Exploring Algorithms Solving Multi-Armed and Dueling Bandits

Jørn Eirik Betten

May 20, 2024

Abstract

This work investigates Multi-Armed Bandits (MABs) and dueling bandits. MABs are problems where a decision-maker, which we refer to as an 'agent', has to choose an action, or an 'arm'. Because reward function engineering is challenging for many tasks, we look into an alternative, a preference model, where an observer, in our case an unknown reward function, ranks the actions. The main purpose of this work is to familiarize myself with MABs and dueling bandits, possibly laying the ground for future work in reinforcement learning with human feedback, which may be interesting for my thesis which involves explainable reinforcement learning (XRL). All code can be found at my GitHub[1].

1 Introduction

The study of decision-making under uncertainty has been a cornerstone of scientific research for decades. A key problem in this field is the exploration-exploitation trade-off, which is neatly captured in the multi-armed bandit (MAB) framework. This paper aims to introduce me to the multi-armed bandit problem, a classic problem of probability theory that illustrates a fundamental tension in decision theory.

The multi-armed bandit problem is a model for sequential decision-making, where an agent is faced with the challenge of choosing between different options (the 'arms' of the bandit) over sequential rounds. Each choice yields a random reward drawn from an unknown distribution. The objective is to devise a strategy that maximizes the total reward over a given time horizon. The fundamental challenge of the MAB problem is balancing the exploration of untested options with the exploitation of options that have provided good rewards in the past.

The concept of multi-armed bandits has been a focal point in the realm of decision-making under uncertainty. As a variant of this foundational model, especially in scenarios where real-valued feedback is difficult to produce, dueling bandits have recently emerged, offering a unique and intriguing perspective. Dueling bandits are a model for preference-based learning, where the goal is to identify the best option based only on noisy pairwise comparison feedback, rather than absolute reward values. This dynamic differs from traditional multi-armed bandit scenarios, where the feedback comes from individual arms. The dueling bandit problem introduces additional complexity, as the agent is required to infer the relative value of options based on indirect information, thus adding another layer to the exploration-exploitation trade-off.

[1] https://github.com/JornEirikBetten/dueling-bandits

This work aims to familiarize myself with the concepts of MAB and dueling bandit problems, with the long-term goal of building a foundation for learning more about preference-based reinforcement learning, like reinforcement learning from human feedback [1] and possibly using it in my work of explainable reinforcement learning, along with building intuitions for learning more about inverse reinforcement learning, a branch concerned with deducing the reward function from demonstrations. The implementation and review of popular algorithms e.g. the epsilon-greedy, upper confidence bound sampling, and Thompson sampling for the MAB problem, and interleaved filter, and Double Thompson sampling are the specific hurdles overcome in my introduction to the field.

The paper is organized as follows: first, there will be a Background section, introducing the MAB and dueling bandit problems, along with brief overviews of Bayesian learning and reinforcement learning. Next, there is a short section on related work, where some of the state-of-the-art of preference-based reinforcement learning and dueling bandit solvers are discussed. The Methods section presents the algorithms used to solve both the MAB and dueling bandit problems in this work. Next, we have a short experimental section explaining the underlying mechanisms of the two bandits used in this work. In the Results and Discussion section the empirical results and some discussion around these are presented. Finally, we conclude the findings in the Conclusions section.

2 Background

2.1 Multi-Armed Bandits

The Multi-Armed Bandit (MAB) [2] are problems where you try to determine the best arm, hereafter referred to as action, judged on some outcome of the chosen action. This outcome is often denoted, as in reinforcement learning, the *reward signal* which is a function from an action space $\mathcal{A}$ to the reals, $R : \mathcal{A} \to \mathbb{R}$. The problem is to determine the best action, i.e.

$$a^\star = \arg\max_{a \in \mathcal{A}} \mathbb{E}[R(a)] \tag{1}$$

in the shortest amount of time, accumulating the least amount of *regret*, the loss of rewards accumulated by sampling suboptimal actions. More specifically, the *total regret* is defined as

$$r(T) = T\mathbb{E}[R(a^\star)] - \sum_{t=1}^{T} R(a_t), \tag{2}$$

where we define the *time horizon* of the sampling process, T, to be the number of samples from the reward model, and a_t denotes the action chosen at time $t \in \{1, 2, \ldots, T\}$. The action chosen at time step t, a_t, is a random quantity, and it, therefore, makes more sense to evaluate the *expected regret*, $\mathbb{E}[r(T)]$, when evaluating different sampling procedures.

