Cooperative Multi-Agent Deep Reinforcement Learning in Content Ranking Optimization

Zhou Qin[1,*,†], Kai Yuan[1,†], Pratik Lahiri[1] and Wenyang Liu[1]

[1]*Amazon.com, Inc. 550 Terry Ave N, Seattle, Washington 98109*

Abstract

In a typical e-commerce setting, Content Ranking Optimization (CRO) mechanisms are employed to surface content on the search page to fulfill customers' shopping missions. CRO commonly utilizes models such as contextual deep bandits model to independently rank content at different positions, e.g., one optimizer dedicated to organic search results and another to sponsored results. However, this regional optimization approach does not necessarily translate to whole page optimization, e.g., maximizing revenue at the top of the page may inadvertently diminish the revenue of lower positions. In this paper, we propose a reinforcement learning based method for whole page ranking to jointly optimize across all positions by: 1) shifting from position level optimization to whole page level optimization to achieve an overall optimized ranking; 2) applying reinforcement learning to optimize for the cumulative rewards instead of the instant reward. We formulate page level CRO as a cooperative Multi-agent Markov Decision Process , and address it with the novel **M**ulti-**A**gent **D**eep **D**eterministic **P**olicy Gradient (MADDPG) model. MADDPG supports a flexible and scalable joint optimization framework by adopting a "centralized training and decentralized execution" approach. Extensive experiments demonstrate that MADDPG scales to a 2.5 billion action space in the public Mujoco environment, and outperforms the deep bandits modeling by 25.7% on the offline CRO data set from a leading e-commerce company. We foresee that this novel multi-agent optimization is applicable to similar joint optimization problems in the field of information retrieval.

Keywords

Recommender Systems, Deep Reinforcement Learning, Actor-Critic

1. Introduction

Traditional e-commerce search results page can be consisted of several positions (a.k.a "slots"), e.g., shown as the blue boxes in Figure 1. Such an explicit design helps maintain a consistent shopping experience for the customers and is easy to maintain. Each position accommodates several rankable pieces of contents. Different positions could also serve different purposes: top-position content (e.g., position 1) includes highly relevant contents or a dedicated module answering or clarifying customers' questions; middle positions can help customers navigate, narrow down, or discover related products and categories; bottom positions may have lower relevance, but aims to inspire customers to either diverge from the

eCom'24: ACM SIGIR Workshop on eCommerce, July 18, 2024, Washington, DC, USA

*Corresponding author.

†These authors contributed equally.

qinzhouhit@hotmail.com (Z. Qin)

https://zhouqin.info (Z. Qin)

0000-0002-1641-772X (Z. Qin)

original intent or explore their search further. Content providers can schedule their contents to show at particular positions, based on different content properties. When multiple pieces of contents compete for the same position, a content optimizer (a.k.a ranker) is often designed to determine which are the most helpful pieces of contents to surface.

In common use cases, content ranking optimization (CRO) can be formalized as a ranking problem, such as building contextual deep learning models to optimize for certain objective functions, e.g., revenue, profit, or click-through rate (CTR), etc. [1] Or more often, a multi-objective optimization problem, i.e., optimizing several metrics simultaneously [2]. In real-world use cases, positions on the search results page are subject to certain restrictions: 1) some positions are designated to fixed content providers; 2) different positions hold contents from different content providers, which are independent from each other; 3) combining multiple positions can lead to a huge action space. These restrictions have resulted in independent ranking decisions for different positions, meaning CRO does not take other positions' information into consideration.

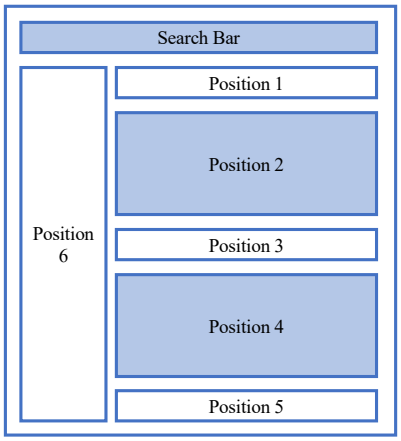


Figure 1: Content Positions

In this paper, we propose to utilize a multi-agent deep reinforcement learning (RL) solution (MADDPG) to tackle the CRO problem, which aims to 1) achieve a joint whole-page level contextual optimization across multiple positions, by learning the interactions among positions and generating whole page contents combinations; 2) optimize for long term benefits instead of instant benefits; 3) improve customer experience to better fulfill customers' shopping missions. The optimizer of each position is referred as an "agent", and contents of each position are referred as "actions". All agents act cooperatively to receive collective whole page level rewards. We utilize full RL to optimize for the accumulative future rewards during the entire customer journey, instead of heuristic instant rewards. We believe this novel method is applicable and innovative to other joint optimization applications in e-commerce.

