International Journal of Applied Science and Engineering

Special Issue-12th International Conference on Awareness Science and Technology (iCAST 2023)

Building the multi-objective periodic recommendation system through integrating optimistic linear support and user clustering to multi-object reinforcement learning

Dang Tien Dat [1], Nguyen Anh Minh [1], Tran Ngoc Thang [1], Rung-Ching Chen [2], Nguyen Linh Giang [3], Nguyen Thi Ngoc Anh [1*]

[1] *Faculty of Mathematics and Informatics, Hanoi University of Science and Technology, Vietnam*
[2] *Department of Information Management, Chaoyang University of Technology, Taiwan*
[3] *School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam*

ABSTRACT

Our study focuses on the diversity of user preferences and the dynamics of the user-product relationship, particularly in the context of periodic product usage. The principal objective of this research is to explore multi-objective optimization for a recommendation system tailored to periodic products. Our methodology employs a multi-objective reinforcement learning (MORL) algorithm. Additionally, we have proposed integrating the optimistic linear support algorithm into a MORL algorithm to collect good weight vectors. We also proposed using user clustering to ensure the model remembers user's preferences in early episodes. The findings of this research demonstrate that our proposed multi-objective approach yields significantly higher effectiveness when contrasted with conventional single-objective methodologies.

Keywords: Deep reinforcement learning, Multi-objective recommendation system, Optimistic linear support, Periodic product

OPEN ACCESS

Received: December 31, 2023
Revised: March 3, 2024
Accepted: June 3, 2024

Corresponding Author:
Nguyen Thi Ngoc Anh
anh.nguyenthingoc@hust.edu.vn

Publisher:
Chaoyang University of Technology
ISSN: 1727-2394 (Print)
ISSN: 1727-7841 (Online)

1. INTRODUCTION

Recommender systems (RS) have become a highly studied and implemented field in numerous domains in recent years. The main goal of RS is to offer a customized experience and enhance the interaction between users and the system by using interaction history, users' demographic information, and other factors to choose the most appropriate products from a large inventory. RS can be considered a pivotal technology that has played a significant role in the creation of billion-dollar empires like YouTube (Covington et al., 2016), Amazon (Linden et al., 2003), Netflix (Gomez-Uribe and Hunt, 2015), and others.

Periodic products (periodic items) refer to items that can be purchased and selected by customers on a predetermined cycle. These products are also recommended based on a schedule while ensuring their essential presence in the market. In contrast to non-periodic products, such as those found on e-commerce platforms or in the entertainment industry, where customers rarely repurchase or select the same item, and these products are seldom resold in a cyclical manner. An example of periodic products is telecommunications packages. These are monthly subscription-based products that are automatically

renewed, providing customers with the option to continue, cancel, or switch to a different package at their discretion. Some other examples include regularly used supplements or seasonal household products and clothing. In this paper, we focus on optimizing the recommender system for a group of products known as periodic products.

Multi-object is understood as needing more than one goal in each problem we want to solve, especially in the problem of RS, multi-object is directly related to the product suggested to customers usually a single goal is to suggest products that match the user's preferences. However, with a variety of products, some other potential goals can be targeted such as suggesting high-value, profitable products and suggesting new products. In particular, in the problem of suggesting phone packages, applying a multi-object solution will help customers receive a suitable package, which can bring a lot of profit to both service providers.

In 2021, Zheng and Wang (2021) conducted a survey on multi-objective recommendation system (MORS), where not only the accuracy of predicting user-preferred products but also the diversity, novelty, and fairness of the system were modeled under objective functions and simultaneously optimized during training. There are two main approaches to solving multi-objective optimization problems in recommender systems: Scalarization (Jahn, 1985) and multi-objective evolutionary algorithms (MOEA) (Von Lucken et al., 2014). For the first approach, a commonly used method is weighted sum, where objective functions are linearly combined into a sum function L, and the recommendation system is optimized based on L. Wu et al. (2022) developed a fairness-aware recommendation system that tailors recommendations to meet user needs while ensuring fairness across different demographic user groups. Cui et al. (2017) utilized the NSGA-II algorithm to avoid conflicts when optimizing both accuracy and diversity objective functions for recommended products simultaneously.

Reinforcement learning (RL) is a subfield of machine learning that investigates the optimal decision-making process of an agent within a given environment in order to maximize a certain long-term reward (Arulkumaran et al., 2017). The combination of RS and RL has recently provided new opportunities to improve the accuracy of predicting client preferences for items (Afsar et al., 2022). The unique ability of a RL agent to gain knowledge from incentives provided by the environment without the need for training data makes RL especially suitable for handling recommendation problems. When discussing the construction of RS, the term “agent” refers to the system itself, the “environment” refers to the user-item dataset, and the “reward” represents the level of satisfaction the user experiences with the recommendation (Hou et al., 2023). Leading IT firms are implementing RS based on RL to improve the immediate user experience and consistently refresh recommendations for the most appropriate products (Mulani et al., 2020).

The first attempt to utilize RL to enhance the quality of recommendations was likely WebWatcher (Joachims et al., 1997). The authors framed the site recommendation problem as a RL problem and employed Q-learning to improve the precision of their underlying site recommendation system. They utilized a similarity algorithm, which relied on the TF-IDF index, to recommend pages that demonstrate a substantial degree of resemblance to the user’s interests. There is a growing emphasis on recommendation models that include deep learning approaches, such as deep Q-network (DQN), double deep Q-network (DDQN), or deep deterministic policy gradient (DDPG). The utilization of DDQN in deep recommendation news (DRN) (Zheng et al., 2018) tackles the problem of volatile changes in news content and user preferences. They utilized two Q networks and partitioned the Q function into distinct value and advantage functions; in addition to the current indices, such as click through rate (CTR), the incentives function now incorporates a new index termed user engagement. This index quantifies the rate at which a user engages with the system. Instead of employing greedy algorithms with epsilon, the upper confidence bound is utilized for probing. The list-wise recommendation framework based on deep reinforcement learning (LIRD) (Zhao et al., 2017) initially utilized a random environment simulator to generate recommendations. They subsequently employed an actor-critic framework to create these recommendations. The simulator generated a reward by considering a specific combination of state and action, employing the principle of collaborative filtering. This method implies that individuals with common interests are likely to have a shared interest in similar products. Furthermore, DDPG was utilized for parameter optimization. Within the deep reinforcement recommendation (DRR) architecture (Liu et al., 2018), the agent acquires a state from the state representation module and generates actions through the utilization of two rectified linear unit (ReLU) layers and one hyperbolic tangent (Tanh) layer. The state module comprised three structures: item-based items, user-based items, and user-based average items. The critic utilized a DQN module consisting of two rectified linear unit (ReLU) layers to evaluate the action generated by the actor. In the end, the DDPG method was used to train the model.

In recent years, there have been numerous studies related to multi-objective reinforcement learning (MORL). The goal of MORL is to learn a policy that optimizes multiple objectives where these objectives may conflict with each other (Roijers et al., 2013). Unlike traditional RL, where the agent optimizes a policy to achieve an optimal scalar reward value, an agent in MORL learns to achieve an optimal vector reward value based on predefined preference criteria among multiple objectives (Hayes et al., 2022). However, there have been few applications of MORL in RS due to the complexity of designing objective functions and creating reward functions that simultaneously ensure the accuracy of predictions and other metrics like fairness, diversity, ... of

recommended products (Paparella et al., 2023). However, in order to accomplish both goals, a differentiable function that takes into account both objectives is necessary, which might be challenging. Additionally, applying RL to RS faces challenges when dealing with an immensely large number of products and users, making the modeling of states particularly challenging (Song et al., 2014). Furthermore, when utilizing MORL frameworks, one must generate corresponding sets of weights for each objective so that the agent can prioritize the selection of products aligned with those objectives (Hayes et al., 2022). However, creating an excessively large search space for weights can escalate the cost and training time of the model.

Modeling the recommendation of routes as a Markov decision process (MDP) has introduced a new direction in applying MORL to location-based and personalized route recommendations (Sarker et al., 2020; Stamenkovic et al., 2022). Sarker et al. (2020) constructed a multi-objective route recommendation system to optimize travel routes with respect to time, cost, and air quality simultaneously. A Q-learning network was trained to maximize rewards, considering user preferences and meeting multiple criteria while simulating real geo-tagged data sources (traffic, weather, points of interest, and GPS traces). Chen et al. (2023) employed an actor - critic network to generate optimal travel routes, simultaneously considering accuracy, popularity, and diversity for locations, thereby maximizing the composite reward. Stamenkovic et al. (2022), for the first time, applied the MORL framework to create three Q-value functions representing the accuracy, novelty, and diversity of recommendations. The system optimization was performed through scalarized deep Q-learning. Constructing multiple rewards corresponding to each criterion ensures that the system maintains accuracy in predicting products while ensuring other criteria such as fairness, novelty, diversity, etc. Recently, Keat et al. (2022) proposed a framework applying deep reinforcement learning (DRL) to optimize MORS with the aim of creating an optimal system across conflicting metrics such as precision, novelty, and diversity. The authors utilize DQN to address the optimization problem for MORS, achieving higher performance compared to optimizing recommender systems using multi-objective optimization algorithms like the probabilistic-based multi-objective approach based on evolutionary algorithms (PMOEA).

In this paper, we will employ a novel MORL framework to apply it to the construction of a recommender system. The principal contributions of our study are delineated as follows:

- Our research offers a comprehensive solution for the effective implementation of reinforcement learning algorithms within recommender systems. This involves the meticulous organization of data, the precise definition of state vectors, actions, and rewards that are closely aligned with user preferences, and the careful preparation of training datasets.
- We elaborate on the implementation of a user clustering mechanism. This strategy is designed to expedite the training process and enhance the efficacy of the training outcomes. We introduce a unique representation for each user based on their preference vector, taking into account the frequency of interactions to identify the most representative users.
- The study goes deeper into explaining the view of leveraging the optimistic linear support (OLS) algorithm in the MORL problem, and then our setup integrated OLS algorithm to envelope multi-objective Q-learning (EMOQ) algorithm to improve the training process.

In the remainder of this article, this paper presented basic knowledge about multi-objective optimization problems, reinforcement learning, MORL, and the optimistic linear support algorithm. In section 2.2 - Proposed method about modelling to reinforcement learning problem, we will divide it into two parts: "modeling based on reinforcement framework" details the modeling process, training, and applying the trained model to the recommender system; Part 3 of section 2, "multi-object recommendation based on RL" will present the application of the OLS algorithm to the EMOQ algorithm to solve the MORL problem, and this part also details the construction of the user clustering block. Finally, in section 3 – Result and Discussion, we will present the results on real data sets, both single-object and multi-object cases.

