Speeding up the Metabolism in E-commerce by Reinforcement Mechanism Design

Hua-Lin He
Alibaba Inc.
Hangzhou, China
hualin.hhl@alibaba-inc.com

Chun-Xiang Pan
Alibaba Inc.
Hangzhou, China
xuanran@taobao.com

Qing Da
Alibaba Inc.
Hangzhou, China
daqing.dq@alibaba-inc.com

An-Xiang Zeng
Alibaba Inc.
Hangzhou, China
renzhong@taobao.com

ABSTRACT

In a large E-commerce platform, all the participants compete for impressions under the allocation mechanism of the platform. Existing methods mainly focus on the short-term return based on the current observations instead of the long-term return. In this paper, we formally establish the lifecycle model for products, by defining the *introduction*, *growth*, *maturity* and *decline* stages and their transitions throughout the whole life period. Based on such model, we further propose a reinforcement learning based mechanism design framework for impression allocation, which incorporates the first principal component based permutation and the novel experiences generation method, to maximize short-term as well as long-term return of the platform. With the power of trial-and-error, it is possible to optimize impression allocation strategies globally which is contribute to the healthy development of participants and the platform itself. We evaluate our algorithm on a simulated environment built based on one of the largest E-commerce platforms, and a significant improvement has been achieved in comparison with the baseline solutions.

CCS CONCEPTS

• **Computing methodologies** → *Reinforcement learning*; *Policy iteration*; • **Applied computing** → *Online shopping*;

KEYWORDS

Reinforcement Learning, Mechanism Design, E-commerce

ACM Reference Format:

Hua-Lin He, Chun-Xiang Pan, Qing Da, and An-Xiang Zeng. 2018. Speeding up the Metabolism in E-commerce by Reinforcement Mechanism Design . In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR 2018 eCom)*. ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

Nowadays, E-commerce platform like Amazon or Taobao has developed into a large business ecosystem consisting of millions of customers, enterprises and start-ups, and hundreds of thousands of service providers, making it a new type of economic entity rather than enterprise platform. In such a economic entity, a major responsibility of the platform is to design economic institutions to achieve various business goals, which is the exact field of *Mechanism Design* [1]. Among all the affairs of the E-commerce platform, impression allocation is one of the key strategies to achieve its business goal, while products are players competing for the resources under the allocation mechanism of the platform, and the platform is the game designer aiming to design game whose outcome will be as the platform desires.

Existing work of impression allocation in literature are mainly motivated and modeled from a perspective view of supervised learning, roughly falling into the fields of information retrieval [2, 3] and recommendation [4, 5]. For these methods, a Click-Through-Rate (CTR) model is usually built based on either a ranking function or a collaborative filtering system, then impressions are allocated according to the CTR scores. However, these methods usually optimize the short-term clicks, by assuming that the properties of products is independent of the decisions of the platform, which may hardly hold in the real E-commerce environment. There are also a few work trying to apply the mechanism design to the allocation problem from an economic theory point of view such as [6–8]. Nevertheless, these methods only work in very limited cases, such as the participants play only once, and their properties is statistically known or does not change over time, etc., making them far from practical use in our scenario. A recent pioneer work named *Reinforcement Mechanism Design* [9] attempts to get rid of nonrealistic modeling assumptions of the classic economic theory and to make automated optimization possible, by incorporating the Reinforcement Learning (RL) techniques. It is a general framework which models the resource allocation problem over a sequence of rounds as a Markov decision process (MDP) [10], and solves the MDP with the state-of-the-art RL methods. However, by defining the impression allocation over products as the action, it can hardly scale with the number of products/sellers as shown in [11, 12]. Besides, it depends on an accurate behavioral model for the products/sellers, which is also unfeasible due to the uncertainty of the real world.

Although the properties of products can not be fully observed or accurately predicted, they do share a similar pattern in terms

SIGIR 2018 eCom, July 2018, Ann Arbor, Michigan, USA

of development trend, as summarized in the *product lifecycle theory* [13, 14]. The life story of most products is a history of their passing through certain recognizable stages including *introduction*, *growth*, *maturity* and *decline* stages.

- *Introduction*: Also known as *market development* - this is when a new product is first brought to market. Sales are low and creep along slowly.
- *Growth*: Demand begins to accelerate and the size of the total market expands rapidly.
- *Maturaty*: Demand levels off and grows.
- *Decline*: The product begins to lose consumer appeal and sales drift downward.

During the lifecycle, new products arrive continuously and outdated products wither away every day, leading to a natural metabolism in the E-commerce platform. Due to the insufficient statistics, new products usually attract few attention from conventional supervised learning methods, making the metabolism a very long period.