The rest of the paper is organized as follows: Section 2 describes the related work used in the industry; Section 3 describes the CRO problem formulation; Section 4 describes the proposed solution; Section 5 describes the evaluation and results; and Section 6 summarizes our work and discusses future research directions.

2. Related Works

Reinforcement learning has gained significant attention in the field of e-commerce for ranking related problems, where the goal is to present customers with ranked list of items to maximize customer engagement and conversions. Specifically, we focus on the work addressing the high-dimensional action space in the ranking scenario.

Multi-Variate Testing (MVT) [3] tackles multi-dimensional joint optimization problem and is

widely adopted in e-commerce companies. It builds on top the of multi-armed bandits (MAB) model [4], and adopts the hill-climbing algorithm [3] to efficiently explore the high dimensional action space. One concern with applying MVT is the latency of neural networks. Hill-climbing in MVT requires several rounds of inferences. However, CRO typically utilizes the neural networks (NN) based models to estimate the objective. The motivation is to reduce manual feature engineering effort and accommodate advanced features into model, such as embedding. As a result, inference with NN models is expensive. Thus, it is intractable to meet the latency requirement of NN models and MVT.

Hierarchical RL (HRL) [5] was proposed to solve the joint optimization problem on e-commerce search page. Optimization is decomposed into 2 sub-tasks: 1) a source selector that decides which sources should be selected at the current page; and 2) an item presenter that decides the presentation order of the items selected from the selected sources in a page to each position. The model fully utilizes the sequential characteristics of user behaviors across different pages to decide both the sources and items to be presented. The item selector is similar to content ranking in CRO, but hierarchical structure is not required in CRO, thus HRL is not practical. However, this approach inspires the joint optimization between page layout optimization (PLO) [6] and content ranking, where PLO decides the page template at a higher-level and CRO decides the contents given the selected page layout at a lower level.

Reinforcement learning has also been applied in different ranking methodologies, e.g., list-wise recommendation [7] and page-wise recommendation [8]. The page-wise solution generates items and the strategy to display them on a two-dimension page.

In CRO, we firstly apply MADDPG to the whole page search ranking problem, providing a scalable solution for an industrial setting with tailored modifications.

3. Problem

3.1. Combinatorial Optimization

We formalize the content ranking optimization (CRO) as a combinatorial optimization problem, where the major challenge is to deal with a combinatorial action space. In CRO, an action is content. With more new positions and contents, the action space will grow exponentially. Formally, for a CRO environment with N positions and n_d discrete actions at each position d, there will be $\prod_{d=1}^{N} n_d$ possible actions to be considered. Table 1 lists the existing contents, containing the possible combinations and the actual max combinations from a single search request: (Actual max is less than the possible number because contents may have different traffic coverage).

Such a large action space is intractable for conventional single-agent single-dimensional RL algorithms, such as DQN [9], since it is difficult to explore efficiently [10]. One solution is to delegate the optimization responsibility to lower-level agents, where each agent optimizes its own dimension, and agents have a way to communicate [11] with other agents to cooperate. As a result, the action space grows linearly for each agent. We formulate CRO as a multi-agent RL due to: 1) optimality: this work evaluates both single-agent RL and multi-agent RL in evaluation Section 5, and demonstrate that multi-agent RL algorithm performs optimally, stably, and scalably in a CRO-like scenario; 2) flexibility: multi-agent RL applies a "centralized training

Table 1
Content Combinations in CRO

Platform: Desktop			
Positions	Region 1	Region 2	Region 3
Position 1	50	28	9
Position 2	68	52	26
Position 3	21	19	5
Position 4	18	13	3
Position 5	7	8	2
Position 6	14	14	4
Possible combinations	125M	40M	28,000
Actual Max Combinations	30,720	20,400	9,600

and decentralized execution" approach [11], where each agent is not required to access another agent's observation at execution time. This supports the flexible system structure enabling joint optimization across different applications. For instance, we can deploy trained agents to different services for online execution.