2. MATERIALS AND METHODS

2.1 Materials and Methods

2.1.1 Multi-objective problem

Consider the unconstrained multi-objective optimization problem (MOP):

$$Min_{\theta\in\mathbb{R}^n}\mathcal{F}(\theta) = Min_{\theta\in\mathbb{R}^n}\begin{bmatrix}\mathcal{F}_1(\theta)\\ \mathcal{F}_2(\theta)\\ \vdots\\ \mathcal{F}_m(\theta)\end{bmatrix},$$

where $\mathcal{F}: \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differentiable vector function, m is the number of objectives and $\theta \in \mathbb{R}^n$.

A solution θ^* is said to dominate another solution θ if $\forall \mathcal{F}_i(\theta^*) \le \mathcal{F}_i(\theta), i = 1, \dots, n$, and there exists $\mathcal{F}_j(\theta^*) < \mathcal{F}_j(\theta), i = 1, \dots, m$. A solution θ^* is a Pareto solution if there is no solution θ that dominates θ^*; θ^* is a local Pareto solution of the (MOP) if there exists a neighborhood $U(\theta^*) \subset \mathbb{R}^n$ such that $\nexists \theta \in U(\theta^*): \mathcal{F}(\theta) < \mathcal{F}(\theta^*)$.

In order to address the multi-objective optimization problem, a viable approach is to employ a gradient-based optimization strategy. Prior to introducing the algorithm, we will elucidate the concept of common descent vectors and outline the ideal circumstances for gradient-based solutions in MOP. A conventional descent vector can be expressed as a convex amalgamation of the gradients of each objective. It can be accurately delineated as follows:

$\nabla_\theta \mathcal{F}(\theta) = \sum_{i=1}^{n} \alpha_i \nabla_\theta \mathcal{F}_i(\theta)$ [*]

In the equation, n represents the number of objectives, θ denotes the model parameters, $\nabla_\theta \mathcal{F}(\theta)$ stands for the common descent vector, $\nabla_\theta \mathcal{F}_i(\theta)$ is the gradient of the i^{th}

objective function, and α_i represents the weight of the i^{th} gradient. Equation [*] adheres to the following conditions:

1. $\alpha_1, \dots, \alpha_n \geq 0$
2. $\sum_{i=1}^{n} \alpha_i = 1$

2.1.2 Reinforcement learning

RL addresses decision-making problems using a mathematical model known as Markov decision process (MDP). A MDP is characterized by five components in the following manner: $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot|\cdot,\cdot), R(\cdot,\cdot,\cdot), \gamma)$ which are the state space, action space, transition probability model, reward model and discount factor, respectively.

For a MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot|\cdot,\cdot), R(\cdot,\cdot,\cdot), \gamma)$, the goal is to construct a policy π to optimize the long-term rewards (also known as the *return*). The return G_t with a discount factor γ is defined as:

$$G_t = R_t + \gamma R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where G_t is the sum of all rewards received from state s_t into the future, where $0 < \gamma < 1$ discounts future rewards, making rewards further in the future less significant. Corresponding to the components of MDP, there are five terms in RL: agent, environment, state, action, and reward. Because often in real problems, the model of the environment is not given, so current RL algorithms are model-free algorithms, including traditional algorithms Q-learning, and state-action-reward-state-action (SARSA) (Sutton and Barto, 2018) to advanced algorithms. More modern mathematics when integrating deep learning such as deep Q-network (Mnih et al., 2013), dueling deep Q-network (Wang et al., 2016).

2.1.3 Multi-objective reinforcement learning problem and some approaches

Similar to the MDP, a multi-objective Markov decision process (MOMDP) can be represented by a set of values $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Phi, g_\phi)$ where the 3 quantities $(\mathcal{S}, \mathcal{A}, \mathcal{P})$ have the same meaning as in the conventional MDP while the reward function $\mathcal{R}$ returns the reward in the form of an m-dimensional vector with m being the number of objects need to optimize.

$$r(s,a) = \left(r_i(s,a)\right)_{i=\overline{1,m}}^{T} = \begin{pmatrix} r_1(s,a) \\ r_2(s,a) \\ \cdots \\ r_m(s,a) \end{pmatrix} \in \mathbb{R}^m$$

The term Φ is the space of the weight vector ϕ while g_ϕ is the preference function that functions to convert the reward vector to a scalar value. This preference function plays an important role in iterative algorithms when supporting choosing actions for the agent. For example, we have a linear preference function:

$$g_\phi\big(\mathrm{r}(s,a)\big) = \phi^T \mathrm{r}(s,a)$$

Under the change from scalar to vector of reward term, other important terms of the Markov process such as the return function of a trajectory, the value function of a state, or the Q-value function of a pair action - state also change. Specifically, the return value $G(\tau)$ of the trajectory $\tau = (s_i, a_i)_{i=1,2,\dots,T-1} \in (\mathcal{S} \times \mathcal{A})^{T-1}$ is the object function of MOMDP problem:

$$\mathrm{G}(\tau) = \mathrm{r}_0 + \gamma \mathrm{r}_1 + \gamma^2 \mathrm{r}_2 + \dots \gamma^{T-1} \mathrm{r}_{T-1} = \sum_{t=0}^{T-1} \gamma^t \mathrm{r}_t$$

Because the trajectory τ consists of many states and actions while the state and action space is often large, even continuous, and the environment models are also unknown, RL-based algorithms are often used. From the MOMDP problem, put into real-life problems, we obtain the MORL problem where the reward term is vector instead of scalar. The two most common types of algorithms are the single-policy and the multi-policy algorithm. For large state and action spaces, the envelope multi-objective Q-learning (EMOQ) (Yang et al., 2019) algorithm below is very effective.

To simplify notation, let $\hat{\mathrm{r}} = \mathrm{G}(\tau)$. Then, the set of return values is considered a Pareto surface set, defined as the set of satisfactory return vectors with no return vector being dominant.

$$\mathrm{C}^* = \left\{\hat{\mathrm{r}} | \nexists \widehat{\mathrm{r}'} \succcurlyeq \hat{\mathrm{r}}\right\}$$

where $\hat{\mathrm{r}} = \sum_t \gamma^t \mathrm{r}(s_t, a_t)$. We aim to find a policy to optimize the expected value of the vector reward $\mathrm{V}^{\pi} = \mathrm{E}(\hat{\mathrm{r}})$. A primary approach to solving the multi-objective optimization problem for V^{π} is scalarization. When employing a linear function class for scalarization of V^{π}, the obtained optimal solution set is the convex coverage sets (CCS) (Roijers et al., 2013). The CCS set is a subset of the Pareto surface value set, and for each point in the CCS, there always exists a weight vector that makes the corresponding scalar value the largest.

$$\mathrm{CCS} = \left\{\hat{\mathrm{r}} \in \mathrm{C}^* | \exists \phi \in \Phi, \phi^T \hat{\mathrm{r}} \geq \phi^T \widehat{\mathrm{r}'}, \forall \widehat{\mathrm{r}'} \in \mathrm{C}^*\right\}.$$

2.1.4 The envelope multi-objective Q learning algorithm

The envelope multi-objective Q learning algorithm (Yang et al., 2019) is a method developed to address the MORL challenge. This algorithm is essentially an extension of deep reinforcement learning algorithms, which are typically used in single-objective contexts, but it modifies the updated formula of the Q-network to suit scenarios involving multiple objectives.

Specifically, looking from the perspective of contraction mapping and using the Banach fixed point theorem (Yang et al., 2019), the following mapping $\mathcal{C}$ of the Q learning algorithm is a contraction mapping and with each tuple (state, action) (s,a), the Q value gradually convergence to the fixed point or the fixed value which is an exactly Q value of that tuple. The definition of the contraction mapping $\mathcal{C}$ and the fixed point of each tuple (s,a) is illustrated by the following formula:

$$(\mathcal{C}Q)(s,a) = r(s,a) + \gamma E_{s' \sim \mathcal{P}(.|s,a)} \sup_{a' \in \mathcal{A}} Q(s',a')$$

$$Q^*(s,a) = r(s,a) + \gamma E_{s' \sim \mathcal{P}(.|s,a)} \sup_{a' \in \mathcal{A}} Q^*(s',a')$$

Along with other theorems about "generalized banach fixed-point theorem" in that paper, the authors proved that the Q value can still converge to a fixed point. The following formula is expression for updating the Q value at k^{th}-loop.

$$Q^{k+1}(s,a) = \mathcal{C}Q^k(s,a)$$

In the multi-object problem, Yang et al. (2019) redefined variables such as contraction mapping, Q value space, metric and fixed point and from there developed the following convergence theorems:

1. The space $\mathcal{X}$: is the Q-value space $\mathcal{Q} \subseteq (\Phi \to \mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}}$, typically, the Q value vector is determined by a tuple including three terms $(s,a,\phi) \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}\times\Phi}$ and $\mathrm{Q}(s,a,\phi) \in \mathbb{R}^m$. Therefore, in the spirit of considering $\mathcal{Q}$ as a matrix of size $\mathcal{S}\times\mathcal{A}$, the value of each position $\mathrm{Q}(s,a)$ will be considered as a function of the vector of the number ϕ or $\mathrm{Q}(s,a) = h(\phi): \Phi \mapsto \mathbb{R}^m$.
2. Metric d: metric in the space $\mathcal{Q}$ above is defined as follows:
$$d_{EMOQ}(\mathrm{Q},\widehat{\mathrm{Q}}) = \sup_{s\in\mathcal{S},a\in\mathcal{A},\phi\in\Phi} \left|\phi^T[\mathrm{Q}(s,a,\phi) - \widehat{\mathrm{Q}}(s,a,\phi)]\right|$$
notice that the metric is pseudo-metric and the space $(\mathcal{Q},d)$ is the complete metric space. The value $\phi^T\mathrm{Q}$ can be understood as the compatibility between the Q-value vector Q and the weight vector ϕ.
3. The mapping $\mathcal{C}$: the mapping $\mathcal{C}$ is defined as follows:
$$(\mathcal{C}\mathrm{Q})(s,a,\phi) = \mathrm{r}(s,a) + \gamma E_{s'\sim\mathcal{P}(.|s,a)}(\mathcal{V}\mathrm{Q})(s',\phi)$$
where $\mathcal{V}$ is called the optimal filter and has the role of finding the convex hull of the current Pareto surface:
$$(\mathcal{V}\mathrm{Q})(s',\phi) = \arg_{\mathrm{Q}} \sup_{a'\in\mathcal{A},\phi'\in\Phi} \phi^T\mathrm{Q}(s,a',\phi')$$
where $\arg_Q$ is the vector $\mathrm{Q}(s',a^*,\phi^*)$ where
$$(a^*,\phi^*) = \mathrm{argmax}_{a'\in\mathcal{A},\phi'\in\Phi}\phi^T\mathrm{Q}(s,a',\phi').$$
Besides, the mapping $\mathcal{V}$ is called an optimal filter for each pair (s,ϕ) then $\phi^T(\mathcal{V}\mathrm{Q})(s,\phi) \geq \phi^T\mathrm{Q}(s,a,\phi)\forall a \in \mathcal{A}$ so the set of points determined by the filter $\mathcal{V}$ will form the convex hull of all Q values, thus also convex hull of the current set of Pareto surfaces. Or in other words, for each weight vector ϕ, $\mathcal{V}$ will create the set of "most relevant" points, thus $\mathcal{V}$ is the convex hull.
4. The fixed point will be defined by the Q vector that maximizes the scalar product between the weight vector ϕ and the expected vector of return value:
$$\mathrm{Q}^*(s,a,\phi) = \arg_Q \sup_{\pi\in\Pi} \phi^T\mathbb{E}_{s'\sim\mathcal{P}(.|s_0=s,a_0=a)}\left[\sum_{t=0}^{\infty}\gamma^t\,\mathrm{r}(s_t,a_t)\right]$$