Inspired by the product lifecycle theory as well the reinforcement mechanism design framework, we consider to develop reinforcement mechanism design while taking advantage of the product lifecycle theory. The key insight is, with the power of trial-and-error, it is possible to recognize in advance the potentially hot products in the introduction stage as well as the potentially slow-selling products in the decline stage, so the metabolism can be speeded up and the long-term efficiency can be increased with an optimal impression allocation strategy.

We formally establish the lifecycle model and formulate the impression allocation problem by regarding the global status of products as the state and the parameter adjustment of a scoring function as the action. Besides, we develop a novel framework which incorporates a first principal component based algorithm and a repeated sampling based experiences generation method, as well as a shared convolutional neural network to further enhance the expressiveness and robustness. Moreover, we compare the feasibility and efficiency between baselines and the improved algorithms in a simulated environment built based on one of the largest E-commerce platforms.

The rest of the paper is organized as follows. The product lifecycle model and reinforcement learning algorithms are introduced in section 3. Then a reinforcement learning mechanism design framework is proposed in section 4. Further more, experimental results are analyzed in section 5. Finally, conclusions and future work are discussed in section 6.

2 RELATED WORK

Many researches have been conducted on impression allocation and dominated by supervised learning. In ranking phase, search engine aims to find out good candidates and brought them in front so that products with better performance will gain more impressions. Among which click-through rate is one of the most common representation of products performance. Some research presents an approach to automatically optimize the retrieval quality with well-founded retrieval functions under risk minimization frame-work by historical click-through data [15]. Some other research proposed an unbiased estimation of document relevance by estimating the presentation probability of each document [16]. Nevertheless, both of these research suffer from low accuracy of click-through rate estimation for the lack of exposure historical data of start-ups.

One of the most related topics in user impressions allocation is *item cold-start problem* [17], which has been extensively studied over past decades. Researches can be classified into three categories: hybrid algorithms combining CF with content-based techniques [18, 19], bandit algorithms [20–22] and data supplement algorithms [23]. Among these researches, the hybrid algorithms exploit items' properties, the bandit algorithms are designed for no item content setting and gathering interactions from user effectively, and the data supplement algorithms view cold-start as data missing problem. Both of these research did not take the whole product lifecycle of items into account for the weakness of traditional prediction based machine learning model, resulting in long-term imbalance between global efficiency and lifecycle optimization.

The application of reinforcement learning in commercial system such as web recommendations and e-commerce search engines has not yet been well developed. Some attempts are made to model the user impression allocation problem in e-commerce platform such as Tabao.com and Amazon.com. By regarding the platforms with millions of users as environment and treating the engines allocating user impressions as agents, an Markov Decision Process or at least Partially Observable Markov Decision Process can be established. For example, an reinforcement learning capable model is established on each page status by limit the page visit sequences to a constant number in a recommendation scene [24]. And another proposed model is established on global status by combining all the item historical representations in platform [11]. However, both of these approaches struggled to manage an fixed dimensionality of state observation, low-dimensional action outputs and suffered from partially observation issues.

Recently, mechanism design has been applied in impression allocation, providing a new approach for better allocating user impressions [9, 25]. However, the former researches are not suitable for real-world scenes because of the output action space is too large to be practical. In this paper, a reinforcement learning based mechanism design is established for the impression allocation problem to maximize both short-term as well as long-term return of products in the platform with a new approach to extract states from all products and to reduce action space into practical level.

3 PRELIMINARIES

3.1 Product Lifecycle Model

In this subsection, we establish a mathematical model of product lifecycle with noises. At step t, each product has an observable attribute vector $x_t \in \mathbb{R}^d$ and an unobservable latent lifecycle state $z_t \in \mathcal{L}$, where d is the dimension of the attribute space, and $\mathcal{L} = \{0, 1, 2, 3\}$ is the set of lifecycle stages indicating the the *introduction*, *growth*, *maturity* and *decline* stages respectively. Let $p_t \in \mathbb{R}$ be the CTR and $q_t \in \mathbb{R}$ be the accumulated user impressions of the product. Without loss of generality, we assume p_t and q_t are observable, p_t, q_t are two observable components of x_t, the platform allocates the impressions $u_t \in \mathbb{R}$ to the product. The dynamics of the system can be written as

$$\begin{cases} q_{t+1} = q_t + u_t \\ p_{t+1} = p_t + f(z_t, q_t) \\ z_{t+1} = g(x_t, z_t, t) \end{cases} \tag{1}$$

where f can be seen as the derivative of the p, and g is the state transition function over $\mathcal{L}$.