2.2 Dueling Bandits

In K-armed dueling bandit problems, the objective is to give a ranking of the arms. The model assumes a hidden preference matrix

$$\mathbb{P} = \{p_{ij}\}_{K \times K}, \tag{3}$$

where the preference p_{ij} determines the probability of action i beating arm j, modeled using the Bradley-Terry model,

$$p_{ij} = \frac{\mu_i}{\mu_i + \mu_j}, \tag{4}$$

where μ_k for $k \in [1, 2, \ldots, K]$ are some true performance measures (like the mean reward for choosing action k). Using only pairwise comparisons between actions, the aim is to determine the best action. A typical approach is to determine the *Condercet winner*, i.e., an action w where $p_{wk} > \frac{1}{2}$ for all $k \in \{1, 2, \ldots, K\} \backslash w$. However, there may not exist a Condorcet winner, and a more general approach is to consider the *Copeland winner*, i.e., the action that beats the maximal amount of other actions. In this work, the regret will be calculated by counting the number of actions less than the Copeland winner the chosen actions by the algorithm corresponds to.

2.3 Bayesian Learning

In Bayesian learning, we assume that the parameters of the model are random variables, and the objective is to update the distribution of the parameters given some data, $\mathcal{D}$. Bayes' theorem provides the update rule,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{5}$$

where $p(\theta|\mathcal{D})$ is the *posterior distribution* of the parameters θ given the likelihood of the data given the parameters, $p(\mathcal{D}|\theta)$, and the prior distribution of the parameters, $p(\theta)$, normalized by the marginal likelihood of the data, $p(\mathcal{D})$. In this section we have abused notation, calling all distributions p, letting the argument give the specification of the distribution.

The *predictive posterior distribution*, $p(d|\mathcal{D})$ for a new data point d then becomes

$$p(d|\mathcal{D}) = \int_{\Theta} \mathrm{d}\theta p(d|\theta)p(\theta|\mathcal{D}), \tag{6}$$

and uncertainty estimates for the prediction of the new data point can be estimated from the distribution.

Thompson sampling In Thompson sampling, the idea is to make a Bayesian model of the reward signal of each action in an MAB. More specifically, the reward signals in a K-armed MAB are modeled as K independent stochastic reward signals, $\{\theta_k\}_{k=1}^{K}$, with corresponding prior distributions, $p_k(\theta_k)$. The sampling procedure consists of drawing independently from the model distributions of all actions, comparing them, and finally drawing from the MAB using the action with the highest draw from the model distributions. The draw from the MAB is then used to update the model reward signal of that action. Bayesian update can be performed sequentially on new data, setting the posterior given previous data points equal to the new prior. We then update the model distribution when observing a new data point using Bayes' theorem again.

2.4 Reinforcement Learning

We model the system as a Markov Decision Process (MDP) [3], which is a tuple, $M = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ is the state space of the environment, $\mathcal{A}$ is the agent's action space, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, is the transition function, determining the next state from the agent acting in a state. Finally, γ is the discount factor, representing the urgency of the agent accumulation of rewards. The goal is to maximize the accumulated rewards, $G_k = \sum_{t=1}^{\infty} \gamma^t r_{k+t}$.

Value-Based Reinforcement Learning The foundations of *value-based* RL are the *Bellman equations*, more specifically, the recursive relations of the *state-value function*, the *action-value function*, and the *action-advantage function*. Assuming the MDP tuple represents the problem, the

state-value function, $v_\pi : \mathcal{S} \to \mathbb{R}$, is a function giving the expected accumulated rewards following policy π from state $s_t \in \mathcal{S}$. Formally,

$$v_\pi(s_t) = \mathbb{E}[G_t|s_t, \pi] = \sum_{a_t} \pi(a_t|s_t) \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)[R(s_t, a_t, s_{t+1}) + \gamma v_\pi(s_{t+1})], \tag{7}$$