There are three theorems in proved to show that Q^* is the fixed point that means $\mathcal{C}\mathrm{Q}^* = \mathrm{Q}^*$, $\mathcal{C}$ is a contraction mapping and for any point $\mathrm{Q} \in \mathcal{Q}$ will be convergence to Q^* where $\lim_{n\to\infty} d_{EMOQ}(\mathcal{C}^n\mathrm{Q},\mathrm{Q}^*) = 0$.

The algorithm operates according to the same main steps as regular DRL. At each step in each episode in the DQN algorithm training process, we will need to determine a new action and then perform that action in the environment, obtaining a response including a reward. In the new state, save it to the replay buffer and randomly get data from the replay buffer for training. In addition, the EMOQ algorithm to solve the MORL problem requires additional steps to determine a weight vector in each episode. After the interaction, use the hindsight experience replay (HER) (Andrychowicz et al., 2017) algorithm to increase experience to make training more diverse. The new updating label in training is as follows:

$$\mathrm{Q}^{k+1}(s,a,\phi) = \mathcal{C}\mathrm{Q}^k(s,a,\phi) = \mathrm{r}(s,a) + \gamma E_{s'\sim\mathcal{P}(.|s,a)}(\mathcal{V}\mathrm{Q})(s',\phi)$$

and to apply updates according to each experience gained, the updated label is:

$$\mathrm{y}(s,a,\phi) = \mathrm{r}(s,a) + \gamma\arg_Q \max_{a'\in\mathcal{A},\phi'\in W}\phi^T\mathrm{Q}(s,a,\phi')$$

Where W is a set of weight vectors sampled according to a certain distribution, this method is called hindsight experience replay (HER) (Andrychowicz et al., 2017); the goal of using this method is to effectively support the process of using experience from replay buffers in the off-policy algorithm.

The loss function is also modified with the contribution of two functions:

$$\mathcal{L}^{\mathrm{A}}(\theta) = \mathbb{E}_{s,a,\phi}[||\mathrm{y} - \mathrm{Q}(s,a,\phi;\theta)||_2^2]$$
$$\mathcal{L}^{\mathrm{B}}(\theta) = \mathbb{E}_{s,a,\phi}[|\phi^T\mathrm{y} - \phi^T\mathrm{Q}(s,a,\phi;\theta)|]$$
$$\mathcal{L}(\theta) = (1-\lambda)\mathcal{L}^{\mathrm{A}}(\theta) + \lambda\mathcal{L}^{\mathrm{B}}(\theta)$$

The effectiveness of this algorithm is that after training, the Q network can flexibly find good enough policies with any weight vector corresponding to the goals. We will use this EMOQ algorithm as the main part to solve the MORL problem. In the following section, we will analyze in more detail how installing the OLS algorithm according to intuition will help the algorithm work well.

2.1.5 Optimistic linear support algorithm

A partial set of CCS, S is defined as a subset of CCS and is built based on an iterative algorithm. In this article, when using the OLS algorithm (Roijers et al., 2015), we will apply it on 2 objects, so each return vector (reward) will be a 2-dimensional vector.

A corner weight vector is determined based on the intersection of two reward vectors. Let the weight vector be w_{corner} and the two reward vectors be $u_1(a,b), u_2(c,d)$

$$w_1 = \frac{1}{|\frac{b-d}{a-c}|+1} \Rightarrow \mathrm{w}_{corner} = [1-w_1, w_1]$$
$$= \left[\frac{|b-d|}{|b-d|+|a-c|}, \frac{|a-c|}{|b-d|+|a-c|}\right]$$

The above definition is a way to determine a new corner weight vector from the intersection of two reward vectors. Based on this determination, the goal of the OLS algorithm is to build a CCS partial set quickly. The spirit of the OLS algorithm considers the new corner weight vector (determined according to the above definition) as potential, puts it into the problem obtains the corresponding reward vector, processes the selection through a number of conditions, and then repeats it. Continue the loop until there

are no more vectors among the potential angles that satisfy the condition. The OLS algorithm can be presented by the following Fig. 1 which illustrated for the pseudocode for OLS algorithm.

In the pseudocode part of OLS, `Vector Reward` (w) is a function that returns the reward vector corresponding to the set of weights w while the function `Potential Corner Weight`($\mathcal{W}_{del}, S$, u) has the role of updating the set S, determining the new corner weight vectors.

2.2 Proposed Method about Modelling to Reinforcement Learning Problem

Fig. 2 illustrates our proposed framework. Our framework consists of two main components: the RL block and the clustering block. The RL block is constructed from user information, product information, and interactions between users and products. This block generates four feature vectors to represent the dataset's information and the five essential components of the RL model: agent, environment, state, action, and reward. When applying RL to build a RS, we encounter the challenge of a vast number of users. Therefore, when training the RL model traditionally, treating each user as an environment may lead to significant training time. To address this, we cluster users, identify representative users, and train the RL model with reduced computational cost. Clustering users helps reduce computation costs, train the model with fewer user interactions, and accelerate policy convergence.

.

Algorithm 1 The original Optimistic Linear Support Algorithm

Input: The partial set CCS $S = \emptyset$, the set of tested vectors $\mathcal{W} = \emptyset$, $\mathcal{K}$: the initially empty row
Output The partial set CCS S

for e=1 to m **do**
  $\mathcal{K} = \mathcal{K} \cup \{(w_e, \infty)\}$
end for
while $\mathcal{K} = \emptyset$ **do**
  w$\leftarrow \mathcal{K}$.put()
  $u \leftarrow$ `VectorReward`(w)
  if $u \notin S$ **then**
    Determine the set of obsolete corner vectors $\mathcal{W}_{del} = \{\mathbf{w}' \in \mathcal{K} \,|\mathbf{w}'^T\mathbf{u}_i \geq \mathbf{w}_i^T\mathbf{u}_i\}$
    $\mathcal{W}_{del} \leftarrow \mathcal{W}_{del} \cup \{\mathbf{w}\}$
    $\mathcal{W}_u =$ `PotetialCornerWeight` $(\mathcal{W}_{\mathtt{del}}, \mathtt{S}, \mathbf{u})$
    $S \leftarrow S \cup \{\mathbf{u}\}$
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\mathbf{w}, \Delta_r(\mathbf{w})) \,|w \in \mathcal{W}_u, \Delta_r(\mathbf{w}) > \varepsilon\}$
  end if
  $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{w}\}$
end while

Fig. 1. The pseudo-code of the optimistic linear support (OLS) algorithm

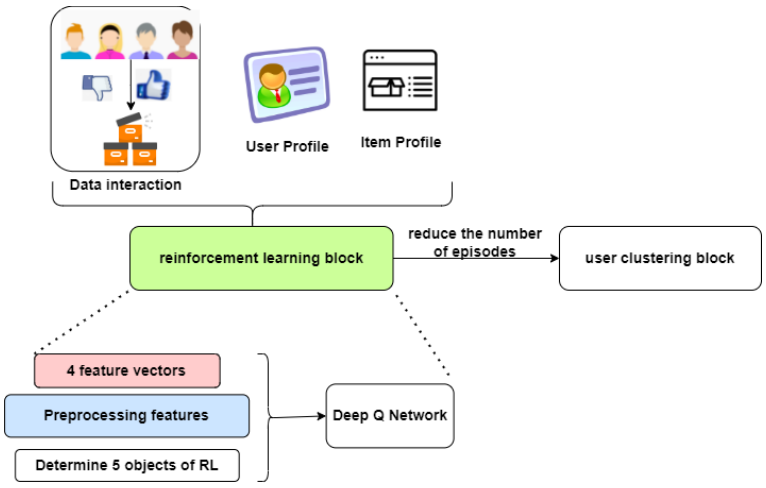


Fig. 2. Proposed framework

2.2.1 Creating four types of feature vectors

It is essential to create four unique kinds of feature vectors in order to effectively capture and describe information about things, such as users and items, as well as their interactions. These vectors attempt to offer effective representations, particularly for periodic product such as telecommunication package and user activity. There are four categories of feature vectors:

- Item vector:
  - Representation: I_i. Detail of item vector will be shown in Fig. 3.
  - Description: Every item in the dataset has distinct features, and vector I_i is used to represent the individual qualities of item i.

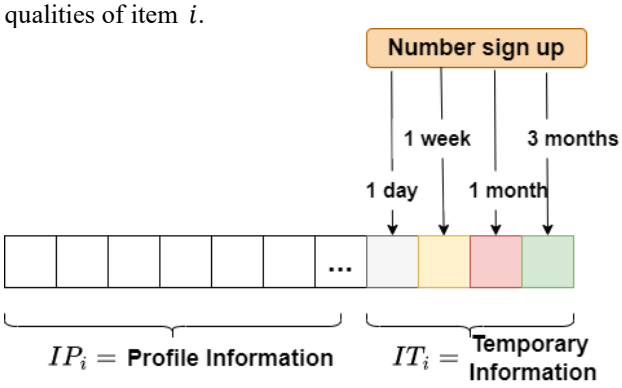


Fig. 3. Detail of item vector

- User transaction vector:
  - Representation: UT_u. Detail of User Transaction vector will be shown in Fig. 4.
  - Description: User preferences are sent by means of transactions, and UT_u is introduced for this specific objective. The time intervals included are "1 month ago" and "3 months ago" relative to the present moment. The vector records user preferences by creating lists of items that the user engaged with, connected to the corresponding item vectors.