According to the product lifecycle theory and online statistics, the derivative of the CTR can be formulated as

$$f(z_t, q_t) = \begin{cases} \dfrac{(c_h - c_l)e^{-\delta(q_t)}}{(2-z)(1+e^{-\delta(q_t)})^2} + \xi, & z \in \{1,3\} \\ \xi, & z \in \{0,2\} \end{cases} \tag{2}$$

where $\xi \sim \mathcal{N}(0, \sigma^2)$ is a gaussian noise with zero mean and variance σ^2, $\delta(q_t) = (q_t - \tilde{q}_{t_z} - \delta_\mu)/\delta_\sigma$ is the normalized impressions accumulated from stage z , $\tilde{q}_{t_z}$ is the initial impressions when the product is firstly evolved to the life stage z, $\delta_\mu, \delta_\sigma$ are two unobservable parameters for normalization, and $c_h, c_l \in \mathbb{R}$ are the highest CTR and the lowest CTR during whole product lifecycle, inferred from two neural networks, respectively:

$$c_l = h(x_t|\theta_l), \quad c_h = h(x_t|\theta_h), \tag{3}$$

where $h(\cdot|\theta)$ is a neural network with the fixed parameter θ, indicating that c_l, c_h are unobservable but relevant to attribute vector x_t. Intuitively, when the product stays in introduction or maturity stage, the CTR can be only influenced by the noise. When the product in the growth stage, f will be a positive increment, making the CTR increased up to the upper bound c_h. Similar analysis can be obtained for the product in the decline stage.

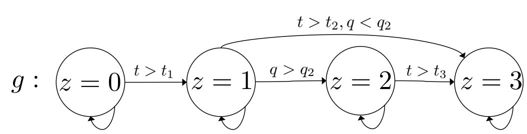

Figure 1: State transition during product lifecycle

Then we define the state transition function of product lifecycle as a finite state machine as illustrated in Fig. 1. The product starts with the initial stage $z = 0$, and enters the growth stage when the time exceeds t_1. During the growth stage, a product can either step in to the maturity stage if its accumulated impressions q reaches q_2, or the decline stage if the time exceeds t_2 while q is less than q_2. A product in the maturity stage will finally enter the last decline stage if the time exceeds t_3. Otherwise, the product will stay in current stage. Here, t_1, t_2, t_3, q_2 are the latent thresholds of products.

We simulate several product during the whole lifecycle with different latent parameters (the details can be found in the experimental settings), the CTR curves follow the exact trend described in Fig. 2.

3.2 Reinforcement Learning and DDPG methods

Reinforcement learning maximizes accumulated rewards by trial-and-error approach in a sequential decision problem. The sequential decision problem is formulated by MDP as a tuple of state

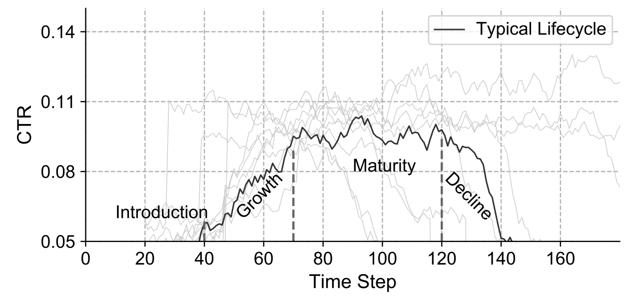

Figure 2: CTR evolution with the proposed lifecycle model.

space $\mathcal{S}$, action space $\mathcal{A}$, a conditional probability distribution $p(\cdot)$ and a scalar reward function $r = R(s, a), R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. For states $s_t, s_{t+1} \in \mathcal{S}$ and action $a_t \in \mathcal{A}$, distribution function $p(s_{t+1}|s_t, a_t)$ denotes the transition probability from state s_t to s_{t+1} when action a_t is adopted in time step t, and the Markov property $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_1, a_1, \cdots, s_t, a_t)$ holds for any historical trajectories $s_1, a_1, \cdots, s_t$ to arrive at status s_t. A future discounted return at time step t is defined as $R_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} R(s_k, a_k)$, where γ is a scalar factor representing the discount. A policy is denoted as $\pi_\theta(a_t|s_t)$ which is a probability distribution mapping from $\mathcal{S}$ to $\mathcal{A}$, where different policies are distinguished by parameter θ.