3.2. Multi-agent MDP in CRO

In this part, we formalize the problem in CRO using MADDPG. We consider a multi-agent extension of a MDP called Markov games [11] with the tuple $G = (N, S, \mathcal{A}, R, P, \mathcal{O}, \gamma)$. Here, N is the number of agents; $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_N\}$ is the discrete set of actions for the agents; S is the state space; R is the reward function; and $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_N\}$ is the set of observations for each agent. To choose actions, each agent i uses a stochastic policy $\pi_\theta : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$, which produces the next state according to the state transition function $P : S \times \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N \to S$. Each agent i obtains rewards as a function of the state and the agent's action $r_i : S \times \mathcal{A}_i \to \mathbb{R}$, and receives a local observation correlated with the state $o_i : S \to \mathcal{O}_i$. The goal of each agent i is to learn a policy $\pi_i : S \to \mathcal{A}$ which maximizes its own total expected return $R_i = \sum_{i=0}^{T} \gamma^t r_i^t$, where γ is a discount factor and T is the time horizon. Notably, since CRO is a fully cooperative environment, at each time step t, all agents observe the same joint reward $r_1^t = r_2^t = ... = r_N^t = r^t$. Concept mapping in CRO will be like $r1 * 0.99^0 + r2 * 0.99^1 + r3 * 0.99^3 + ...$

- $Environment$: e-commerce website and customers;
- $Episode$: a customer session on the same day;
- S: a customer's shopping state which represents the global context x, such as region, query, device, membership status;
- $TerminalState$: when the session is abandoned (no more interactions) at the end of the day;
- P : state transition probability;
- N : position agents to choose content at their respective positions, such as top/middle/bottom/side positions;
- $\mathcal{A}_i$: content candidates at position i;

- $\mathcal{O}_i$: the observation of position i, consisting of two parts:
  global observation x;
  local observation $\wr_i$ regarding the content features at position i, such as CTRs, ads bids, etc.;
- R: the reward which CRO receives from the environment, detailed discussion at 4.2 section;

At each time step, a customer issues a request to CRO, CRO observes the customer's shopping state, chooses a content combination and display to the customer, and then receives the rewards from the session. CRO goal is to learn the policy π_i for each agent i which maximizes the expected return across all sessions.

3.3. Mix-offline RL System in CRO

It is intractable to collect business rewards instantly online, which is the key limitation preventing online RL. As Figure 2 shows, CRO is defined as a mix-offline RL system in the current scenario: every day, a trained agent is deployed online to interact with the environment; at the end of the day, the trajectories are collected to train the agent, then the trained agent is deployed online to act. CRO is closer to an offline RL scenario where the agent can interact with the environment.

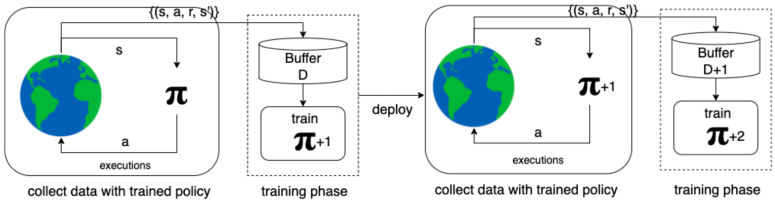


Figure 2: Mix-Offline RL System in CRO

4. Modeling

4.1. MADDPG

We utilize the MADDPG model [12] to solve the CRO problem. MADDPG is a multi-agent version of actor-critic method (MAAC) [13]. It adopts the framework of centralized training with decentralized execution approach [11]. A key characteristic of MADDPG is utilizing a fully observable critic which involves the observations and actions of all agents to ease joint training. Each agent trains a Deep Deterministic Policy Gradient [14] model, which concurrently learns a Q-function and a policy. The actor $\pi_{\theta_i}(o_i)$ with policy weights θ_i observes the local observations o_i, while the critic Q_i^{μ} is allowed to access the observations, actions, and the target policies of all agents during training. The critic of each agent concatenates all state-actions together as the input and uses the local reward to obtain the corresponding Q-value. Formally, consider an environment with N agents and policies parameterized by $\theta = \{\theta_1, ..., \theta_N\}$, and

$\pi = \{\pi_1, ..., \pi_N\}$ as the set of agent policies. The gradient of the expected return for agent i, $J(\theta_i) = \mathbb{E}[R_i]$ is:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\pi}, a\sim\pi^{\theta}} \left[\nabla_{\theta_i} \log \pi_i(a_i|o_i) Q_i^{\pi}(x, a_1, ..., a_N) \right] \tag{1}$$

where $Q^{\pi}(x, a_1, ..., a_N)$ is a centralized action-value function that takes the actions of all agents, $a_1, ..., a_N$ as input, and the concatenated state information $x = (o_1, ..., o_N)$, and outputs the Q-value for agent i. We then extend the work with deterministic policies μ_i with parameters θ_i, so the gradient can be written as:

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{x,a\sim D} \left[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^{\mu}(x, a_i, ..., a_N)|_{a_i=\mu_i(o_i)} \right] \tag{2}$$