where $R(s, a, s')$ is the reward the agent receives for taking action a in state s, and the system transitions into state s', which probabilities are defined via the transition function $P(s'|s, a)$. This relation is based on a *stochastic policy* π, meaning that $\sum_a \pi(a|s) = 1$ and $\pi(a|s) \in [0, 1]$ for all actions $a \in \mathcal{A}$ for all states $s \in \mathcal{S}$. Similarly, we define the action-value function for a state-action pair as

$$q_\pi(s_t, a_t) = \mathbb{E}[G_t|s_t, a_t, \pi] = \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)[R(s_t, a_t, s_{t+1}) + \gamma v_\pi(s_{t+1})], \tag{8}$$

where the action at time step t is now determined, and the policy π is followed for consecutive time steps. The action-advantage function is the difference in value between the state-value function and action-advantage for a specific action chosen, given the policy π. Formally,

$$a_\pi(s_t, a_t) = v_\pi(s_t) - q_\pi(s_t, a_t), \tag{9}$$

yielding the advantage of taking action a_t at time step t compared to following the policy π. The estimation of these value functions therefore becomes crucial to reinforcement learning, and deep learning schemes often incorporate a *temporal difference* based approach to train an estimator for a value function, like in Deep Q-Network (DQN) [4] and Soft Actor-Critic (SAC) [5], where the action-value is estimated. DQN is an example of an *off-policy* learning algorithm, where experiences are stored in a replay buffer, possibly reused for training.

Policy-Gradient Reinforcement Learning In policy-gradient reinforcement learning methods, we use the likelihood ratio method popularized by REINFORCE [6]. The idea is to sample a trajectory $\tau = \{s_t, a_t\}_{t=0}^T$ from a probability distribution dependent on the parameterized policy, such that $\tau \sim p(\tau|\theta)$ with accumulated rewards $r(\tau) = \sum_{t=0}^T \gamma^t r_{t+1}$, where θ represent the policy parameterization. The expected accumulated rewards can then be written as

$$\mathbb{E}[r(\tau)] = \int_{\mathbb{T}} \mathrm{d}\tau p(\tau|\theta) r(\tau), \tag{10}$$

where $\mathbb{T}$ is the space of all possible trajectories. The gradient of the expected accumulated rewards with respect to the policy parameters can be computed through

$$\nabla_\theta \mathbb{E}[r(\tau)] = \mathbb{E}\left[\sum_{t=0}^T \gamma^t r_{t+1} \nabla_\theta \log \pi(a_t|s_t, \theta)\right]. \tag{11}$$

This gradient can be deduced by recognizing the trajectory probability distribution as

$$p(\tau|\theta) = I(s_0) \prod_{t=0}^T P(s_{t+1}|s_t, a_t) \pi(a_t|s_t, \theta), \tag{12}$$

where $I(\cdot)$ is a distribution over initial states. Using the formula $\nabla_x f(x) = f(x) \nabla_x \log f(x)$ to rewrite the gradient of Equation 10 with respect to the policy parameters as

$$\nabla_\theta \mathbb{E}[r(\tau)] = \int_{\mathbb{T}} \mathrm{d}\tau \nabla_\theta p(\tau|\theta) r(\tau) = \int_{\mathbb{T}} \mathrm{d}\tau p(\tau|\theta) r(\tau) \nabla_\theta \log p(\tau|\theta), \tag{13}$$

and identifying

$$\nabla_\theta \log p(\tau|\theta) = \nabla_\theta \left[\log I(s_0) + \sum_{t=0}^{T} \log P(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T} \log \pi(a_t|s_t, \theta) \right] \tag{14}$$

where only the policy terms are dependent on θ, leading to

$$\nabla_\theta \mathbb{E}[r(\tau)] = \int_{\mathbb{T}} \mathrm{d}\tau p(\tau|\theta) r(\tau) \nabla_\theta \log p(\tau|\theta) = \int_{\mathbb{T}} \mathrm{d}\tau p(\tau|\theta) \sum_{t=0}^{T} \gamma^t r_{t+1} \nabla_\theta \log \pi(a_t|s_t, \theta) \tag{15}$$

where we have identified the right-hand side of Equation 10. In practice, the policy gradient is estimated using trajectories sampled from $p(\tau|\theta)$, highlighting that the sampling has to be done at every optimization iteration, which makes policy-gradient RL an *online* learning procedure.