The target of agent in reinforcement learning is to maximize the expected discounted return, and the performance objective can be denoted as

$$\begin{aligned} \max_\pi J &= \mathbb{E}\left[R_1^\gamma \middle| \pi\right] \\ &= \mathbb{E}_{s \sim d^\pi, a \sim \pi_\theta}\left[R(s, a)\right] \end{aligned} \tag{4}$$

where $d^\pi(s)$ is a discounted state distribution indicating the possibility to encounter a state s under the policy of π. An action-value function is then obtained iteratively as

$$Q(s_t, a_t) = \mathbb{E}\left[R(s_t, a_t) + \gamma \mathbb{E}_{a \sim \pi_\theta}\left[Q(s_{t+1}, a_{t+1})\right]\right] \tag{5}$$

In order to avoid calculating the gradients of the changing state distribution in continuous action space, the Deterministic Policy Gradient(DPG) method [26, 27] and the Deep Deterministic Policy Gradient [28] are brought forward. Gradients of the deterministic policy π is

$$\begin{aligned} \nabla_{\theta^\mu} J &= \mathbb{E}_{s \sim d^\mu}\left[\nabla_{\theta^\mu} Q^w(s, a)\right] \\ &= \mathbb{E}_{s \sim d^\mu}\left[\nabla_{\theta^\mu} \mu(s) \nabla_a Q^w(s, a)|_{a=\mu(s)}\right] \end{aligned} \tag{6}$$

where μ is the deep actor network to approximate policy function. And the parameters of actor network can be updated as

$$\theta^\mu \leftarrow \theta^\mu + \alpha \mathbb{E}\left[\nabla_{\theta^\mu} \mu(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu(s)}\right] \tag{7}$$

where Q^w is an obtained approximation of action-value function called critic network. Its parameter vector w is updated according to objective

$$\min_w L = \mathbb{E}_{s \sim d^\mu}\left[y_t - Q^w(s_t, a_t))^2\right] \tag{8}$$

where $y_t = R(s_t, a_t) + \gamma Q^{w'}(s_{t+1}, \mu'(s_{t+1}))$, μ' is the target actor network to approximate policy π, $Q^{w'}$ is the target critic network to approximate action-value function. The parameters $w', \theta^{\mu'}$ are updated softly as

$$\begin{aligned} w' &\leftarrow \tau w' + (1-\tau) w \\ \theta^{\mu'} &\leftarrow \tau \theta^{\mu'} + (1-\tau)\theta^{\mu} \end{aligned} \tag{9}$$

4 A SCALABLE REINFORCEMENT MECHANISM DESIGN FRAMEWORK

In our scenario, at each step, the platform observes the global information of all the products, and then allocates impressions according to the observation and some certain strategy, after which the products get their impressions and update itself with the attributes as well as the lifecycle stages. Then the platform is able to get a feedback to judge how good its action is, and adjust its strategy based on all the feedbacks. The above procedures leads to a standard sequential decision making problem.

However, application of reinforcement learning to this problem encounters sever computational issues, due to high dimensionality of both action space and state space, especially with a large n. Thus, we model the impression allocation problem as a standard reinforcement learning problem formally, by regarding the global information of the platform as the state

$$s = [x_1, x_2, ..., x_n]^{\mathrm{T}} \in \mathbb{R}^{n \times d} \tag{10}$$

where n is the number of the product in the platform, d is the dimension of the attribute space, and regarding the parameter adjustion of a score function as the action,

$$a = \pi(s|\theta^{\mu}) \in \mathbb{R}^{d} \tag{11}$$

where π is the policy to learn parameterize by θ^{μ}, and the action a can be further used to calculate scores of all products

$$o_i = \frac{1}{1 + e^{-a^{\mathrm{T}} x_i}}, \quad \forall i \in \{1, 2, ..., n\} \tag{12}$$

After which the result of impression allocation over all n products can be obtained by

$$u_i = \frac{e^{o_i}}{\sum_i^n e^{o_i}}, \quad \forall i \in \{1, 2, ..., n\} \tag{13}$$

Without loss of generosity, we assume at each step the summation of impressions allocated is 1, i.e., $\sum_i^n u_i = 1$. As is well known, products number n(billions) is far bigger than products attributes dimensions d(thousands) in large scale E-commerce platforms. By such definition, the dimension of the action space is reduced to d, significantly alleviating the computational issue in previous work [12], where the the dimension of the action space is n.