Here the experience replay buffer $\mathcal{D}$ contains the tuples $(x, x^{'}, a_1,$ $..., a_N, r)$, recording the experience of all agents. The centralized action-value function Q_i^{μ} is updated as follows:

$$L(\theta_i) = \mathbb{E}_{x,a,r,x'} \left[(Q_i^{\mu}(x, a_i, ..., a_N) - (r + \gamma Q_i^{\mu^{'}}(x^{'}, a_1^{'}, ..., a_N^{'}))|_{a_j^{'}=\mu_j^{'}(o_j)})^2 \right] \tag{3}$$

where $\mu^{'} = \{\mu_{\theta_1^{'}}, ..., \mu_{\theta_N^{'}}\}$ is the set of target policies with delayed parameters $\theta_i^{'}$. This method resolves the non-stationarity issue [15], as $P(s^{'}|s, a_1, ..., a_N, \pi_1, ..., \pi_N) = P(s^{'}|s, a_1, ..., a_N, \pi_1^{'}, ..., \pi_N^{'})$ for any $\pi_i \neq \pi_i^{'}$, since the environment returns the same next-state regardless of the changes in the policy of other agents.

4.2. Reward

Rewards are critical to an RL system. In our setting, we formulate the reward as follows:

$$R_t = Revenue_t + Profit_t + \alpha * LongTerm_t + \beta * Clicks_t + \gamma * Abandonments_t \tag{4}$$

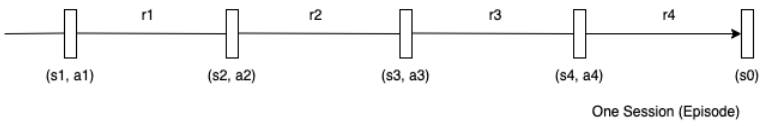


Figure 3: Rewards of One Episode

Clicks and Abandonments are part of the reward due to two reasons: 1) we aim to encourage more engagements in the session and reduce session abandonments, which are an unbiased indicators and indicate long-term value; 2) to enable rewards shaping [16] to

stabilize the learning since purchases are sparse. Figure 3 shows how to collect the rewards from the session. s represents the customer state, a represents the content combination displayed to the customer, and reward r_t is the total reward received between impressions (time steps). s_0 is the terminal state. From this episode, 4 transitions are collected: $(s_1, a_1, r_1, s_2, 0), (s_2, a_2, r_2, s_3, 0), (s_3, a_3, r_3, s_4, 0), (s_4, a_4, r_4, s_0, 1)$.

4.3. Exploration

We sample actions from the Gumbel Softmax [17] action distributions with a temperature of 1.0 for each agent. This is the conventional noisy-based method in RL. With higher temperature, the model encourages more explorations. For new and unrecognized contents, we will assign a fixed probability to ensure they can be explored.

5. Evaluation

We conduct extensive evaluations both in a public Mujoco environment [18] and on a CRO offline data set.

5.1. Evaluation On HalfCheetah-v2

5.1.1. Environment

We choose the Mujoco [18] environment HalfCheetah-v2 [19] to simulate the CRO scenario. The robot in this environment has six limbs, with each limb representing one continuous action $a_i \in [0.0, 1.0]$. The goal is to train agents to control these limbs cooperatively to move faster and receive higher rewards. Since CRO has a discrete action space - contents, we need to discretize the continuous space into a discrete space to bridge the gap. For instance, when the per-dimension action size is set to 5, we discretize the action space to be [1.0, 0.5, 0.0, 0.5, 1.0] and at every time step, the agents interact with the environment using one of those discretized values. If the max per-dimension action size is 5, the action space expands to $5^6 = 15625$; if action size is 50, the action space reaches $50^6 = 15.6B$, which simulates a extremely complex cooperative scenario.

5.1.2. Benchmarks

We leverage the following models as benchmarks:

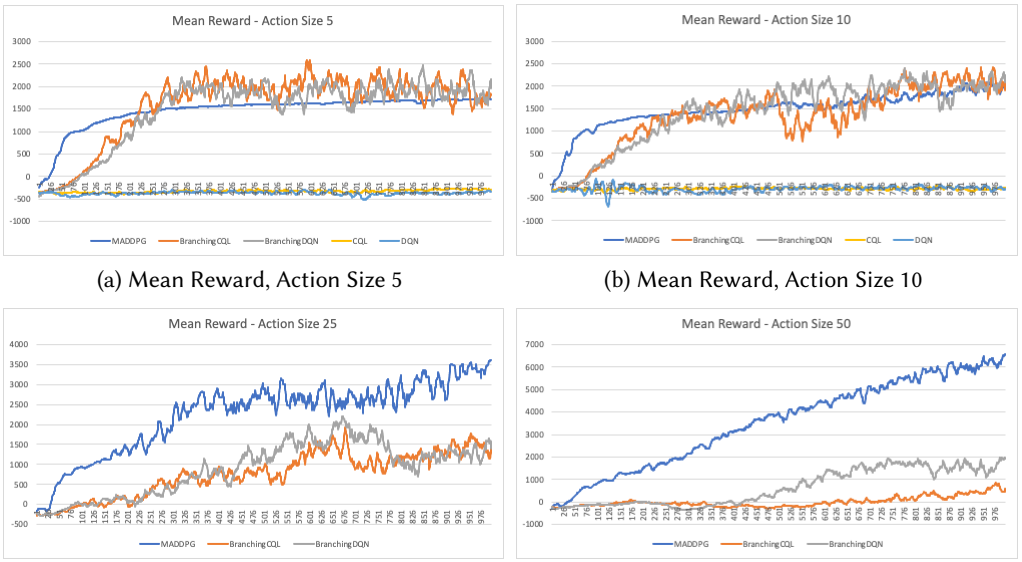
- **DQN**: The classical DQN model. We flatten multi-dimensional action spaces into a single dimension
- **CQL**: Conservative Q-learning [20], tweaked DQN model with a conservative Q function. CQL is designed to address the over-estimation issue [20] in an offline RL setting using an off-policy algorithm. We also flatten action spaces into a single dimension;
- **BranchingDQN**: The action branching architecture [10] was proposed to address the large action space problem in single-agent RL. The key insight is that to solve problem in combinatorial action space, it is possible to optimize for each action dimension with a

degree of independence. The shared network module computes a latent representation of the input state that is then propagated to the several action branches. Each branch is responsible for controlling an individual degree of freedom, and the concatenation of the selected sub-actions results in a joint-action tuple;

- **BranchingCQL**: A conservative version of BranchingDQN to fit into an offline RL scenario;
- **MADDPG**: This paper's proposed model.

5.1.3. Online Setting

We run models in HalfCheetah-v2 in an online fashion with the discrete per-dimension action sizes of [5, 10, 25, 50] resulting in [15K, 1M, 244M, 15.6B] action spaces. Models are run 3 times with 1,000 episodes and mean rewards are averaged. The CQL and DQN model cannot be tested with action size > 10, due to Out-of-Memory issues on the GPU. All policy and critic networks have the same 2 MLP layers with 256 hidden units and ReLU activation function [21]. Target networks are updated by the τ-soft update method [13] where $\tau = 0.001$. For training the CQL, DQN, and Branching*, we utilize an epsilon-greedy [22] strategy as the behavioral policy [23] with an exponential decay $\epsilon = (0.01 + 0.99 * e^{\frac{-steps}{1e5}})$; for MADDPG, we sample actions from a Gumbel Softmax [17] action distribution. The experience replay buffer is set to 105, and the batch size is set to 128 for all models.



(a) Mean Reward, Action Size 5

(b) Mean Reward, Action Size 10

(c) Mean Reward, Action Size 25

(d) Mean Reward, Action Size 50

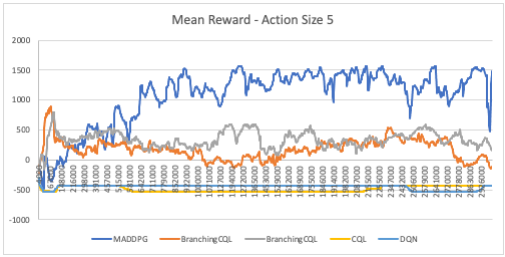
Figure 4: Model Performance in Online Setting

The results in Figure 4 indicates that typical DQN and CQL models are intractable for handling the large action space due to the insufficient explorations, which is expected. Branching models learn well at relatively low dimension action size but do not scale as well as MADDPG. MADDPG demonstrates consistent and optimal performance in all online experiments.

