to find a hidden size that can balance performance and efficiency.

5 CONCLUSION

In this paper, we propose a novel Huffman-tree Policy Recommendation (HTPR) framework. HTPR is an efficient policy learning framework for addressing the large discrete action space problem in RL-based IRSs, which has not been well addressed in the literature. HTPR is built on a newly designed Huffman tree structure, which is highly efficient for policy representation learning by utilizing different learning strategies on frequent and infrequent items in the dataset. In addition, a novel parameter-sharing scheme is designed to further improve learning efficiency by reducing unnecessary computations. Extensive experiments on two real-world benchmark datasets demonstrate that HTPR consistently and significantly outperforms the state-of-the-art IRS methods in terms of both recommendation accuracy and efficiency. One deficiency of HTPR is that it needs to estimate the occurrence of candidate items, which is not always applicable to online recommendation tasks where new items may keep coming. In the future, we will improve HTPR to address this issue.

APPENDIX: EXPERIMENT DETAILS

For the two Amazon datasets, the length of each episode is set to 32 and the discount factor γ in RL is searched over $\{0.6, 0.9, 0.99, 1\}$ and we found that $\gamma = 1$ obtains the best performance for most methods.

For HTPR, we use a single-layer neural network with 64 neurons as the node of the tree policy network. In the attentive state representation model, the hidden size for the fully-connected layer is equal to the hidden size of the decision node of the Huffman tree. The learning rate is searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$. For the matrix factorization method, we decompose the user-item matrix to obtain 128-dimensional user and item feature vectors. Additionally, The user rating (with range $[a, b]$) is mapped to a one-hot vector with a mapping function as below:

$$onehot\_mapping(rating) = onehot(l - floor(\frac{l \times (b - rating)}{b - a}), l)$$

where $floor(x)$ returns the largest integer no greater than x and $one\_hot(i, l)$ returns an l-dimensional vector with the ith elements is set to 1 while others are set to 0.

For DDPG-KNN, we use 3-layer neural networks to represent the actor-network and critic network. The hidden size of the neural network is set to 64 for each layer. The learning rates for both actor and critic are searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$ to obtain the best performance. The size of the experience replay is set to 1,000,000 and the batch size of each learning step is set to 64. The soft-update parameter τ for DDPG is set to 0.05. To improve the search efficiency of KNN, we use FLANN [12] for implementation.

For DQN-R, we also use a 3-layer neural network to represent the Q-network. The hidden size of the neural network is set to 64 for each layer. The learning rate for Q-network is searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$ to obtain the best performance. The soft-update period is set to 500 steps. In addition, the size of the experience replay is set to 1,000,000 and the batch size of each learning step is set to 64.

The implementation of TPGR is similar to [3], except some hyperparameters are well-tuned. To make a fair comparison, both state representation models of TPGR and HTPR are trained online. The learning rate for TPGR is searched over $\{1e-2, 1e-3, 1e-4, 1e-5\}$.

ACKNOWLEDGMENTS

This work is supported by the Natural Science Foundation of Zhejiang Province (Grant No. LQ22F020014) and Zhejiang Provincial Key Research and Development Program of China (Grant NO. 2021C01164).

REFERENCES

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.
[2] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. 2019. Learning action representations for reinforcement learning. In *International Conference on Machine Learning*. PMLR, 941–950.
[3] Haokun Chen, Xinyi Dai, and et al. 2019. Large-scale interactive recommendation with tree-structured policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3312–3320.
[4] Gabriel Dulac-Arnold, Richard Evans, and et al. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
[5] Simon Funk. 2016. Netflix update: Try this at home. *http://sifter.org/simon/journal/20061211.html* (2016).
[6] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 549–558.
[7] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.
[8] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[9] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 43–52.
[10] Andriy Mnih and Geoffrey E Hinton. 2008. A scalable hierarchical distributed language model. *Advances in neural information processing systems* 21 (2008), 1081–1088.
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[12] M Muja and D Lowe. 2013. Fast library for approximate nearest neighbors (FLANN). *git://github. com/mariusmuja/flann. git. url: http://www. cs. ubc. ca/research/flann* (2013).
[13] Feiyang Pan, Qingpeng Cai, Pingzhong Tang, Fuzhen Zhuang, and Qing He. 2019. Policy gradients for contextual recommendations. In *The World Wide Web Conference*. 1421–1431.
[14] Yoon-Joo Park. 2012. The adaptive clustering method for the long tail problem of recommender systems. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (2012), 1904–1915.

[15] Yoon-Joo Park and Alexander Tuzhilin. 2008. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*. 11–18.
[16] Jason Pazis and Ronald Parr. 2011. Generalized value functions for large action sets. In *ICML*.
[17] Brian Sallans and Geoffrey E Hinton. 2004. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research* 5 (2004), 1063–1088.
[18] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4902–4909.
[19] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[20] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.
[21] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. 2018. Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[22] Robin Van Meteren and Maarten Van Someren. 2000. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, Vol. 30. 47–56.
[23] Shoujin Wang, Longbing Cao, Liang Hu, and et al. 2021. Hierarchical attentive transaction embedding with intra- and inter-transaction dependencies for next-item recommendation. *IEEE Intell. Syst.* 36, 4 (2021), 56–64.
[24] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. 2021. A Survey on Session-Based Recommender Systems. *ACM Comput. Surv.* 54, 7, Article 154 (jul 2021), 38 pages. https://doi.org/10.1145/3465401
[25] Shoujin Wang, Liang Hu, Longbing Cao, and et al. 2018. Attention-based transactional context embedding for next-item recommendation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
[26] Shoujin Wang, Xiuzhen Zhang, Yan Wang, Huan Liu, and Francesco Ricci. 2022. Trustworthy Recommender Systems. *arXiv preprint arXiv:2208.06265* (2022).
[27] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.
[28] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 95–103.
[29] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1040–1048.
[30] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. 2013. Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 1411–1420.
[31] Guanjie Zheng, Fuzheng Zhang, and et al. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*. 167–176.
[32] Sijin Zhou, Xinyi Dai, Haokun Chen, and et al. 2020. Interactive recommender system via knowledge graph-enhanced reinforcement learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 179–188.
[33] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.