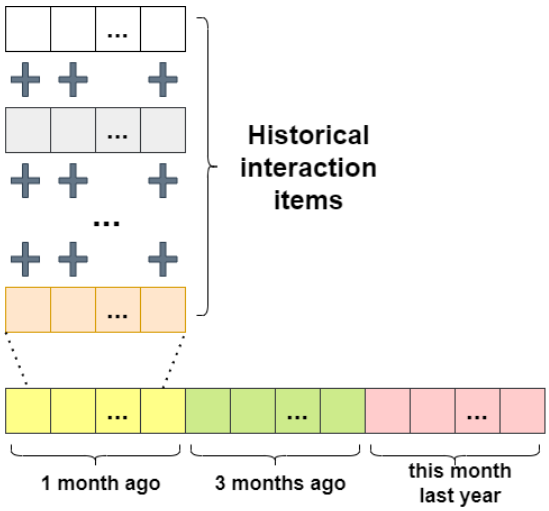


Fig. 4. Detail of user transaction vector

- User-item vector:
  - Representation: UI_{ui}. Detail of user transaction vector will be shown in Fig. 5.
  - Description: There are various ways to represent the information about the relationship between users and products. However, we utilize two factors: the quantity of product i purchased by user u and the ranking order of product i in the list of products that user u has purchased, sorted by the number of purchases.

Fig. 5. Detail of user transaction vector

- User vector:
  - Representation: U_u. Detail of user vector will be shown in Fig. 6.
  - Description: Incorporating both explicit feedback (e.g., subscription status) and hidden feedback (e.g., spending amount and data usage), the user vector U_u represents the combination of these features.

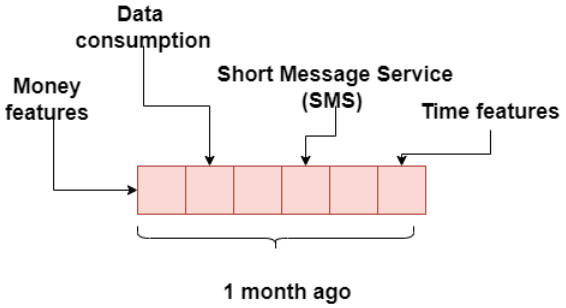


Fig. 6. Detail of user vector

2.2.2 Definition of 5 components

Fig. 7 illustrates the 5 essential components of every RL model.

System components will be specified as follows:

- Agent: The agent operates as a recommender, with the explicit aim of offering consumers recommended products.
- Environment: The environment is constructed from customer (user) information. The RL model will learn how to recommend products that best match the customer's preferences.
- Action: Actions are designated as the recommended items for consumers. The action is indicated by the combination of the item vector and the user-item vector $[I_i, UI_{ui}]$. The variable u denotes the customer receiving the service, whereas i represents the recommended item.
- State: States are used to represent customer interactions. We will combine the user vector and user transaction vector, denoted as $[UT_u, U_u]$.
- Reward: We define the reward in two cases: single-

objective and multi-objective. The reward formula for single-objective problems is expressed as the product of x and the exponentiated value of $(1-y)$ raised to the power of a. Here, a represents the time elapsed since the user u last interacted with the item i at the time of recommendation t. The parameter y is a small value less than 1, and x indicates the magnitude of the reward. The reward value falls within the ranges of $(0, x]$, and the reward will be higher when the model recommends the product that the user likes the most at time t, specifically when $a = 0$. In the case of the multi-objective reward function, in addition to the first component, which is the reward as in the single-objective case, we aim to maximize the daily profit for the company from package purchases. Therefore, we combine the information C_i representing the price and T_i representing the duration of using item i to create the second component of the reward.

Some advantages of our reward function are as follows:

- This is a completely new reward function for periodic items. Currently, there is not much RL research for this type of item.
- Continuous value easily accurately evaluates user feedback.
- This applies to items not selected at the time of suggestion because the user may not have seen the recommended item at that timestamp.
- The reward function has many meanings as a weighted sum, showing that items that have been used recently are more suitable to the user's preferences, while for items that have not been used for a long time, the reward is small.

After having the 5 RL components, we will use RL algorithms to train the model.

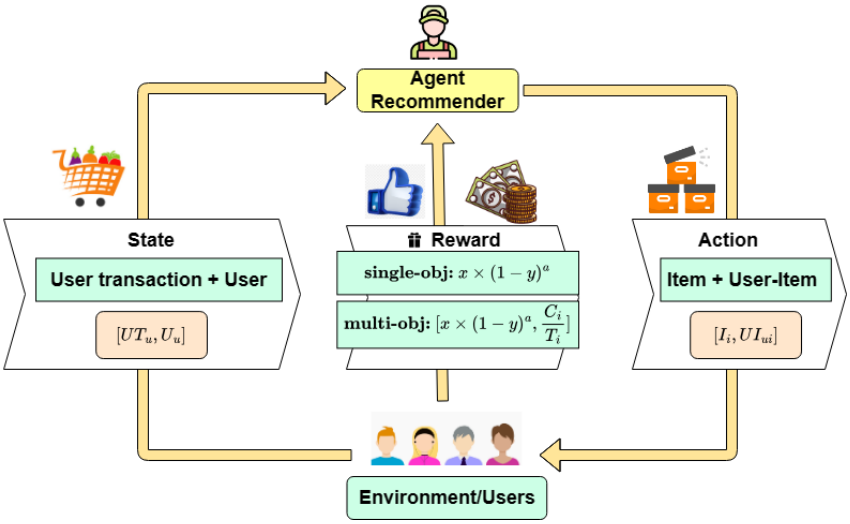


Fig. 7. System components

2.2.3 Using trained Q network to generate recommendation sets

Utilizing the RL approach for the recommendation problem necessitates a training process that is more intricate than that of typical RL problems. The reason stems from the delineation of each user's environment, which may encompass a subordinate sub-environment. Hence, it is imperative to accurately update the appropriate state for the user being attended to during the process of transitioning between states. Fig. 8 depicts the proposed Q network training process, which utilizes preexisting data, hence constituting an offline training methodology.

In the illustration above, two initial users are u_0 and u_1 with corresponding interaction counts M and N. Each step corresponds to t_{0i} with $i \in 1,2,\ldots,M$ for u_0, and t_{1j} with $j \in 1,2,\ldots,N$ for u_1. The PROCESS block is where the algorithm starts training, and at each step, there are three subtasks:

- Set up feature vectors: The feature vectors for the state of user u_0 at the current step $[UT_{u_0}, U_{u_0}]$ and the set of candidate items K_0 for recommendation to user u_0 with $K_0 = 2 \times NR_0$, where NR_0 is the number of items that user u_0 has interacted with in the entire interaction file. Since each user can only see a limited number of items at any given time, setting K_0 results in effective training.
- Process: After normalizing the state description vectors for user u_0: $[UT_{u_0}, U_{u_0}]$ and the set of items $K_0 = \left[I_{i_j}, UI_{u_0 i_j}\right] | j \in 1,2,\ldots,K_0$, these vectors are passed into a pre-designed Q network that outputs the item with the highest Q value ("highest" based on either real value in single-objective tasks or the inner item with test weights in multi-objective tasks) in that training set.
- Push recommendation sets to users and update their new states: The recommendation system, after obtaining the output of the Process, sends the recommended items to user u_0 at the current step. Then, it compares these recommendations with the actual interactions of user u_0 at that moment, calculates the reward to train the Q network, and updates the state of user u_0 for the next step.

It is crucial to acknowledge that the length of each set will vary because each user has a unique amount of interactions. Therefore, the epsilon value in the ϵ- greedy algorithm is modified individually for each user, adhering to a requirement of 50% exploration and 50% exploitation.