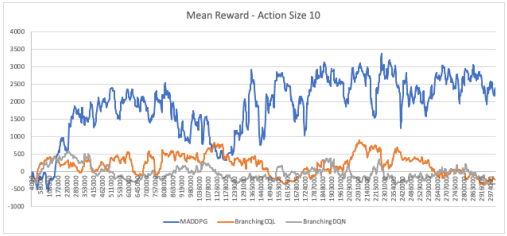
5.1.4. Offline Setting

The online study demonstrates that MADDPG scales well in an online setting. However, CRO is more suitable for an offline setting. Specifically, the CRO agent is required to learn from the trajectories collected by online policies at the end of the day. In this section, we conduct experiments to verify whether MADDPG can perform well in an offline setting. We still apply per-dimension action sizes of [5, 10, 25, 50] to observe its scalability. Offline learning consists of two steps: 1) collecting trajectories from some online policies (or random policy); 2) training the offline agent with these trajectories. To generate the trajectories, we adopt the D4RL approach [24]: first, we train an online MADDGP agent in a HalfCheetah-v2 environment, recording its mean rewards and saving its checkpoints until the model converges; second, we load 3 agents: 2 agents from checkpoints with 30% and 100% performance, and 1 random agent; then, we have the 3 agents interact with the environment (not train) to collect 3M trajectories equally (1M - 100% - expert, 1M - 30% - medium, 1M - random); finally, we train the models on this data set and observe the performance trending during training. All model settings are identical to the online setting.

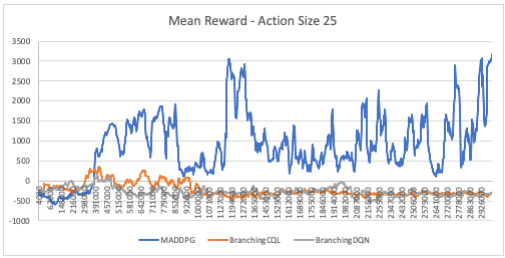
The results in Figure 5 demonstrates that 1) conservative models can learn better than DQN in an offline setting, which aligns with their design purpose, but CQL does not scale well with larger action sizes; 2) MADDPG scales well in the offline setting.



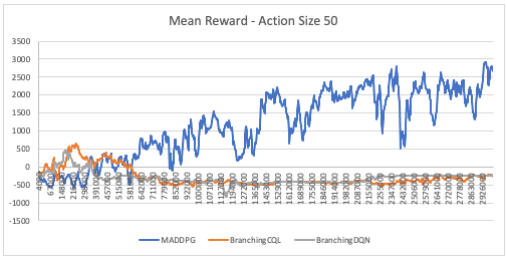
(a) Mean Reward, Action Size 5



(b) Mean Reward, Action Size 5



(c) Mean Reward, Action Size 5



(d) Mean Reward, Action Size 5

Figure 5: Model Performance in Offline Setting

5.2. Evaluation on CRO Data Set

In this section, we conduct experiments to evaluate MADDPG on the CRO data set. For simplicity, we extract a business metric as the reward for benchmarking. We evaluate the models on desktop

traffic only and consider joint optimization for 3 positions - top/middle/bottom positions, as they have the most impacts and content competition.

5.2.1. Trajectory Preparation

Trajectory data serves as the training source for RL. We follow a standard procedure to extract the training trajectories from CRO logs. As a result, we extracted 40M training trajectories from desktop traffic. The trajectories are from baseline policies: 5% random policy (generated by 5% epsilon greedy) and 95% deep bandits policy. For evaluation trajectories, we follow the same steps but use an exploration filter to extract 4M trajectories from the random policy.

5.2.2. Benchmarks

We leverage the following models and and rewards as benchmarks:

- **Random**: It randomly selects content combinations from available options.
- **Baseline (Multi-component reward)**: It is a baseline ranker with click-based Multi-objective Optimization objective.
- **Position-level bandits**: It is identical to the baseline model structure, past search queries are tokenized by the pre-trained SentencePiece tokenizer [25] and encoded by Embedding and LSTM [26] layers. Then we concatenate embedding with other inputs and feed it to an MLP layer with 256 hidden units. The final output is one unit with a Mean Squared Error (MSE) loss. This model is similar to the baseline, with the major differences in the features and reward.
- **Page-level bandits**: It utilizes an MLP structure with a multi-head output structure. Each "head" represents one position and outputs the content probability for available contents at this position. Features are the same as described above: past search queries, customer context, top position content ID, inline stop content ID, and bottom position content IDs.
- **MADDPG w/ gamma=0**: We train it on trajectories with a discounted factor=0.0. State is encoded by SentencePiece, Embedding and LSTM layers as well. Since the discounted factor is 0, this actually acts as a bandits model which optimizes for the instant reward without considering the future rewards.
- **MADDPG w/ gamma=0.99**: We train it on trajectories with a discount factor as 0.99; others settings are the same as described above.

The difference between the baseline and random models demonstrates the impact of the baseline model. The difference between position-level and page-level bandits presents the benefit of joint optimization. The difference between “MADDPG w/ gamma=0.99” and “MADDPG w/ gamma=0” illustrates the benefit of RL over bandits.