The goal of policy is to speeded up the metabolism by scoring and ranking products under the consideration of product lifecycle, making the new products grow into maturity stage as quickly as possible and keeping the the global efficiency from dropping down during a long term period. Thus, we define the reward related to s and a as

$$R(s, a) = \frac{1}{n} \sum_i^n \left[\frac{1}{t_i} \int_{t=0}^{t_i} p(t) \frac{dq(t)}{dt} dt \right] \tag{14}$$

where t_i is the time step of the i-th product after being brought to the platform, $p(t), q(t)$ is the click through rate function and accumulated impressions of a product respectively. The physical meaning of this formulation is the mathematical expect over all products in platform for the average click amount of an product during its lifecycle, indicating the efficiency of products in the platform and it can be calculated accumulatively in the online environment, which can be approximately obtained by

$$R(s, a) \approx \frac{1}{n} \sum_i^n \frac{1}{t_i} \sum_{\tau=0}^{t_i} p_\tau^i u_\tau^i \tag{15}$$

A major issue in the above model is that, in practices there will be millions or even billions of products, making combinations of all attribute vectors to form a complete system state with size $n \times d$ computationally unaffordable as referred in essays [11]. A straightforward solution is to applying feature engineering technique to generate a low dimension representation of the state as $s_l = \mathcal{G}(s)$, where $\mathcal{G}$ is a pre-designed aggregator function to generate a low dimensional representation of the status. However, the pre-designed aggregator function is a completely subjective and highly depends on the the hand-craft features. Alternatively, we attempt to tackle this problem using a simple sampling based method. Specifically, the state is approximated by n_s products uniformly sampled from all products

$$\hat{s} = [x_1, x_2, \cdots, x_{n_s}]^{\mathrm{T}} \in \mathbb{R}^{n_s \times d} \tag{16}$$

where $\hat{s}$ is the approximated state. Then, two issues arise with such sampling method:

- In which order should the sampled n_s products permutated in $\hat{s}$, to implement the *permutation invariance*?
- How to reduce the bias brought by the sampling procedure, especially when n_s is much smaller than n?

To solve these two problem, we further propose the first principal component based permutation and the repeated sampling based experiences generation, which are described in the following subsections in details.

4.1 First Principal Component based Permutation

The order of each sampled product in the state vector has to be proper arranged, since the unsorted state matrix vibrates severely during training process, making the parameters in network hard to converge. To avoid it, a simple way for permutation is to make order according to a single dimension, such as the brought time t_i, or the accumulated impressions q_i. However, such ad-hoc method may lose information due to the lack of general principles. For example, if we sort according to a feature that is almost the same among all products, state matrix will keep vibrating severely between observations. A suitable solution is to sort the products in an order that keep most information of all features, where the first principal components are introduced [29]. We design a first principal component based permutation algorithm, to project each x_i into a scalar v_i and sort all the products according to v_i

$$e_t = \arg\max_{\|e\|=1} \left(e^{\mathrm{T}} s_t{}^{\mathrm{T}} s_t e \right) \tag{17}$$

$$\hat{e} = \frac{\beta \hat{e} + (1-\beta)(e_t - \hat{e})}{\|\beta \hat{e} + (1-\beta)(e_t - \hat{e})\|} \tag{18}$$

$$v_i = \hat{e}^{\mathrm{T}} x_i, i = 1, 2, \cdots, n_s \tag{19}$$

where e_t is the first principal component of system states in current step t obtained by the classic PCA method as in Eq. 17. $\hat{e}$ is the projection vector softly updated by e_t in Eq. 18, with which we calculate the projected score of each products in Eq. 19. Here $0 < \beta < 1$ is a scalar indicating the decay rate of $\hat{e}$. Finally, the state vector is denoted as

$$\hat{s} = [x_{k_1}, x_{k_2}, \cdots, x_{k_{n_s}}]^{\mathrm{T}} \tag{20}$$

where $k_1, k_2, \cdots, k_{n_s}$ is the order of products, sorted by v_i.

4.2 Repeated Sampling based Experiences Generation

We adopt the classic experience replay technique [30, 31] to enrich experiences during the training phase just as other reinforcement learning applications. In the traditional experience replay technique, the experience is formulated as (s_t, a_t, r_t, s_{t+1}). However, as what we describe above, there are $C_n^{n_s}$ observations each step theoretically, since we need to sample n_s products from all the n products to approximate the global statistics. If n_s is much smaller than n, such approximation will be inaccurate.

To reduce the above bias, we propose the repeated sampling based experiences generation. For each original experience, we do repeated sampling s_t and s_{t+1} for m times, to obtain m^2 experiences of

$$(\hat{s}_t^i, a_t, r_t, \hat{s}_{t+1}^j), \quad i, j \in 1, 2, \cdots, m \tag{21}$$

as illustrated in Fig. 3. This approach improves the stability of observation in noise environment. It is also helpful to generate plenty of experiences in the situation that millions of times repetition is unavailable.