3 Related Work

3.1 Dueling Bandit

This work only scratches the surface of what has been done in dueling bandit research in the last decade. Among these works are Sheth and Rajkumar's ranking procedure PAIRWiS [7] where an active version of a spectral ranking procedure is combined with a pair selection procedure. Bengs et al [8] gives a survey of preference-based online learning procedures, focusing more on the properties of preference-based feedback. The *Copeland and Scalable Confidence Bound* algorithms provided in the work of Zoghi et al. [9] provide other confidence bounds methods. Another work examining the (approximate) ranking of items from pairwise comparisons using confidence bounds is the work of Heckel et al. [10].

3.2 Preference-Based Reinforcement Learning

Since Fürnkranz et al. released their paper in 2012 [11], the search for better ranking-based RL algorithms have been proposed. Just the year after, Wirth et al. (et al. includes Fürnkranz) published another work in the direction [12] and again in 2016 [13]. The review of the current state-of-the-art of reinforcement learning from human feedback (RLHF) from Kaufmann et al. [1] gave a thorough overview of the RLHF state-of-the-art and a deep analysis of the problems and opportunities related to models trained through human feedback.

4 Methods

4.1 Multi-Armed Bandits

In this section, we introduce a set of sampling procedures for the MAB problem. Specifically, we present the explore-first algorithm, the standard ϵ-greedy exploration strategy, successive elimination, and a Bayesian approach called Thompson sampling.

Explore-First The explore-first algorithm [2] is what you might expect; the first $M = N \cdot K$ sampling steps we explore all actions equally (N samples for each action). After exploration, we

perform the action with the highest estimated expected reward. The upper bound on the regret is approximately minimized when $N = (T/K)^{2/3}(\log T)^{1/3}$, yielding an upper bound on the regret,

$$r(T) \leq T^{2/3}\mathcal{O}\Big((K \log T)^{1/3}\Big). \tag{16}$$

We therefore set $N \approx (T/K)^{2/3}$ in our implementation (rounding to the nearest integer). Algorithm 1 shows how the explore-first algorithm is implemented.

Algorithm 1 Explore First: K, T

```
# K is the number of actions, T is the number of rounds.
N ← round((T/K)^{2/3})                    ▷ Choose number of exploration rounds.
# Exploration phase.
for all k ∈ 1, 2, ..., K do
    Sample N rewards from R(a_k).
    Collect the mean reward for action a_k, μ_k.
end for
Choose the empirically best action, k̂ ← arg max μ.
# Exploitation phase.
Sample rewards from a_k̂ for the remainder T − NK rounds.
```

Epsilon-Greedy Exploration In ϵ-greedy exploration, we balance the exploration-exploitation tradeoff through sampling random actions with probability ϵ at every sampling step, or making the greedy action choice, the action with the highest expected reward at the current sampling step, with probability $1 - \epsilon$. In our implementation, we use the minimizer of the upper bound of the regret from [2], the adaptive $\epsilon = t^{-1/3}$ for $t = \{1, 2, \ldots, T\}$ yielding an upper bound on the regret at each sampling step t,

$$r(t) \leq t^{2/3}\mathcal{O}\Big((K \log t)^{1/3}\Big). \tag{17}$$

The ϵ-greedy algorithm used in this work is shown in Algorithm 2.

Algorithm 2 Adaptive ϵ-Greedy: K, T

```
# K is the number of actions, T is the number of rounds.
for t = 1, 2, ..., T do
    ε ← t^{-1/3}                          ▷ Adaptive ε.
    Sample random number r from uniform distribution on [0, 1].
    if r < ε then
        # Explore.
        Sample k uniformly from {1, 2, ..., K}.
        Sample reward from R(a_k).
        Update sample mean for action k, μ_k.
    else
        # Exploit.
        Pick current best action, k ← arg max μ.
    end if
end for
```

Successive Elimination The successive elimination algorithm [2] maintains upper and lower confidence bounds on the expected rewards for each action. The best action is first selected randomly, and is swapped out if the lower confidence bound for another action is higher than the upper confidence bound for the currently best action. Similarly, we remove actions for which the upper confidence bound of the expected reward is lower than the lower confidence bound of the currently best action. The *confidence radii* are derived using the Hoeffding concentration inequality [14].