5.2.3. Metrics

We adopt the Inverse Propensity Scoring (IPS) method [27] to evaluate the trained policies: Given logged exploration data with a policy μ, and an evaluation policy π, the following IPS estimator of π is unbiased:

$$\hat{V}_{IPS}(\pi) = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}r_i \tag{5}$$

where n is the trajectory count, r_i is the trajectory reward. Since it is a random policy, $\mu(a_i|x_i) = \frac{1}{|a|}$ is extracted by softmax [11] from the output of models. The higher the IPS, the better the policy. Due to the high variance of the reward and IPS calculation, we choose 3 scores for the benchmark: 1) IPS on source rewards; 2) IPS on logged rewards; 3) IPS on indicator rewards (1 when source reward > 0 otherwise 0). The results are shown in Table 2.

Table 2
content Combinations in CRO

Model	IPS	IPS (logged)	IPS (indicator)
Random	0.92649 (0.91583, 0.93713)	0.10628 (0.10586, 0.10670)	0.15863 (0.15780, 0.15946)
Baseline	0.92887 (0.91815, 0.93958)	0.10639 (0.10597, 0.10681)	0.15895 (0.15811, 0.15978)
Position-Level Bandits	0.93238 (0.91965, 0.9451)	0.10679 (0.10632, 0.10726)	0.15970 (0.15874, 0.16067)
Page-Level Bandits	1.06876 (1.04743, 1.09008)	0.12579 (0.12486, 0.12672)	0.18569 (0.18383, 0.18754)
MADDPG gamma=0.0	1.12623 (1.11051, 1.14193)	0.12755 (0.12692, 0.12819)	0.19239 (0.19108, 0.19369)
MADDPG gamma=0.99	1.1684 (1.15286, 1.18394)	0.13353 (0.13288, 0.134188)	0.20086 (0.19954, 0.20218)

Table 2 demonstrates that MADDPG outperforms the the bandits model with logged and unlogged reward. The difference between "MADDPG with gamma=0.99" and "MADDPG with gamma=0" shows that RL is beneficial by considering state transition and delayed reward. The difference between page-level and position-level models shows that ranking holistically is beneficial. The baseline IPS is not high, which we assume is due to the composite effect of different objectives and the over-estimation issue. We also conducted an experiment to verify whether MADDPG can gradually learn during training. We periodically dump the model during training and evaluate it on the CRO evaluation data set and observe the IPS trending. Figure 6 shows the IPS trending, demonstrating that MADDPG can learn during offline model training.

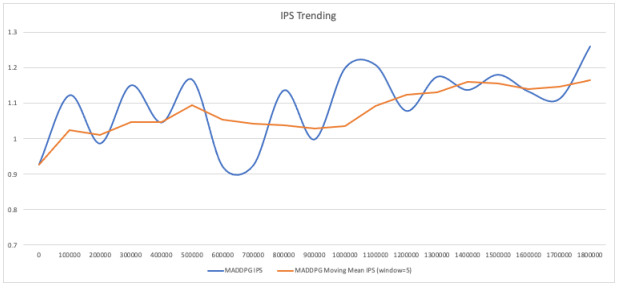


Figure 6: IPS Trend

5.3. Online A/B Test Performance

We implemented this approach on one of the leading e-commerce platforms and ran an A/B test to verify its performance. We tested with both discount factor as 0 (refer to as Treatment 1)

and 0.99 (refer to as Treatment 2), optimizing a multi-component reward function. By training and deploying it on the online traffic, both treatments demonstrated incremental multi-million revenue value. However, treatment 2 resulted in significant profit losses, leading to a flat overall reward, i.e., a larger discount factor resulted in poorer performance in our setting.

5.4. Evaluation Summary

The evaluation demonstrates that the MADDPG model scales well in both online and offline multi-agent environment setting, and is able to learn from the CRO offline data set. MADDPG also significantly outperforms the baseline like bandits model by 25.7% on IPS evaluation. We also evaluated this through online A/B test and the proposed solution brought incremental multi-million revenue.

6. Conclusion

In this paper, we frame page-level content ranking as a joint optimization problem, and tackle it using the novel MADDPG model. We conduct extensive experiments on both online Mujoco environment and an offline CRO data set. The evaluations demonstrate that MADDPG scales well up to a 2.5 billion action space and outperforms the baseline bandits model by 25.7%. In the future, we will further invest in multi-agent RL in several directions: 1) explainability: interpretation of why an agent takes certain action is important for content owners but this is not a trivial task, especially for neural network-based RL. Fortunately we have several directions to explore, such as reward decomposition [28] and shapley on Q-values [29]; 2) efficient exploration: exploration is critical for RL. This paper applies a noisy-based method which may not be the optimal for new contents. We will test entropy-based and variance-based exploration methods; 3) enhanced lifelong RL: retraining new models with a saved experience replay when encountering new contents is not ideal. We will research transfer RL to support the onboarding of new contents without needing to retrain the models.