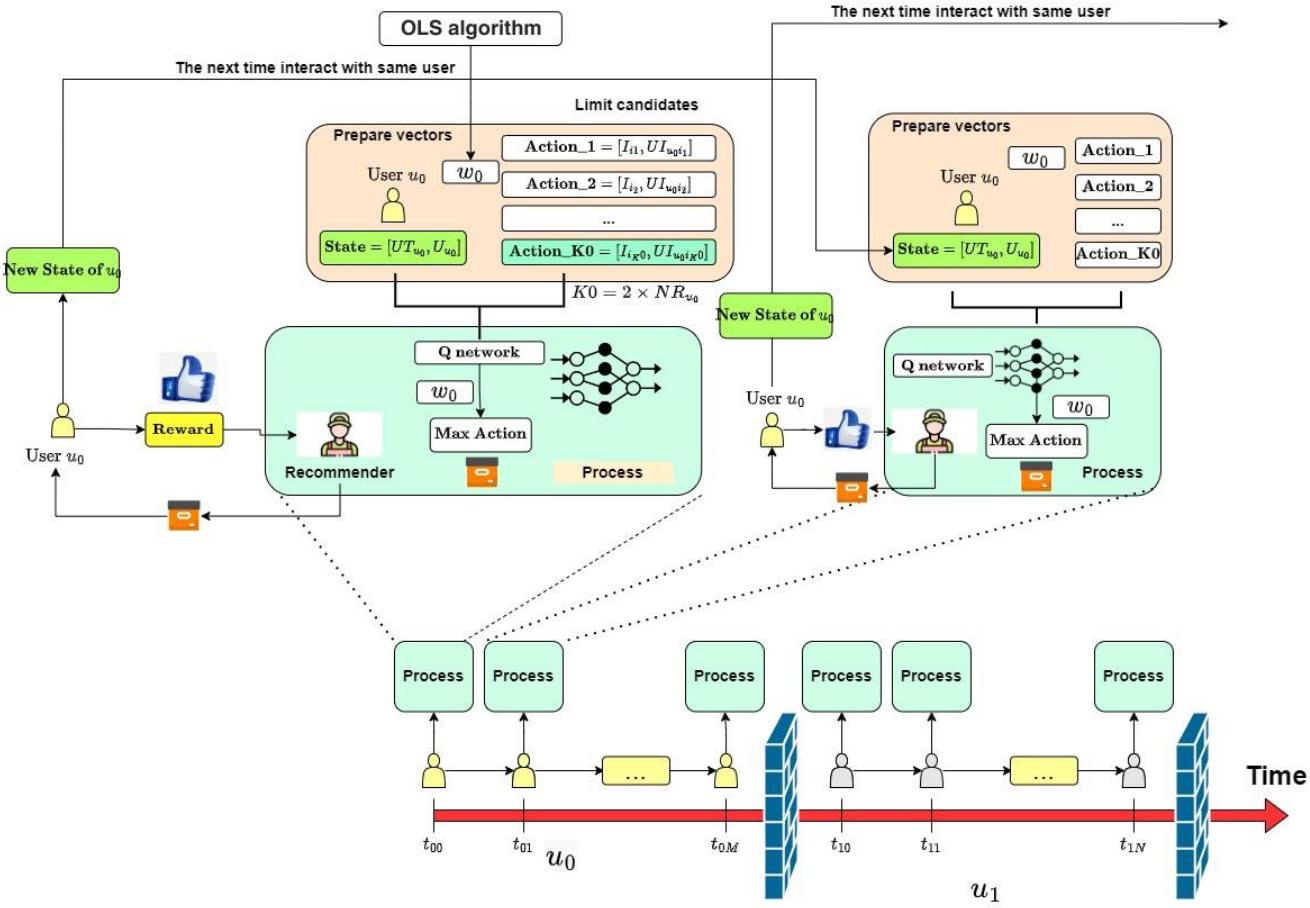


Fig. 8. Illustration of the training process of the algorithm

2.3 Proposed Method About Multi-object Recommendation Based on Reinforcement Learning

2.3.1 Fine-tuning and installing the OLS algorithm in training EMOQ

In the original EMOQ algorithm, at each episode, we choose a weight vector that follows a given distribution for testing, this can lead to a prolongation of training time while not being very effective because if we test nearly identical weight vectors will intuitively replicate quite similar experiences of interacting with the environment. Besides, the OLS algorithm provides a way to build a partial set of CCS, a set of "potential" weight vectors, and between the weight vectors there will be a distinction because each vector is calculated based on the intersection of reward vectors, intuitively this will help each experience when exploring the environment be different, leading to more effective training.

For MORL problems, the calculation for the reward vector is often "dynamic" $(\mathrm{u} \leftarrow \mathtt{Vector\ Reward}(\mathrm{w}))$ because the environment usually has the stochastic factor. This leads to some differences when wanting to apply OLS to the MORL problem:

- Note that at each execution of the OLS algorithm, only one result is obtained from the small set of the CCS due to each weight vector can corresponding to many reward vectors, for example, the simple vectors $\mathrm{w_e} = [1,0]$ (are vectors that only focus on a single target) then vector. The corresponding reward will be of the form $(\max obj_1, x)$ for any x belonging to the admissible set. Furthermore, because the environment in real life as a recommendation system always has a stochastic factor, so performing an action only once can't obtain the expected reward vector, which makes some errors in evaluation.
- Set of "outdated" corner weight vectors $\mathcal{W}_{del}$ helps eliminate ineffective corner weight vectors to reduce training time, however, due to the above-mentioned stochastic property of the environment, the weight vector $\mathrm{w_i}$ which is determined by the intersection of two reward vectors is not ensured be the lowest corner weight vector.
- Computing the new corner weight vector $\mathcal{W}_u = \mathtt{Potential\ Corner\ Weight}(\mathcal{W}_{\mathrm{del}}, \mathrm{S}, \mathrm{u})$ requires

certain conditions because u is uncertain to be a CCS point, so the particle set CCS (S) is not only possible to add points but also able to remove points when the policy is improved and reward vectors have a higher value.

- The selection of vectors from $\mathcal{W}_u$ is based on the percentage improvement value $\Delta_r(\mathrm{w})$, which becomes difficult because points belonging to S are not necessarily true CCS points and the algorithm needs to be explored, so vectors are eliminated corner weights can make policy training difficult.

The proposed method refines OLS into multi-object reinforcement learning. The title suggests the following changes:

- Removing the percentage improvement value term $\Delta_r(\phi)$ for selection corner weight vector in $\mathcal{W}_u$. We proposed to use all these weight vectors.
- After each $\mathcal{K} = \emptyset$ queue, performing the OLS algorithm again from step 2 (Determine simple vectors $\mathrm{w_e}$).
- Removing the "outdated" set of vectors $\mathcal{W}_{del}$, every corner weight vector is tested.
- $\mathtt{Potential\ Corner\ Weight}(\mathcal{W}_{\mathtt{del}}, \mathrm{S}, \mathrm{u_i})$ is designed with the following requirements: When the reward vector $\mathrm{u}_i = (u_{i1}, u_{i2})$ where u_{ik} is the k th target value of u_i. This vector will be compared with each $\mathrm{s} = (s^1, s^2) \in S$:
  - If all target values of u_i are equal better than s (denoted $\mathrm{u}_i \geq \mathrm{s}$) then eliminate s and if $\mathrm{u}_i \leq \mathrm{s}$ then ignore u_i. Execute a new loop.
  - Let $(\mathrm{w_{s,low}}, \mathrm{w_{s,high}})$ be the interval of the value of the second weight corresponding to the second object with the following property:
    $\forall\, \mathrm{w_r} \in [\mathrm{w_{s,low}}, \mathrm{w_{s,high}}], \mathrm{w}_r = (1 - \mathrm{w_r}, \mathrm{w_r}) \rightarrow \mathrm{w}_r^T \mathrm{s} \geq \mathrm{w}_r^T \mathrm{s}' \ \forall \mathrm{s}' \in S$
  - When u_i can intersect with vector s at point with coordinates $\in [0,1]$ then the intervals $(\mathrm{w_{s,low}}, \mathrm{w_{s,high}})$ of each s and u_i will be updated.
  - $S \leftarrow S \cup \{\mathrm{u}_i\}$

Based on the above sections, in the final part of the proposed algorithm. The report will summarize the steps of the proposed method in pseudocode snippet in Fig. 9 below.

The pseudocode for Algorithm 2 in Fig. 9 can be summarized by the following overview Fig.10.

The Fig. 10 above shows that in each training set, there are four blocks including: Determining the weight vector, training the Q network, and then using the result that has just been trained to estimate the reward vector corresponding to the weight vector, and finally creating the new weight vectors for the next training sets. The first and fourth block is directly related to the implementation of the OLS algorithm while two others are related to the deep Q network algorithm. The details of each block correspond to each part in the pseudocode code. The work block "Determine weight vector" has the role of deciding the type of target that the agent will perform in the training set. These weight vectors will be taken according to the queue $\mathcal{K}$. As explained in the above section on OLS, each application of OLS only yields S which is a subset of CCS and the duration for performing one OLS pass is very small compared to the number of steps that can be trained in RL. Therefore, every time the OLS algorithm stops or $\mathcal{K} = \emptyset$, the algorithm is re-executed with the simple addition of weight vectors to $\mathcal{K}$. However, the set S remains the same due to its role in supporting the finding of angle weight vectors, helping to avoid too many repetitions of testing on one vector in the number.

The second block of work plays a major role in policy training. Based on the results of proving the convergence theorem, we can see that using deep reinforcement learning algorithms will be approximately $\mathcal{CQ}$ in a complex space. The DQN algorithm is used and combined with the label update formula for training along with the combination of the HER algorithm and the design of a new loss function.

Next, the third main block of work has a similar role to the $VectorR\ eward(w_e)$ function in the pure OLS algorithm. However, because the environment in reinforcement learning is random and the algorithm used is model-free, after each training session, the reward vector after 100 steps will be considered with other reward vectors in the component set S. Finally, the fourth block of work is the OLS algorithm presented in the above section with the role of multiplying the new reward vector and then editing the set S, determining new corner weight vectors.

2.3.2 User clustering block

Our research focuses on the issue of model memory retention during episodic interactions. We specifically observe that the model's ability to recall information from earlier episodes, such as the 1000^{th} episode, is significantly reduced in later stages. This phenomena reflects the challenges faced in the transmission of information across a sequence of processing units in recurrent neural network (RNN) (Sherstinsky, 2020) architectures. The large amount of data encountered in successive user encounters poses considerable hurdles in terms of representation. The intricacy here is comparable to the process of enclosing a wide range of information within a single cell state in an RNN. This, in turn, makes it challenging to directly apply tactics commonly utilized in LSTM (Hochreiter and Schmidhuber, 1997) networks or transformer models.

Training a reinforcement learning model requires setting values for episodes and steps for each episode. If we associate each episode with a single user, and each step represents a user transaction, it can lead to issues when dealing with a large number of users. Training the model over many episodes may result in the model forgetting the preferences of users from the initial episodes (Yalnizyan-Carson and Richards, 2022). Therefore, employing clustering becomes crucial to identify characteristic users within a group, aiding the model in understanding the general preferences of customers. This approach reduces training time for the MORL framework when dealing with a significantly smaller number of virtual customers compared to the actual customer base.

Algorithm 2 Intergrating OLS to MORL algorithm

Input: N_e: number of training episodes, N_s: number of interactions per episode, N_B: batch size, tried weight set $\mathscr{W}$, queue of weight vectors $\mathscr{K}$.
Initialize θ_0, $\theta^{target} \leftarrow \theta_0$, replay buffer $B = \emptyset$, CCS component set $S = \emptyset$, $\mathscr{D}_\tau, \mathscr{D}_\phi$ is distribution to sample interaction experience in replay buffer and weight vector in HER method, respectively
Output: Trained Q network.