Algorithm 3 Successive Elimination: K, T

```
# K is the number of actions, T is the number of rounds.
Set all actions as active, A = {1, 2, ..., K}.
for t = 1, 2, ..., T do
    Sample rewards from all active actions and update all mean estimations, μ_k ∀k ∈ A.
    Use confidence radii r_t = sqrt(2 log T / t) to calculate high probability confidence intervals.
    if μ_k + r_t < μ_k' − r_t for any k, k' ∈ A then
        # Remove actions with rewards that are confidently lower than any other action.
        Remove action k from the set of active actions.
    end if
end for
```

Upper Confidence Bound Sampling The Upper Confidence Bound (UCB) sampling procedure maintains upper confidence bounds on the expected reward for each action. Each round samples the action with the highest upper confidence bound, ensuring that actions for which the expected reward has a large uncertainty are explored. Algorithm 4 shows a sketch of the implementation.

Algorithm 4 Upper Confidence Bound: K, T

```
# K is the number of actions, T is the number of rounds.
# Sample all actions once and calculate their upper confidence bound.
μ_k ← R(a_k); n_k = 1; r_k ← sqrt(2 log T / n_k);
Calculate upper confidence bounds, UCB_k ← μ_k + r_k.
for t = K + 1, K + 2, ..., T do
    Sample the action with the maximal upper confidence bound, k' ← arg max_k UCB_k.
    # Update upper confidence bound of action k'.
    n_k' ← n_k' + 1.
    μ_k' ← ((n_k' − 1)μ_k' + R(a_k')) / n_k'.
    r_k' ← sqrt(2 log T / n_k').
    UCB_k' ← μ_k' + r_k'.
end for
```

Beta-Bernoulli Thompson Sampling Thompson sampling [15] for the Beta-Bernoulli K-armed bandit sets a prior beta distribution on the Bernoulli parameters of each action. We then sample from these beta distributions to determine which action to sample from the MAB and update the distribution of the sampled action Bernoulli parameter. An overview of the algorithm can be found in Algorithm 5.

Algorithm 5 Beta-Bernoulli Thompson Sampling: K, T

```
# K is the number of actions, T is the number of rounds.
Define priors, α ← (α_1, α_2, ..., α_K); β ← (β_1, β_2, ..., β_K).
for t ∈ {1, 2, ..., T} do
    # Sample from the model reward signals.
    for k ∈ {1, 2, ..., K} do
        Sample rewards from model, r ← r_k ~ beta(α_k, β_k).
    end for
    Choose best action from model, k' ← arg max_k r.
    Sample reward from bandit using best action from model, R_{k'} ← R(a_{k'}).
    # Update reward model for action k'.
    α_{k'} ← α_{k'} + R_{k'}; β_{k'} ← β_{k'} + 1 − R_{k'}.
end for
```

4.2 Dueling Bandits

This section presents three procedures for determining the Copeland winners of K-armed dueling bandits, the *Interleaved Filter* [16] and *Double Thompson Sampling* (D-TS) [17].

Interleaved Filter (IF) The interleaved filter algorithm compares all actions in the K-armed dueling bandit with a current best action. It maintains confidence intervals for all duels with the current best action. When the algorithm is confident (meaning that the lower bound for the confidence interval of another action is higher than the upper bound for the current best) that an action is better than the current best, it removes the current best and swaps it for another. Similarly, it removes all actions that are confidently worse than the current best. In this way, the actions that are deemed suboptimal are removed until there is only one action left. The algorithm is shown in Algorithm 2 of [16].

Double Thompson Sampling In double Thompson sampling [17] the sampling procedure for the first action is based on the upper confidence bound, sampling from a beta distribution only from the top Copeland scores, as determined by $\mathrm{C-score}(i) = \frac{1}{K-1}\sum_k \mathbb{I}(u_{ik} > 0.5)$. The second action is sampled based on two criteria: previous wins and uncertainty. If the lower confidence bound is above 0.5, the second action of the pair will not be sampled. A schematic of the algorithm is shown in Algorithm 1 of [17].

