Optimization strategy for the pull arm function in the Multi-Armed Bandit algorithm

Shaozhi Pi

Viterbi School of Engineering, University of Southern California, Los Angeles, 90089, United States

spi@usc.edu

Abstract. Within the domain of Multi-Armed Bandit (MAB) quandaries, the efficacy exhibited by the arm-pulling function stands as a determinant factor governing algorithmic performance. This study undertakes a systematic exploration of enhancing this function's optimization vis-à-vis the MovieLens dataset, with a specific focus on three prominent MAB algorithms: Upper Confidence Bound (UCB), Explore-Then-Commit (ETC), and Thompson Sampling (TS). The present exposition introduces an innovative methodology tailored to the Python 3.11 computational environment. Central to this approach is the pre-calculation of arm-specific reward frequencies, obviating the necessity for dataset look-up and thereby streamlining the reward selection process. Subsequently, a binomial distribution is fitted to these frequencies, ensuring rapid and accurate reward sampling. When benchmarked, the optimized version of the pull arm function showcased a remarkable reduction in execution time compared to their initial counterparts, all while maintaining algorithmic accuracy. The Thompson Sampling algorithm notably outperformed the others, exhibiting the lowest cumulative regret across diverse trial sizes. This research emphasizes the significance of refining core components in MAB algorithms and paves the way for practical enhancements in real-world applications.

Keywords: Multi-Armed Bandit, UCB, ETC, TS, reinforcement learning, optimization.

1. Introduction

The Multi-Armed Bandit (MAB) problem was initially introduced by Robbins [1] in 1952 and now has since evolved into a fundamental quandary in reinforcement learning, encapsulating a pervasive conundrum of decision-making agents: the exploration-exploitation trade-off. In this problem, an agent is confronted with a set of options, or "arms", each associated with an unknown reward distribution. The agent's objective is to devise a strategy that maximizes cumulative reward over a sequence of trials, necessitating a balance between exploring less-known arms and exploiting arms known to yield high rewards.

With the advent of the MAB problem, various algorithms have been developed to tackle it. These include epsilon-greedy, Upper Confidence Bound (UCB), Explore-Then-Commit (ETC), and Thompson Sampling (TS). Each algorithm offers unique approaches to the exploration-exploitation trade-off inherent in the MAB problem [2]. epsilon-greedy, for instance, balances exploration and exploitation by occasionally choosing a random arm, while UCB and Thompson Sampling incorporate statistical confidence intervals and Bayesian inference, respectively, to guide their decision-making process. On the other hand, ETC dedicates an initial phase to exploration before committing to the arm that has yielded the highest reward. The emergence of these algorithms has significantly enhanced decision-making capabilities and found applications across various industries. For instance, Netflix leverages MAB algorithms to match users with thumbnails that align with their interests, thereby boosting click-through and play rates [3]. Similar to Netflix, Lyft utilizes the adaptability of MAB algorithms to adjust to rapidly changing environmental and market conditions [4].

However, the proliferation of application scenarios and the increasing volume of data has posed a challenge in the realm of algorithmic optimization, specifically with respect to accelerating computational performance. While various MAB algorithms have different time complexities, they all inevitably share one common step: the execution of the pull arm function. This function, which simulates the act of selecting an arm and receiving a corresponding reward, is a critical component of any MAB algorithm and a prime candidate for optimization.

Despite its importance, systematic discussions on optimizing this particular algorithm component are relatively scarce. Therefore, this paper will primarily focus on the performance optimization of MAB algorithms UCB, ETC, and TS, specifically those based on the MovieLens dataset. In the context of the MovieLens dataset, each arm represents a movie genre, and pulling an arm equates to recommending a movie of that genre and obtaining a user rating as a reward. Given its repetitive invocation throughout the MAB algorithm's execution, the efficiency of the pull arm function significantly influences the algorithm's overall performance.

This paper first introduces the initial implementation of the pull arm function looks up the rating for each genre upon every invocation. While this approach needed to be more complex and straightforward, it demonstrated computationally costly, particularly for extensive datasets and extended horizons. This paper then introduces an optimized version of the pull arm function, substantially reducing its time complexity. The proposed strategy hinges on pre-calculating the reward distribution for each genre and storing it in a data structure conducive to efficient sampling. Consequently, when the pull arm function is invoked, it can simply sample a reward from the pre-calculated distribution, thereby circumventing on-the-fly distribution calculation. Experimental results using the MovieLens dataset demonstrate that the optimized pull arm function significantly accelerates the execution of MAB algorithms without compromising their performance. This study’s contribution will benefit researchers and practitioners working with MAB problems, particularly those with large datasets and extended horizons.