1: **for** n = 1 to N_e **do**
2: **if** $\mathscr{K} = \emptyset$ **then**
3: $\mathscr{K} = \mathscr{K} \cup \{\phi_e | \phi_e$ is one-hot vectors$\}$
4: **end if**
5: $\phi \leftarrow \mathscr{K}$.put()
6: $\varepsilon = 1$
7: **for** $t \in 1, 2, ..., N_s$ **do**
8: s_t: the current state at t-th step
9: Choose the next action a_t:

$$a_t = \begin{cases} \text{random action in } \mathscr{A}, & \text{w.p } \varepsilon \\ \max_{a \in \mathscr{A}} \phi^T \mathbf{Q}(s_t, a, \phi; \theta), & \text{w.p } 1 - \varepsilon \end{cases}$$

10: Response of environment: $\mathbf{r}_t$ - the immediate reward vector and s_{t+1} - the next state .
11: $\varepsilon = \varepsilon * \beta$
12: Update the replay buffer B $\leftarrow B \cup (s_t, s_{t+1}, a_t, \mathbf{r}_t)$
13: Sample N_B experience $(s_j, a_j, \mathbf{r}_j, s_{j+1}) \sim \mathscr{D}_\tau$
14: Sample N_ϕ weight vectors $W = \{\phi_i \sim \mathscr{D}_\phi\}$
15: Compute

$$y_{ij} = (\mathscr{C}\mathbf{Q})_{ij} = \begin{cases} \mathbf{r}_j, & \text{for terminal state } s_{j+1} \\ \mathbf{r}_j + \gamma \arg_Q \max_{a \in \mathscr{A}, \phi' \in W} \phi_i^\top \mathbf{Q}(s_{j+1}, a, \phi'; \theta), & \text{o.w.} \end{cases}$$

for all $1 \le i \le N_\phi$ and $1 \le j \le N_B$.
16: Update $\theta^{target} \leftarrow \theta$ by descending its stochastic gradient on loss function:

$$\nabla_\theta \mathscr{L}(\theta) = (1 - \lambda) \cdot \nabla_\theta \mathscr{L}^{\mathrm{A}}(\theta) + \lambda \cdot \nabla_\theta \mathscr{L}^{\mathrm{B}}(\theta)$$

17: Increase the value of λ
18: **end for**
19: $To_r = (0, 0)$
20: **for** t **do**=1 to 100
21: Observe the current state s_t
22: Select action $a_t = argmax_a \phi^T Q(s_t, a, \phi; \theta)$
23: Obtain $\mathbf{r}_t$ - reward vector and the next s_{t+1} - state
24: $To_r = To_r + \gamma * \mathbf{r}_t$
25: **end for**
26: $\mathbf{r}_{new} = To_r$
27: $\mathscr{W} =$ `PotentialCornerWeight`$(S, \mathbf{r}_{new})$
28: $\mathscr{K} = \mathscr{K} \cup \mathscr{W}$.
29: **end for**

Fig. 9. The pseudo-code of integrating OLS to MORL algorithm.

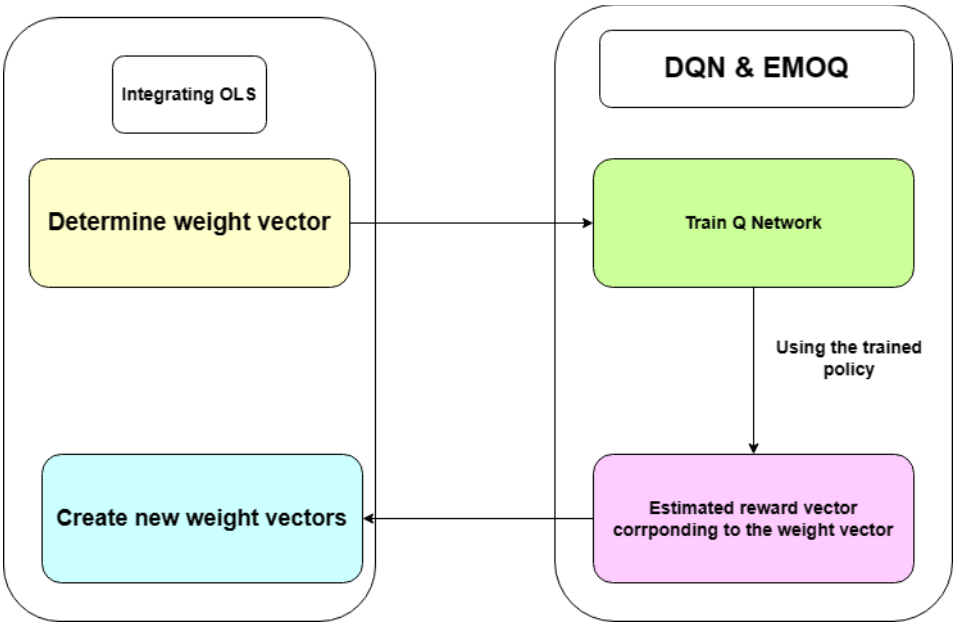


Fig. 10. Diagram overview of work blocks in a training set

In order to tackle this problem, our research presents a more sophisticated methodology. One component of this strategy entails limiting the number of episodes or decreasing the user sample size for model training. A more innovative approach entails the development of "virtual" user profiles. These profiles combine and express the collective features and preferences of a group of people. This approach provides a practical way to handle the challenges that arise from a large amount of information being exchanged, guaranteeing the preservation of important knowledge gained from the user community.

The following part is our design for user clustering vector. The user vector for the clustering problem will be different from the user vector for RL training in the above section because the user vector is temporal, changes continuously, and the task of the user clustering vector will be to generalize the characteristics of each user in the entire data. The user-clustering vector will be created similarly to the user transaction vector, which is a weighted sum of user vectors. Each vector has a main component that is the user vector within 10 months with a coefficient depending on the time, the farther the time, the smaller the contribution:

$$UC_u = \sum_{t=3}^{12} (1+b)^{t-2} U_{ut}$$

With UC_u being the user-clustering vector, U_{ut} is the user vector at month t in 2020, and the coefficient $b > 0$ to represent recent months is more meaningful in expressing user preferences.

This paragraph presents the clustering algorithm and metric: The clustering algorithm that the proposed method uses is Kmeans with the clustering metric being the cosine similarity between 2 user clustering vectors corresponding to 2 users. The number of clusters to be divided is examined through two evaluation metrics for clustering: Silhouette and Elbow. At each cluster, the method determines the user center - the user closest to the center of the cluster after implementing the Kmeans algorithm. Because training RL requires a certain number of training episodes and each episode needs to have a large enough number of steps to be able to explore and exploit effectively. To select the users closest to the user center and have a large enough number of interactions, the article proposes a way to calculate the similarity number interaction score:

$$SNI(u) = 0.5 \times cos\left(v_u, v_{u_{ccenter}}\right) + 0.5 \times \frac{NI_u}{\max_u NI_u}$$

where u is the user whose score needs to be calculated, u_{center} is the center corresponding to the cluster to which user u belongs, v_u is the corresponding user clustering vector, and NI_u is the number of interactions of user u. The coefficient 0.5 shows that the method considers two factors close to the central user and the number of interactions to be equally important.

3. RESULTS AND DISCUSSION

In this section 3, we will detail the installation, experiments, and results. Specifically, regarding the installation part, our proposed model includes the three most important components preprocessing data, reinforcement learning framework, and clustering part. Each component's setting is shown in a subsection: preprocessing settings in subsection 3.2; RL settings include everything related to applying the RL framework in section 3.3; and user clustering settings in section 3.5. In the RL setting section, there are 3 subparts: state, action setting in part 3.3.1, reward setting: in part 3.3.2, and model Q Network in part 3.3.3. The experiment part is shown in two subsections 3.6 and 3.7.

3.1 Dataset

The data set used in this paper is the interaction data set of users using a phone network in Vietnam from March 2020 to December 2020. The data set includes three files:

- User profile: The first file contains monthly data for a period of ten months, detailing records of 1000 users. Each record represents the consumption of each user in 1 month in 2020. This consumption information encompasses user IDs, provincial details, total expenditure, and the breakdown of expenses by primary purposes. Additionally, it covers the total amount of data consumed and short message service (SMS) usage.
- Package profile: The second file provides details on 275 different packages, each record is corresponding to profile of each package. The profile information of each package consists of various elements such as package ID, registration cost, the quantity of data included, type of package, expiration date, and other related information.
- Transaction: The final file documents over 170,000 transactions between 1000 users and the 275 mobile packages, each record is corresponding to one transaction. The detailed information of each transaction consists of the date of interaction, ItemID (representing the package), AccountID (identifying the user), and the price associated with the interaction.

The dataset is split into two subsets: a training set and a test set. The training set encompasses data from a 9-month period, spanning from March 2020 to November 2020. The test set, on the other hand, is focused on data from a specific date, December 1, 2020.

3.2 Normalizing Feature Vectors

Because of the diversity of dataset, we need to preprocess feature vectors before utilizing these vectors for training. There are two common types of features: non-seasonal features and seasonal features. Our preprocessing steps as follow:

- Non-seasonal features: $v^{new} = \frac{v - \min_v}{\max_v - \min_v}$, which is max min-normalization, v^{new} is the new normalized value of v.
- Seasonal features: $u^{new} = \cos\left(2\pi \times \frac{u}{S_j}\right)$, where S_j denotes the cycle of the feature.

3.3 Reinforcement Learning Setting

3.3.1 Settings with 4 vectors and preprocessing the features of the vectors

The following settings were obtained through multiple trials of various parameter sets, selecting the optimal parameter set. This approach is similar to that used in papers on DRL (Lillicrap et al., 2015; Zhao et al., 2017; Hu et al., 2018). The four vectors in the proposed method will be set up with the custom dimensions and meanings. The detailed meaning is as follows:

- Product vector: $I_i \in \mathbb{R}^6$ includes features such as price, supply data flow level, intra-network traffic, off-network traffic, allowed number of sms messages, time limit.
- User vector: $U_u \in \mathbb{R}^{19}$ includes features such as the amount of money used for calls, internet access, texting, numbers call, time.
- User-item vector: $UI_{ui} \in \mathbb{R}^2$ includes 2 information corresponding to a pair (u, i) is the number of interactions that user u has registered for package i and the rank of package i in all packages that the user has registered within the last 3 months.
- User transaction vector: $UT_u \in \mathbb{R}^{12}$ includes 2 time periods: 1 month ago and 3 months ago to describe user preferences according to characteristics of product.

Among the features in all 4 vectors, only the time features are cyclically normalized, while the remaining features are normalized so that the value is in the range $[0,1]$ as presented in the methodology section.

3.3.2 Reward settings

The first setting is for single objective case: With the reward formula defined as $r(u,i,t) = x \times (1-y)^a$, the values for $x = 1$ and $y = 0.02$ are set. Thus, the reward formula becomes $r(u,i,t) = 0.98^a$, where a represents the time distance from the recommendation timestamp t to the nearest date when the user interacted with item i in the interaction file.

Next, in the multi objective case: this paper proposed the additional object about the obtained profit which is defined as the ratio between the price of an item divided by the period of that item. However, to avoid the case that the recommender only focuses on the profit, which makes the recommended item does not fit to user's preference, we set the condition requiring a positive user response to that item, denoted as $r_1 > 0$:

$$\mathrm{r} = [r_1, r_2] = \left[x \times (1-y)^a, \frac{C_i}{T_i} \times \{r_1 > 0\}\right]$$

where $\frac{C_i}{T_i}$ is the ratio between the cost (price) of item $\boldsymbol{i}$ divided by the period (duration) of that item.

3.3.3 Q network settings

The Q network is designed with 2 hidden layers with the number of units 64 and 32 respectively. The activation functions are both ReLU.