5 Experiments

In the first paragraph, we present the synthetic K-armed Bernoulli bandit used to illustrate the effectiveness of the ϵ-greedy, successive elimination, upper confidence bound, and Thompson sampling algorithms used to solve MABs. The K-armed Gaussian bandit is used to compare algorithms solving the K-armed dueling bandit problem using only pairwise comparisons between the arms.

5.1 K-armed Bernoulli Bandit

A reward of $+1$ is given with probability p_k for each arm $k \in [1, 2, \ldots, K]$, with a corresponding probability $1 - p_k$ for sampling a reward of 0. The probabilities, p_k, of each arm were independently

drawn from a uniform distribution on $[0, 1]$. The K-armed Bernoulli bandit problem is used as a comparison generator between the MAB solvers presented in Section 4.1. Specifically, the algorithms are compared on the empirical expectation over 1,000 5-armed Bernoulli bandits of the regret-per-round, each algorithm ran with T =10,000 rounds in total.

5.2 K-armed Bradley-Terry Bandit

A K-armed Bradley-Terry bandit is determined by K values, which we will denote μ_k for $k \in \{1, 2, \ldots, K\}$. The preference matrix, $\mathbb{P} = \{p_{ij}\}_{i,j=1}^{K}$ is then determined from the $\{\mu_k\}$ as

$$p_{ij} = \frac{\mu_i}{\mu_i + \mu_j}. \tag{18}$$

When comparing two bandits i and j, the winner is determined stochastically with probability p_{ij}.

6 Results and Discussion

6.1 Multi-Armed Bandit

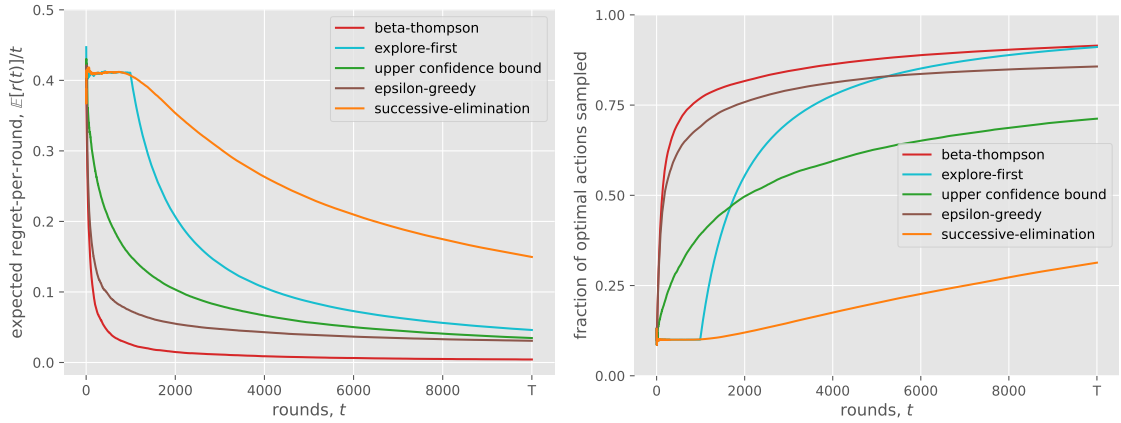
In this section, we provide the empirical evaluations of the algorithms in Section 4.1 on the 10-armed Bernoulli bandit.

Empirical evaluations of algorithms on the K-armed Bernoulli bandit This section contains an empirical evaluation of the algorithms used to solve the 10-armed Bernoulli bandit, explained in Section 5.1. The reasons for performing this experiment are two-fold; familiarizing myself with MABs and MAB-solvers, and comparing the performance of the algorithms.