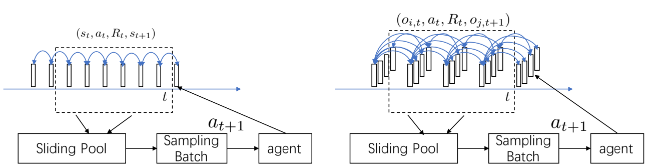

Figure 3: Classical experiences generation(left): One experience is obtained each step by pair(s_t, a_t, r_t, s_{t+1}); Repeated sampling based experiences generation(right): m^2 experiences are obtained each step by pair$(\hat{s}_t^i, a_t, r_t, \hat{s}_{t+1}^j)$

It is worth noting that, the repeated sampling is conducted in the training phase. When to play in the environment, the action a_t is obtained through a randomly selected approximated state $\hat{s}_t$, i.e., $a_t = \pi(\hat{s}_t^1)$. Actually, since a_t does not necessarily equal to $\pi(\hat{s}_t^i), \forall i \in 1, 2, \cdots, m$, it can further help learning a invariant presentation of the approximated state observations.

The overall procedure of the algorithm is described in Algorithm 1. Firstly, a random sampling is utilized to get a sample of system states. And then the sample is permutated by the projection of the first principal components. After that, a one step action and multiple observations are introduced to enrich experiences in experience pool. Moreover, a shared convolutional neural network is applied within the actor-critic networks and target actor-critic networks to extract features from the ordered state observation [32, 33].

Algorithm 1: The Scalable Reinforcement Mechanism Design Framework

Initialize the parameters of the actor-critic network $\theta^\mu, w, \theta^{\mu'}, w'$
Initialize the replay buffer M
Initialize m observations $\hat{s}_0^j$
Initialize the first principal component $\hat{p}$ by $\hat{s}_0$
foreach *training step* t **do**
  Select action $a_t = \mu(\hat{s}_t^1|\theta^\mu)$
  Execute action a_t and observe reward r_t
  foreach $j \in 1, 2, \cdots, m$ **do**
    Sample a random subset of n_s products
    Combine an observation in the order of $x_k^{\mathrm{T}}\hat{e}$
$$\hat{s}_t^j \leftarrow \left(x_{k_1}, x_{k_2}, \cdots, x_{k_{n_s}}\right)^{\mathrm{T}}$$
    Update first principal component
$$e_t \leftarrow \arg\max_{\|e\|=1} \left(e^{\mathrm{T}} \hat{s}_t^{j\mathrm{T}} \hat{s}_t^j e\right)$$
$$\hat{e} \leftarrow \mathrm{norm}\left(\beta\hat{e} + (1-\beta)(e_t - \hat{e})\right)$$
  end
  foreach $i, j \in 1, 2, \cdots, m$ **do**
$$M \leftarrow M \cup \{(\hat{s}_t^i, a_t, r_t, \hat{s}_{t+1}^j)\}$$
  end
  Sample n_k transitions from M: $(\hat{s}_k, a_k, r_k, \hat{s}_{k+1})$
  Update critic and actor networks
$$w \leftarrow w + \frac{\alpha_w}{n_k} \sum_k (y_k - Q^w(\hat{s}_k, a_k)) \nabla_w Q^w(\hat{s}_k, a_k)$$
$$\theta^\mu \leftarrow \theta^\mu + \frac{\alpha_\mu}{n_k} \sum_k \nabla_{\theta^\mu} \mu(\hat{s}_k) \nabla_{a_k} Q^w(\hat{s}_k, a_k)$$
  Update the target networks
$$w' \leftarrow \tau w' + (1-\tau) w$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu'} + (1-\tau)\theta^\mu$$
end

Finally, the agent observes system repeatedly and train the actor-critic network to learn an optimized policy gradually.

5 EXPERIMENTAL RESULTS

To demonstrate how the proposed approach can help improve the long-term efficiency by speeding up the metabolism, we apply the proposed reinforcement learning based mechanism design, as well as other comparison methods, to a simulated E-commerce platform built based on the proposed product lifecycle model.

5.1 The Configuration

The simulation is built up based on product lifecycle model proposed in section 3.1. Among all of the parameters, q_2 is uniformly sampled from $[10^4, 10^6]$, $t_1, t_2, t_3, \delta_\mu, \delta_\sigma$ are uniformly sampled from $[5, 30], [35, 120], [60, 180], [10^4, 10^6], [2.5 \times 10^3, 2.5 \times 10^5]$ respectively, and parameter σ is set as 0.016 . The parameters c_l, c_h are generated by a fixed neural network whose parameter is uniformly sampled from $[-0.5, 0.5]$ to model online environments, with the outputs scaled into the intervals of $[0.01, 0.05]$ and $[0.1, 0.15]$

Table 1: Parameters in learning phase

Param	Value	Reference
n_s	10^3	Number of products in each sample
β	0.999	First principal component decay rate
γ	0.99	Rewards discount factor
τ	0.99	Target network decay rate
m	5	Repeated observation times

respectively. Apart from the normalized dynamic CTR p and the accumulated impressions q, the attribute vector x is uniformly sampled from $[0, 1]$ element-wisely with the dimension $d = 15$. All the latent parameters in the lifecycle model are assumed unobservable during the learning phase.