3.4 Evaluation Metrics

We have a set of five recommend items $I' = (I_1, I_2, I_3, I_4, I_5)$ for user u at timestamp t, and the real items user u interacting at that time is I_{real}. In this paper, two common metrics to evaluating the quality of the recommendation system are hit rate which presents the ratio of the real item in the recommended list, and discounted cumulative gain (DCG) which presents the ranking of the real item in list (both the range of value of two metrics is $[0,1]$).

To evaluate all recommended item list to all user U, we denote the number of users to test is n, the rank score of the real item I_{real} in the list I' is l_k with k is the order of

recommendation for that user. The formula of two evaluated metrics as follow:

$$hit_k = \begin{cases} 1 & \text{if } i_{real} \in I' \\ 0 & \text{if } i_{real} \notin I' \end{cases} \Rightarrow hit@5 = \frac{1}{n}\Sigma_{k=1,n} hit_k$$

$$dcg_k = \begin{cases} \frac{1}{log_2(l_k + 2)} & \text{if } I_{real} \in I' \\ 0 & \text{if } I_{real} \notin I' \end{cases} \Rightarrow dcg@5 = \frac{1}{n}\Sigma_{k=1,n} d_k$$

3.5 User Clustering Setting

In the formula for computing user-clustering vector:

$$UC_u = \sum_{m=3}^{12} (1 + b)^{m-2} U_{um}$$

The value is set to $b = 0.1$ to show that the months closer to December have a greater influence. Then, based on the survey results of the number of clusters from 2 to 20 of the two metrics Silhouette and Elbow, the number of user clusters is 5.

3.6 Single Objective Results

To illustrate the effectiveness of modeling from the perspective of reinforcement learning in the recurring product recommendation problem, for simplicity, we will experiment with the case of a single target first. Here, we have a table of experiments like Table 1, where we compare the reward vector with our continuous value $r = x \times (1 - y)^a$ with discrete values from 0 to 1, this discrete reward formula is based on the reward setting of models DRR (Liu et al., 2018) and LIRD (Zhao et al., 2017) for single-use products such as movies, news or songs. Specifically, value $r = 0.5$ represents user u has ever interacted with item i but not at the time of suggestion while $r = 0$ means user u hasn't ever chosen that item and $r = 1$ means recommend right the real item to user at the timestamp t.

Besides, we also want to see the effectiveness of the user clustering block. Furthermore, to evaluate the impact of user clustering, a scenario utilizing only cosine similarity scores is tested, without considering the number of user interactions in the comparison. There are 5 main experiments conducted in the following Table 1.

Table 1. Table of experiment list

Experiment	Reward formula	User clustering
1	$\in \{0,0.5,1\}$	None
2	$\in \{0,0.5,1\}$	None but with embedding features
3	$x \times (1 - y)^a$	None
4	$x \times (1 - y)^a$	Only using cosine similarity scores
5	$x \times (1 - y)^a$	With user clustering - our proposed model

Comparison between experiments: The results from the five experiments mentioned above have demonstrated the effectiveness of the proposed model. The two Fig. 11 and Fig. 12 below show the results of the experiments with the hit and dcg metrics across each training episode. The yellow line corresponds to the model in experiment 1, red - experiment 2, purple - experiment 3, blue - experiment 4, and green - the proposed model.

From the results of both metrics for the five experiments, it can be observed that the values of hit@5 and dcg@5 exhibit minimal changes in the later episodes for all five experiments. The effectiveness of considering the interaction time with recommended items, as proposed in the "change_reward" model, is evident since even in the early training episodes, the hit@5 and dcg@5 values of the "change_reward" model (experiment 3 in Table 1) are consistently higher compared to the other two models (experiments 1 and 2), where the rewards are not time-differentiated. Specifically, the initialization values of the "change_reward" model (representing different reward formulas) start at around 0.85, while the other two models remain at 0.4 and below 0.2.

On the other hand, the effectiveness of user clustering is also demonstrated, as the proposed model achieves higher values after the initial training episodes compared to the "change_reward" model (without clustering) and significantly higher than the model in experiment 4 (which has clustering but score based on interactions is omitted), indicating improved exploration and exploitation due to the heavy influence.

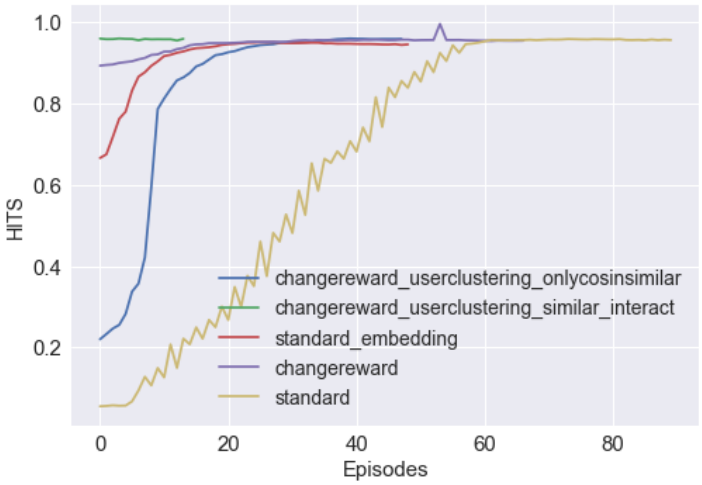


Fig. 11. Training results for each episode of the experiments with the HIT metric

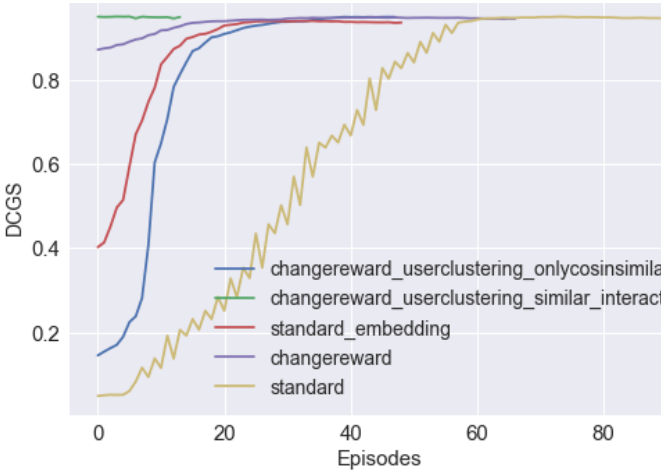


Fig. 12. Training results for each episode of the experiments with the DCG metric

Stability of user clustering: From Fig. 11 and Fig. 12, we can see that our proposed model starts better than other models, leading to faster stabilization. However, that is only the result of running 1 time, to evaluate more objectively, we test each experiment by running the first training episode (in total we run 30 times). To demonstrate the reliability of the proposed model, the paper showcases the following box plots of two metrics to evaluate the stability of our model (change_reward 2) in Fig. 13 and Fig. 14.

The box plots display a compilation of experiments along with their related names. Implementation details are provided in Table 2.

Table 2. Compilation of trials assessing the durability of the model

Experiment	Reward	User clustering
change_reward_2	$x \times (1-y)^a$	With score as in the proposed model
change_reward_1	$x \times (1-y)^a$	None
standard_2	$\in \{0,0.5,1\}$	With score as in the proposed model
standard_1	$\in \{0,0.5,1\}$	None

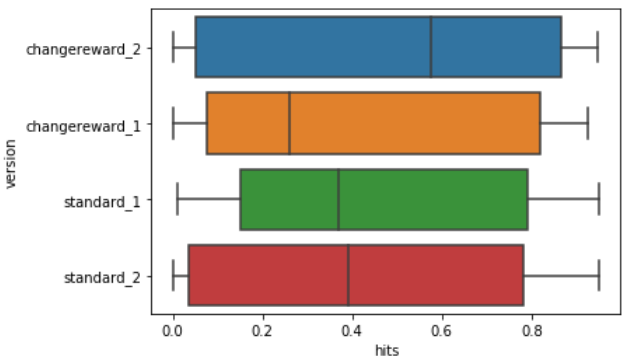


Fig. 13. The box plot corresponds to the HIT metrics

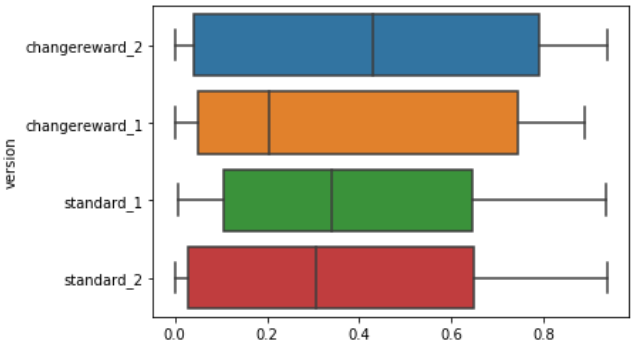


Fig. 14. The box plot corresponds to the DCG metrics

It is clear that both the median value and the max value of our proposed are higher than other experiments, which means our model "often" have the initial point better than other models.

Comparison with another methods: The proposed method was also tested with the DRR algorithm and the Table 3 is the result. Note that, to test with the DRR algorithm, we need to convert the interaction dataset to the rating dataset, so we can also see that which algorithm only receive the rating dataset to be input can not generalized as our proposed model, our model only can received interaction dataset. Furthermore, from the result in Table 3, we see that our model is better both on hit metric and dcg metric.

Table 3. Single-objective: Comparison with the DRR method

Method	***hit*@5**	***dcg*@5**
Proposed method	0.95	0.95
DRR algorithm	0.81	0.77

3.7 Multi-Objective Results

To optimize training efficiency, the user clustering results are applied, reducing the amount of training required. During this process, two metrics are plotted for each trained set to assess performance: hit@5 and dcg@5 (discounted cumulative gain at 5). The best-case scenario demonstrated a hit@5 of 0.93 and a dcg@5 of 0.71, indicating a high level of accuracy in the recommendations. This is depicted in a Fig. 15 illustrating the training process of the proposed algorithm across each training set.