Figure 1a shows the evolution of the mean regret-per-round over the 100 different bandits for every round. The Beta-Bernoulli Thompson sampling algorithm outperforms all the other sampling algorithms by quite a margin. Despite the explore-first algorithm successfully identifying the best action in all the 100 bandits it is still only outperforming successive elimination in terms of mean regret-per-round. The successive elimination algorithm seems to have too large confidence bounds, specifically, they were set to $\sqrt{2\log(T)/n_k(t)}$, where $n_k(t)$ is the number of times action k was sampled, struggling to remove suboptimal actions. Figure 1b, the fraction of optimal actions sampled, confirms this idea. The same confidence bounds used in the successive elimination algorithm were also used in the UCB algorithm, and the fraction of optimal actions chosen is also not as high for these algorithms as the others. The explore-first algorithm and Beta-Bernoulli Thompson sampling algorithms have identical fractions of optimally chosen actions after T =10,000 rounds. However, although the explore-first algorithm chooses the optimal action a similar amount of times after the completion of the 10,000 rounds, the accumulated regret is much higher, as the Beta-Bernoulli Thompson algorithm samples the actions based on the reward model, favoring sampling of actions with high modeled reward over the worse, while explore-first samples all actions equally in the beginning.

6.2 Dueling Bandit

This section contains the evaluations of the algorithms in Section 4.2 on the hidden preference matrix generated from 5-armed Bradley-Terry bandit are presented in the second paragraph.



(a) (b)

Figure 1: The mean regret-per-round **(a)** and mean fraction of optimal action choices **(b)** over 100 bandits in each of the 10,000 rounds each algorithm was run.

Empirical Evaluations of Algorithms on the K-armed Dueling Bandit In this section, the two algorithms presented in Section 4.2 are tested on the Bradley-Terry bandit introduced in Section 5.2 to show that they are implemented and to compare their qualities on a single 10,000 rounds-run. We report the regret-per-comparison (regret-per-duel) and the fraction of duels where the first chosen action of the procedure is the Condorcet winner in the left and right figures of Figure 2, respectively. We observe that the double Thompson sampling procedure has a much lower regret-per-comparison in the beginning and that it quickly finds the Condorcet winner as the first chosen action. However, in the interleaved filter algorithm, the algorithm continues until all rounds are done, and the 'dent' in the plot of the regret-per-comparison of the interleaved filter around 4000 comparisons, represents the point where the only remaining action is the Condorcet winner, so it starts comparing to itself. That is the reason for the interleaved filter algorithm beating the double Thompson sampling algorithm; the double Thompson sampling algorithm never compares the Condorcet winner to itself, but to other actions.

7 Conclusions

This work contains examinations of the MAB and dueling bandit problems, specifically implementations and background for exploration-exploitation trade-off algorithms using confidence bounds (UCB, successive elimination, interleaved filter), initial exploration before exploitation (explore-first), Bayesian modeling of reward signal (Thompson sampling, double Thompson sampling), and the epsilon-greedy algorithm. We found that the Thompson sampling procedures outperformed the others in both problems, although the competition was a bit too sparse in the dueling bandits problem.