References

[1] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, K. Gai, Deep interest evolution network for click-through rate prediction, in: Proceedings of the AAAI conference on artificial intelligence, volume 33, 2019, pp. 5941–5948.

[2] K. Deb, K. Sindhya, J. Hakanen, Multi-objective optimization, in: Decision sciences, CRC Press, 2016, pp. 161–200.

[3] D. N. Hill, H. Nassif, Y. Liu, A. Iyer, S. Vishwanathan, An efficient bandit algorithm for realtime multivariate optimization, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1813–1821.

[4] N. Silva, H. Werneck, T. Silva, A. C. Pereira, L. Rocha, Multi-armed bandits in recommendation systems: A survey of the state-of-the-art and future directions, Expert Systems with Applications 197 (2022) 116669.

[5] R. Takanobu, T. Zhuang, M. Huang, J. Feng, H. Tang, B. Zheng, Aggregating e-commerce search results from heterogeneous sources via hierarchical reinforcement learning, in: The World Wide Web Conference, 2019, pp. 1771–1781.
[6] Z. Qin, W. Liu, Automate page layout optimization: An offline deep q-learning approach, in: Proceedings of the 16th ACM Conference on Recommender Systems, 2022, pp. 522–524.
[7] X. Zhao, L. Zhang, L. Xia, Z. Ding, D. Yin, J. Tang, Deep reinforcement learning for list-wise recommendations, arXiv preprint arXiv:1801.00209 (2017).
[8] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, J. Tang, Deep reinforcement learning for page-wise recommendations, in: Proceedings of the 12th ACM conference on recommender systems, 2018, pp. 95–103.
[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).
[10] A. Tavakoli, F. Pardo, P. Kormushev, Action branching architectures for deep reinforcement learning, in: Proceedings of the aaai conference on artificial intelligence, volume 32, 2018.
[11] A. Oroojlooy, D. Hajinezhad, A review of cooperative multi-agent deep reinforcement learning, Applied Intelligence 53 (2023) 13677–13722.
[12] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Advances in neural information processing systems 30 (2017).
[13] V. Konda, J. Tsitsiklis, Actor-critic algorithms, Advances in neural information processing systems 12 (1999).
[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971 (2015).
[15] T. Kobayashi, W. E. L. Ilboudo, T-soft update of target network for deep reinforcement learning, Neural Networks 136 (2021) 63–71.
[16] A. Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: Icml, volume 99, Citeseer, 1999, pp. 278–287.
[17] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, arXiv preprint arXiv:1611.01144 (2016).
[18] E. Todorov, T. Erez, Y. Tassa, Mujoco: A physics engine for model-based control, in: 2012 IEEE/RSJ international conference on intelligent robots and systems, IEEE, 2012, pp. 5026–5033.
[19] Mujoco, Mujoco halfcheetah-v2., https://gym.openai.com/envs/HalfCheetah-v2/., 2023.
[20] A. Kumar, A. Zhou, G. Tucker, S. Levine, Conservative q-learning for offline reinforcement learning, Advances in Neural Information Processing Systems 33 (2020) 1179–1191.
[21] A. F. Agarap, Deep learning using rectified linear units (relu), arXiv preprint arXiv:1803.08375 (2018).
[22] C. Dann, Y. Mansour, M. Mohri, A. Sekhari, K. Sridharan, Guarantees for epsilon-greedy reinforcement learning with function approximation, in: International conference on machine learning, PMLR, 2022, pp. 4666–4689.
[23] S. Fujimoto, D. Meger, D. Precup, Off-policy deep reinforcement learning without exploration, in: International conference on machine learning, PMLR, 2019, pp. 2052–2062.
[24] J. Fu, A. Kumar, O. Nachum, G. Tucker, S. Levine, D4rl: Datasets for deep data-driven

reinforcement learning, arXiv preprint arXiv:2004.07219 (2020).
[25] T. Kudo, J. Richardson, Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, arXiv preprint arXiv:1808.06226 (2018).
[26] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.
[27] E. J. Williamson, A. Forbes, Introduction to propensity scores, Respirology 19 (2014) 625–635.
[28] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, F. Doshi-Velez, Explainable reinforcement learning via reward decomposition, in: IJCAI/ECAI Workshop on explainable artificial intelligence, 2019.
[29] J. Wang, Y. Zhang, T.-K. Kim, Y. Gu, Shapley q-value: A local reward approach to solve global reward games, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 7285–7292.