2. Methodology

This section will provide a brief introduction to the three algorithms that have been optimized through the pull arm function. These algorithms included in this study are the UCB, ETC, and TS.

2.1. UCB

The UCB algorithm, concluded by Cesa-Bianchi et al. [5], is a widely adopted approach for the MAB problem due to its excellent balance between exploration and exploitation. It operates by maintaining an upper confidence bound for each arm, which is a measure of the potential reward that could be obtained from the arm. The algorithm then selects the arm with the highest upper confidence bound at each step. The UCB algorithm has been demonstrated to achieve logarithmic regret, making it a highly efficient solution to the MAB problem. The formula for the UCB algorithm applied in this paper is shown as follows:

$$UCB_i(t) = \hat{u}_i(t) + B\left(\frac{4\,log(t)}{T_i(t)}\right)^{1/2} \tag{1}$$

where $\boldsymbol{\hat{u}_i(t)}$ is the average reward of arm i until round t, B is a constant that controls the level of exploration (in this case, B = 4), and $\boldsymbol{T_i(t)}$ is the number of times arm i has been pulled until round t.

2.2. ETC

The ETC algorithm is a straightforward yet efficacious approach to the MAB problem introduced by Perchet et al. in 2015 [6] and then further researched by Garivier et al. in 2016 [7]. It operates in two phases: an exploration phase, where each arm is pulled an equal number of times, and a commit phase, where the arm with the highest average reward during the exploration phase is selected for all remaining pulls. The ETC algorithm is particularly effective when the differences in rewards between arms are large. The exploration phase is determined by a parameter $\boldsymbol{m}$, where each arm is pulled m times. The commit phase starts after $\boldsymbol{m * k}$ pulls ($\boldsymbol{k}$ being the number of arms). In this paper, the $m * k$ or the exploration phase is set be to 10% of total runs.

2.3. TS

Thompson Sampling is a probabilistic algorithm for the MAB problem, as detailed extensively by Agrawal and Goyal [8]. This algorithm maintains a probability distribution over the reward of each arm, which is updated as more pulls are made from the arm. At each step, the algorithm samples a reward from each arm's distribution and selects the arm with the highest sample. This paper assumes the mean reward of arm i at time t is modelled as a normal distribution:

$$F_i(t) \sim \mathcal{N}\left(\hat{u}_i(t), \frac{B}{T_i(t)}\right), t = k + 1 \tag{2}$$

where $\hat{u}_i(t)$ is the average reward of arm $\boldsymbol{i}$ until round $\boldsymbol{t}$, B is a constant that controls the level of exploration (in this case, $\boldsymbol{B}$ = 4), and$T_i(t)$ is the number of times arm $\boldsymbol{i}$ has been pulled until round $\boldsymbol{t}$.

2.4. Implementation details

In order to enhance the efficiency of these algorithms under investigation, this study has optimized the pull arm function, a shared component across all three algorithms. The arm function selects a random reward from each arm (or machine). The dataset employed in this study is the MovieLens dataset, which comprises information about movies, genres, and user ratings. In the initial setup of the experiment, movie ratings were designated as rewards, and genres were treated as arms or machines. Subsequently, the three algorithms were applied across ten independent experiments, each running 500, 5000, and 50000 times over arms $\boldsymbol{K}$ equal to the number of genres, respectively. Additionally, two performance metrics were selected for evaluating these algorithms:1) The total cumulative regret over each experiment. 2) The overall time taken by each algorithm.

The first metric assesses the performance of each algorithm and the precise loss following the optimization of the pull arm function. To formally define the regret, let $\boldsymbol{\mu}^*$ donated maximum reward in $\boldsymbol{n}$ round, $\boldsymbol{\mu_t}$ donated the reward at current $\boldsymbol{t}$ round, thus the cumulative regret can be formalized as:$\boldsymbol{Cumulative\ regret} = \sum_{t=1}^{n}(\boldsymbol{\mu}^* - \boldsymbol{\mu_t})$. The second metric represents the algorithm's running time before and after the optimization. These metrics provide a comprehensive evaluation of the effectiveness of the optimization process. The total setup is shown in Algorithm 1:

Algorithm 1. Implementation of MAB Algorithm

```
Input: number of arms $K$, number of experiments, number of rounds $N$
For each experiment:
        Initial $\mu^*$, $\mu_t$, and cumulative regret
        For $t$ = 1, 2…, $n$:
                Call the pull arm function and MAB function
                Calculate regret and add to cumulative regret
        End (Record the execution time)
```