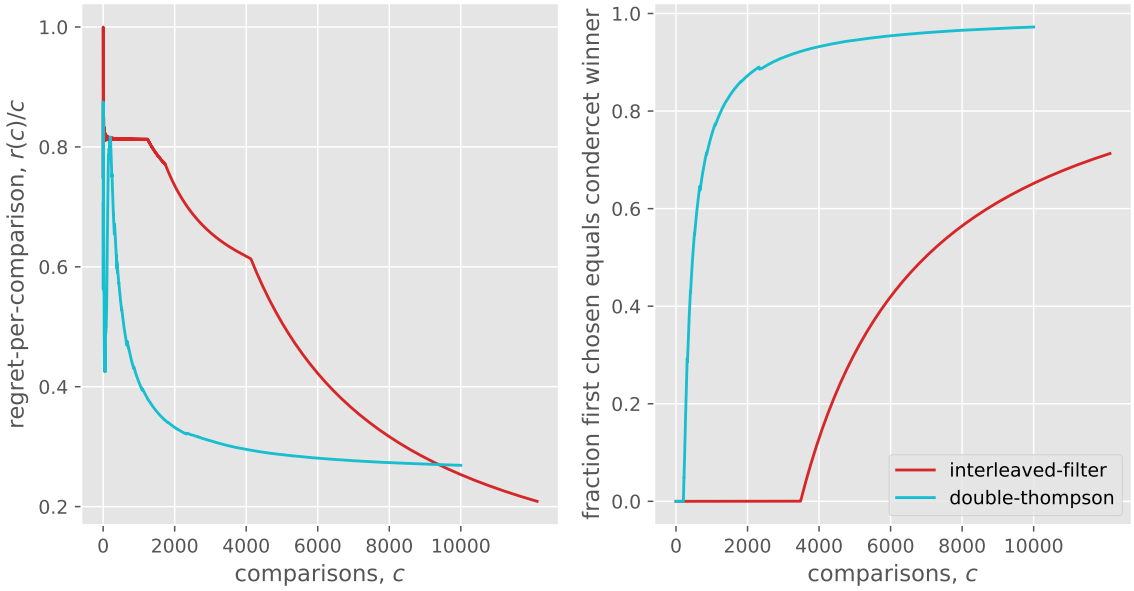


Figure 2: Left: Accumulated Copeland regrets-per-round for the interleaved filter and double Thompson sampling procedures. Right: The fraction of comparisons where the first action chosen was the Condorcet winner.

References

[1] T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier, *A Survey of Reinforcement Learning from Human Feedback*, arXiv:2312.14925 [cs], Dec. 2023.

[2] A. Slivkins, *Introduction to Multi-Armed Bandits*, arXiv:1904.07272 [cs, stat], Apr. 2024.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, en, Second edition. Cambridge, Massachusetts: The MIT Press, 2015.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, Number: 7540 Publisher: Nature Publishing Group, ISSN: 1476-4687.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," en, in *Proceedings of the 35th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2018, pp. 1861–1870.

[6] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Language*, vol. 8, no. 3-4, pp. 229–256, May 1992, ISSN: 0885-6125.

[7] D. Y. Sheth and A. Rajkumar, "PARWiS: Winner determination from Active Pairwise Comparisons under a Shoestring Budget," in *2021 IEEE International Conference on Data Mining (ICDM)*, ISSN: 2374-8486, Dec. 2021, pp. 569–578.

[8] V. Bengs, R. Busa-Fekete, A. E. Mesaoudi-Paul, and E. Hüllermeier, *Preference-based Online Learning with Dueling Bandits: A Survey*, arXiv:1807.11398 [cs, stat], Jul. 2021.

[9] M. Zoghi, Z. Karnin, S. Whiteson, and M. de Rijke, *Copeland Dueling Bandits*, arXiv:1506.00312 [cs], May 2015.

[10] R. Heckel, M. Simchowitz, K. Ramchandran, and M. J. Wainwright, *Approximate Ranking from Pairwise Comparisons*, arXiv:1801.01253 [cs, math, stat], Jan. 2018.

[11] J. Fürnkranz, E. Hüllermeier, W. Cheng, and S.-H. Park, "Preference-based reinforcement learning: A formal framework and a policy iteration algorithm," en, *Machine Learning*, vol. 89, no. 1, pp. 123–156, Oct. 2012, ISSN: 1573-0565.

[12] C. Wirth and J. Fürnkranz, "A Policy Iteration Algorithm for Learning from Preference-Based Feedback," en, in *Advances in Intelligent Data Analysis XII*, A. Tucker, F. Höppner, A. Siebes, and S. Swift, Eds., Berlin, Heidelberg: Springer, 2013, pp. 427–437, ISBN: 978-3-642-41398-8.

[13] C. Wirth, J. Fürnkranz, and G. Neumann, "Model-free preference-based reinforcement learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16, Phoenix, Arizona: AAAI Press, Feb. 2016, pp. 2222–2228.

[14] J. M. Phillips, *Chernoff-Hoeffding Inequality and Applications*, arXiv:1209.6396 [cs], Feb. 2013.

[15] W. Thompson, "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, Dec. 1933, ISSN: 0006-3444.

[16] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims, "The K-armed dueling bandits problem.," in *Journal of Computer and System Sciences*, vol. 78, Elsevier, 2012, pp. 1538–1556.

[17] H. Wu and X. Liu, *Double Thompson Sampling for Dueling Bandits*, arXiv:1604.07101 [cs, stat], Oct. 2016.