The DDPG algorithm is adopted as the learning algorithm. The learning rates for the actor network and the critic network are 10^{-4} and 10^{-3} respectively, with the optimizer ADAM [34]. The replay buffer is limit by 2.5×10^4. The most relevant parameters evolved in the learning procedure are set as table 1.

Comparisons are made within the proposed reinforcement learning based methods as

- **CTR-A**: The impressions are allocated in proportion to the CTR score.
- **T-Perm**: The basic DDPG algorithm, with brought time based permutation and a fully connected network to process the state
- **FPC**: The basic DDPG algorithm, with first principal component based permutation and a fully connected network to process the state.
- **FPC-CNN**: FPC with a shared two-layers convolutional neural network in actor-critic networks.
- **FPC-CNN-EXP**: FPC-CNN with the improved experiences generation method.

where CTR-A is the classic supervised learning method and the others are the proposed methods in this paper. For all the experiments, CTR-A is firstly applied for the first 360 steps to initialize system into a stable status, i.e., the distribution over different lifecycle stages are stable, then other methods are engaged to run for another 2k steps and the actor-critic networks are trained for 12.8k times.

5.2 The Results

We firstly show the discounted accumulated rewards of different methods at every step in Fig. 4. After the initialization with the CTR-A, we find that the discounted accumulated reward of CTR-A itself almost converges to almost 100 after 360 steps (actually that why 360 steps is selected for the initialization), while that of other methods can further increase with more learning steps. It is showed that all FPC based algorithms beat the T-Perm algorithm, indicating that the FPC based algorithm can find a more proper permutation to arrange items while the brought time based permutation leads to a loss of information, making a drop of the final accumulated rewards. Moreover, CNN and EXP algorithms perform better in extracting feature from observations automatically, causing a slightly

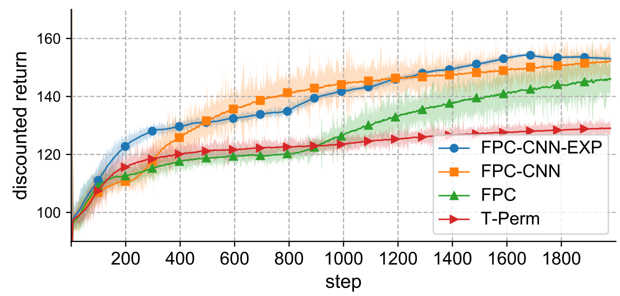

Figure 4: Performance Comparison between algorithms

improvement in speeding up the converging process. Both the three FCP based algorithms converge to same final accumulated rewards for their state inputs have the same observation representation.

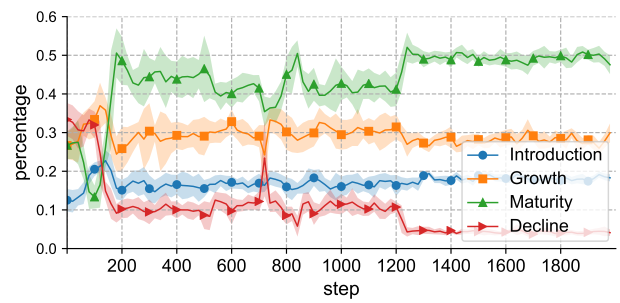

Figure 5: Percentage of impressions allocated to different stages.

Then we investigate the distribution shift of the impression allocation over the 4 lifecycle stages after the training procedure of the FPC-CNN-EXP method, as shown in Fig. 5. It can be seen that the percentage of decline stage is decreased and percentage of introduction and maturity stages are increased. By giving up the products in the decline stage, it helps the platform to avoid the waste of the impressions since these products are always with a low CTR. By encouraging the products in the introduction stage, it gives the changes of exploring more potential hot products. By supporting the products in the maturity stage, it maximizes the short-term efficiency since the they are with the almost highest CTRs during their lifecycle.