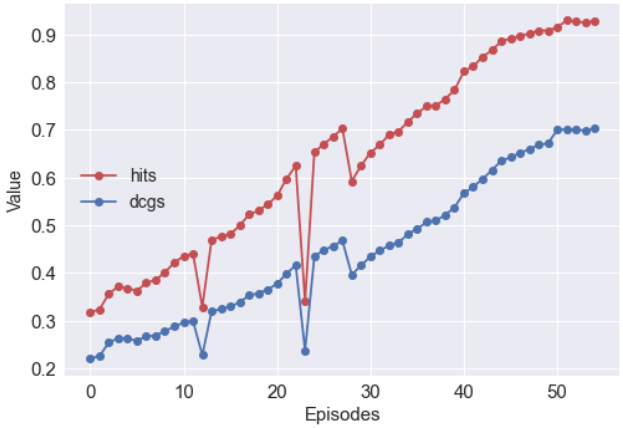


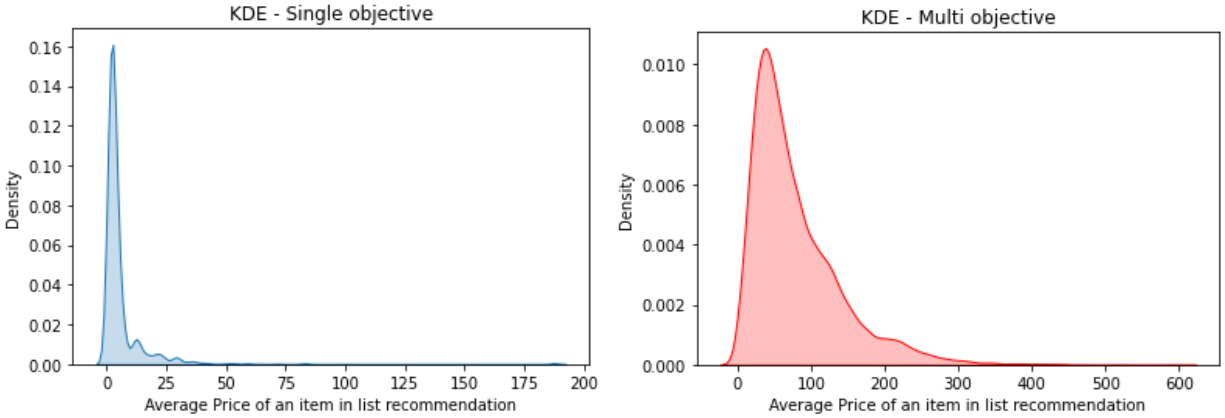
Fig. 15. Training results through each episode with multi-object modeling

We know that we can check that the first component of the reward vector corresponds to user preferences, that is, the relationship between that product and the user, while the second component is more difficult to evaluate when we do not know what users actually choose if we recommend a product list. Therefore, we evaluate based on the average value of a package in the recommendation list, that is, for each recommendation, we calculate the following value: (PPI - profit per item)

$$PPI = \frac{1}{5}\sum_{j=1}^{5}\left(\frac{C_{I_j}}{T_{I_j}} \times \left(sgn\left(r_1^{I_j}\right)\right)\right)$$

where $sgn\left(r_1^{I_j}\right) \in \{1,0\}$ is the sign of the first element in the reward vector.

Figs. 16(a) and 16(b) show Kernerl density estimation for the average price of each recommendation to users for the two cases single-object and multi-obj. The multi-objective framework generally leads to a higher average price of products than the single-objective method. With the single-objective model, the recommended products tend to be cheaper, mostly within the 0 to 25 price range, dominating the recommendation lists. In contrast, applying the multi-objective framework results in recommendations with notably higher average prices when mostly within from around 50 to 200.



(a) (b)

Fig. 16. Kernel density estimator of the average price for five recommended items: (a) Single-object (b) Multi-objective

Furthermore, both metrics, hit@5 and dcg@5, achieve notable levels, signifying that the actual items users are interested in frequently appear high in the recommended list. This indicates not just the relevance in the recommendations but also their potential for higher engagement and satisfaction from the users.

3.8 Discussion

According to our understanding, there is limited research applying MORL to recommender system (Paparella et al., 2023). In this study, we introduce a novel MORL framework into the RS, enhancing recommendation performance and simultaneously boosting company revenue compared to a focus solely on improving recommendation accuracy. While constructing MORS with various objective functions, a straightforward method is employing the Scalarization approach (Zheng and Wang, 2021). However, existing approaches often fix the weights for each objective function in advance (Stamenkovic et al., 2022), or they do not address rapid adaptation to continuous changes in task priorities in practical scenarios. To address this, we leverage the MORL framework proposed by Yang et al. (2019) for application in RS, enabling the system to dynamically adapt to continuous shifts in priority among objectives.

Additionally, to swiftly determine optimal preference weights, we incorporate the OLS algorithm, optimizing the training process in a shorter timeframe and expediting the identification of the CCS. Notably, in contrast to typical RL

approaches in RS, we do not train the agent for each individual customer to assist in recommending suitable products (Afsar et al., 2022). Instead, we employ a clustering methodology to identify representative customers. This approach facilitates rapid training, computational cost savings, and accelerates the convergence speed of the MORL algorithm in deriving optimal recommendations.

Our framework introduces a comprehensive approach to integrating MORL into RS, offering flexibility, efficiency, and enhanced adaptability to changing objectives, thereby contributing to the advancement of recommendation system research.

4. CONCLUSION

Our research presents details from modeling to integrating other algorithms to increase efficiency in training to building a multi-target personalized recommendation system for periodic products. Our modeling includes defining terms in reinforcement learning, especially defining vectors that represent user preferences and identifying optimization goals. The two goals that we identify for a recommendation system are recommended products that match user preferences and profit optimization (high value of the product). Our approach is to extract characteristic users from building a user clustering block and, after that, explain and integrate the OLS algorithm into envelope multi-object Q-learning. Our proposed method is tested on real datasets, the results show that our model is remarkable and has potential.

ACKNOWLEDGMENT

This work was supported by the Vietnam Ministry of Education and Training [grant number B2023-BKA-07]

REFERENCES

Afsar, M.M., Crump, T., Far, B., 2022. Reinforcement learning based recommender systems: A survey. ACM Computing Surveys, 55(7), 1–38.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W., 2017. Hindsight experience replay. Advances in Neural Information Processing Systems, 30.

Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A., 2017. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26–38.

Chen, L., Zhu, G., Liang, W., Wang, Y., 2023. Multi-objective reinforcement learning approach for trip recommendation. Expert Systems with Applications, 226, 120145.

Covington, P., Adams, J., Sargin, E., 2016. Deep neural networks for YouTube recommendations. In Proceedings of the 10th ACM conference on recommender systems, pp. 191–198.

Cui, L., Ou, P., Fu, X., Wen, Z., Lu, N., 2017. A novel multi-objective evolutionary algorithm for recommendation systems. Journal of Parallel and Distributed Computing, 103, 53–63.

Gomez-Uribe, C.A., Hunt, N., 2015. The Netflix recommender system: Algorithms, business value, and innovation. ACM Transactions on Management Information Systems, 6(4), 1–19.

Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Roijers, D.M., 2022. A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems, 36(1), 26.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation, 9(8), 1735–1780.

Hou, Y., Gu, W., Dong, W., Dang, L., 2023. A deep reinforcement learning real-time recommendation model based on long and short-term preference. International Journal of Computational Intelligence Systems, 16(1), 4.

Hu, Y., Da, Q., Zeng, A., Yu, Y., Xu, Y., 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 368–377.

Jahn, J., 1985. Scalarization in multi objective optimization. Springer Vienna, pp. 45–88.

Joachims, T., Freitag, D., Mitchell, T., 1997. Webwatcher: A tour guide for the world wide web. International Joint Conference on Artificial Intelligence, pp. 770–777.

Keat, E.Y., Sharef, N.M., Yaakob, R., Kasmiran, K.A., Marlisah, E., Mustapha, N., Zolkepli, M., 2022. Multiobjective deep reinforcement learning for recommendation systems. IEEE Access, 10, 65011–65027.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. International Conference on Learning Representations, abs/1509.02971.

Linden, G., Smith, B., York, J., 2003. Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet computing, 7(1), 76–80.

Liu, F., Tang, R., Li, X., Ye, Y., Chen, H., Guo, H., Zhang, Z., 2018. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. Neurocomputing, 307, 139–150.

Mulani, J., Heda, S., Tumdi, K., Patel, J., Chhinkaniwala, H., Patel, J., 2020. Deep reinforcement learning based personalized health recommendations. “Deep learning techniques for biomedical and health informatics”, Springer, pp. 231–255.

Paparella, V., Anelli, V.W., Boratto, L., Di Noia, T., 2023. Reproducibility of multiobjective reinforcement learning recommendation: Interplay between effectiveness and beyond-accuracy perspectives. In Proceedings of the 17th ACM Conference on Recommender Systems, pp. 467–478.

Roijers, D.M., Vamplew, P., Whiteson, S., Dazeley, R., 2013.

A survey of multi-objective sequential decision-making. Journal of Artificial Intelligence Research, 48, 67–113.

Roijers, D.M., Whiteson, S., Oliehoek, F.A., 2015. Computing convex coverage sets for faster multi-objective coordination. Journal of Artificial Intelligence Research, 52, 399–443.

Sarker, A., Shen, H., Kowsari, K., 2020. A data-driven reinforcement learning based multiobjective route recommendation system. IEEE 17th international conference on mobile ad hoc and sensor systems (mass), pp. 103–111.

Sherstinsky, A., 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.

Song, L., Tekin, C., Van Der Schaar, M., 2014. Online learning in large-scale contextual recommender systems. IEEE Transactions on Services Computing, 9(3), 433–445.

Stamenkovic, D., Karatzoglou, A., Arapakis, I., Xin, X., Katevas, K., 2022. Choosing the best of both worlds: Diverse and novel recommendations through multi-objective reinforcement learning. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, pp. 957–965.

Sutton, R.S., Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

Von Lucken, C., Baran, B., Brizuela, C., 2014. A survey on multi-objective evolutionary algorithms for many-objective problems. Computational optimization and applications, 58, 707–756.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H.V., Lanctot, M., De Freitas, N., 2016. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48, pp. 1995–2003.

Wu, H., Ma, C., Mitra, B., Diaz, F., Liu, X., 2022. A multi-objective optimization framework for multi-stakeholder fairness-aware recommendation. ACM Transactions on Information Systems, 41(2), 1–29.

Yalnizyan-Carson, A., Richards, B.A., 2022. Forgetting enhances episodic control with structured memories. Frontiers in Computational Neuroscience, 16, 757244.

Yang, R., Sun, X., Narasimhan, K., 2019. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 1311, 14636–14647.

Zhao, X., Zhang, L., Xia, L., Ding, Z., Yin, D., Tang, J., 2017. Deep reinforcement learning for list-wise recommendations. In Proceedings of the 12th ACM Conference on Recommender Systems. Association for Computing Machinery, New York, NY, USA, 95–103.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N.J., Xie, X., Li, Z., 2018. DRN: A deep reinforcement learning framework for news recommendation. In Proceedings of the 2018 world wide web conference, pp. 167–176.

Zheng, Y., Wang, D.X., 2021. A survey of recommender systems with multi-objective optimization. Neurocomputing, 474, 141–153.