We finally demonstrate the change of the global clicks, rewards as well as the averaged time durations for a product to grow up into maturity stage from its brought time at each step, in terms of relative change rate compared with the CTR-A method, as is shown in Fig. 6. The global average click increases by 6% when the rewards is improved by 30%. The gap here is probably caused by the inconsistency of the reward definition and the global average click metric. In fact, the designed reward contains some other implicit objectives related to the metabolism. To further verify the guess, we show that the average time for items to growth into maturity stage has dropped by 26%, indicating that the metabolism is significantly speeded up. Thus, we empirically prove that, through the proposed reinforcement learning based mechanism design which utilizes the lifecycle theory, the long-term efficiency can be increased by speeding up the metabolism.

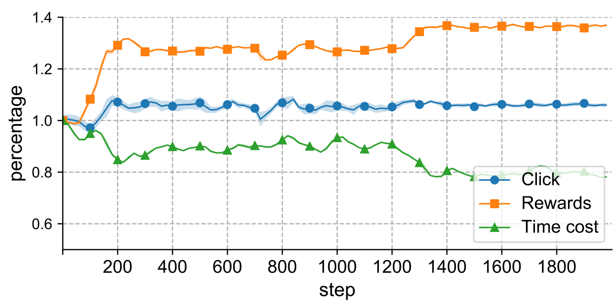

Figure 6: Metabolism relative metrics

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose an end-to-end general reinforcement learning framework to improve the long-term efficiency by speeding up the metabolism. We reduce action space into a reasonable level and then propose a first principal component based permutation for better observation of environment state. After that, an improved experiences generation technique is engaged to enrich experience pool. Moreover, the actor-critic network is improved by a shared convolutional network for better state representation. Experiment results show that our algorithms outperform the baseline algorithms.

For the future work, one of the promising directions is to develop a theoretical guarantee for first principal component based permutation. Another possible improvement is to introduce the nonlinearity to the scoring function for products.

REFERENCES

[1] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.
[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.
[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.
[4] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.
[5] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015.
[6] Roger B Myerson. Optimal auction design. *Mathematics of operations research*, 6(1):58–73, 1981.
[7] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.
[8] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
[9] Pingzhong Tang. Reinforcement mechanism design. In *Early Carrer Highlights at Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI, pages 5146–5150*, 2017.
[10] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.
[11] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. Reinforcement mechanism design for fraudulent behaviour in e-commerce. 2018.
[12] Qingpeng Cai, Aris Filos-Ratsikas, Pingzhong Tang, and Yiwei Zhang. Reinforcement mechanism design for e-commerce. *CoRR*, abs/1708.07607, 2017.
[13] Theodore Levitt. Exploit the product life cycle. *Harvard business review*, 43:81–94, 1965.
[14] Hui Cao and Paul Folan. Product life cycle: the evolution of a paradigm and literature review from 1950–2009. *Production Planning & Control*, 23(8):641–662, 2012.
[15] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.
[16] Georges E Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2008.
[17] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.
[18] Seung-Taek Park and Wei Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 21–28. ACM, 2009.
[19] Martin Saveski and Amin Mantrach. Item cold-start recommendations: learning local collective embeddings. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 89–96. ACM, 2014.
[20] Jin-Hu Liu, Tao Zhou, Zi-Ke Zhang, Zimo Yang, Chuang Liu, and Wei-Min Li. Promoting cold-start items in recommender systems. *PloS one*, 9(12):e113457, 2014.
[21] Oren Anava, Shahar Golan, Nadav Golbandi, Zohar Karnin, Ronny Lempel, Oleg Rokhlenko, and Oren Somekh. Budget-constrained item cold-start handling in collaborative filtering recommenders via optimal design. In *Proceedings of the 24th International Conference on World Wide Web*, pages 45–54. International World Wide Web Conferences Steering Committee, 2015.
[22] Michal Aharon, Oren Anava, Noa Avigdor-Elgrabli, Dana Drachsler-Cohen, Shahar Golan, and Oren Somekh. Excuseme: Asking users to help in item cold-start recommendations. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 83–90. ACM, 2015.
[23] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. In *Advances in Neural Information Processing Systems*, pages 4964–4973, 2017.
[24] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 113–120. ACM, 2007.
[25] Qingpeng Cai, Aris Filos-Ratsikas, Chang Liu, and Pingzhong Tang. Mechanism design for personalized recommender systems. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 159–166. ACM, 2016.
[26] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.
[27] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.
[28] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
[29] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.
[30] Long Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.
[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
[32] Yu-Hu Cheng, Jian-Qiang Yi, and Dong-Bin Zhao. Application of actor-critic learning to adaptive state space construction. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 5, pages 2985–2990. IEEE, 2004.
[33] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.
[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.