The initial implementation of the pull arm function entailed retrieving the ratings for the selected genre each time the function was invoked. This was achieved by filtering the dataset for the selected genre and subsequently selecting a rating at random. While this approach was straightforward and comprehensible, it was also notably inefficient. Each call to pull arm necessitated a search through the entire dataset, which could be considerably large, to locate the ratings for the selected genre. This resulted in a time complexity of O(n), where n represents the size of the dataset. To optimize the pull arm function, the frequency of each rating for each genre was pre-calculated. This was accomplished by grouping the dataset by genre and rating and calculating the frequency of each rating within each genre. This process yielded a new data structure, rating_freq, which could be accessed quickly and efficiently to obtain the frequency of each rating for a given genre. This optimization reduced the time complexity of the pull arm function to O(1), marking a significant improvement. The final optimization involved pre-calculating the probability distribution for each genre based on the `rating_freq` data structure, resulting in another new data structure, rating_dist. Consequently, when pull arm was invoked, it simply sampled a rating from this distribution. This approach maintained the O(1) time complexity of the optimized pull arm function, while also making the reward selection process more accurate and representative of the actual rating distribution for each genre.

3. Results and discussion

In this section, this paper initially presents the results of three distinct algorithms (i.e., UCB, ETC, and TS) tested under Python3.11 environment, and each applied to the MovieLens dataset with both the initial and optimized versions of the pull arm function. Each algorithm is configured with an identical number of arms $\boldsymbol{K}$ and experiments. Subsequently, the focus shifts to performance analysis of the various algorithms. Additionally, the potential trade-off between computational efficiency and accuracy that may arise from optimizing the pull arm function was also explored.

Figure 1 for $\boldsymbol{N}$ = 500, Figure 2 for $\boldsymbol{N}$ = 5000, and Figure 3 for $\boldsymbol{N}$ = 50000 illustrate the graphical representation of the average cumulative regret concerning the UCB; Figure 4, Figure 5, and Figure 6 for $\boldsymbol{N}$ = 500, 5000, and 50000 for ETC; Lastly, Figure 7, Figure 8, and Figure 9 with $\boldsymbol{N}$ = 500, 5000, and 50000 for TS, respectively. In these experiments, the number of arms $\boldsymbol{K}$ is set to 18. Each algorithm was run for 500, 5000, and 50000 iterations, with each setup repeated across 10 independent experiments. The x-axis represents the number of iterations, while the y-axis denotes the average cumulative regret. The error bars indicate one standard deviation above and below the mean, providing a measure of the variability in the results.

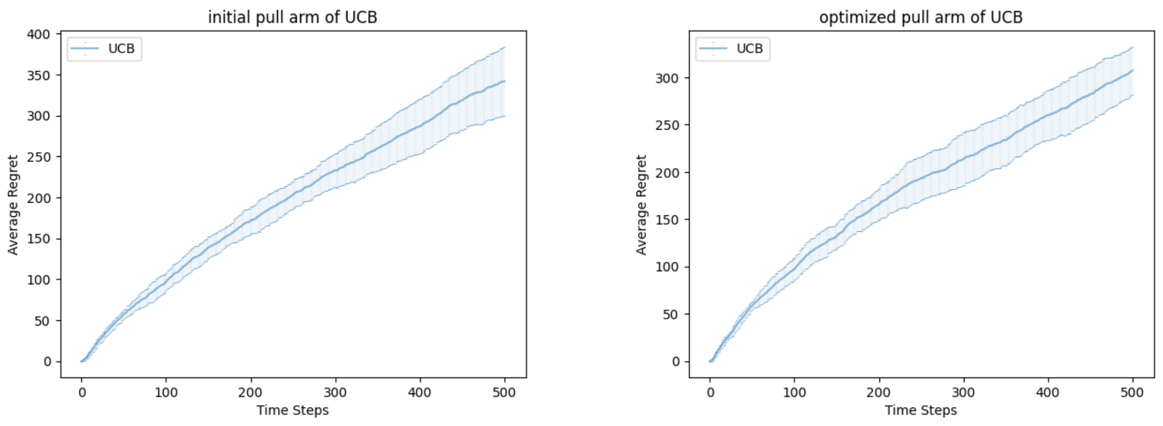

Figure 1. Cumulative Regret for initial and optimized pull arm of UCB with $\boldsymbol{N}$ = 500 (Photo/Picture credit: Original).

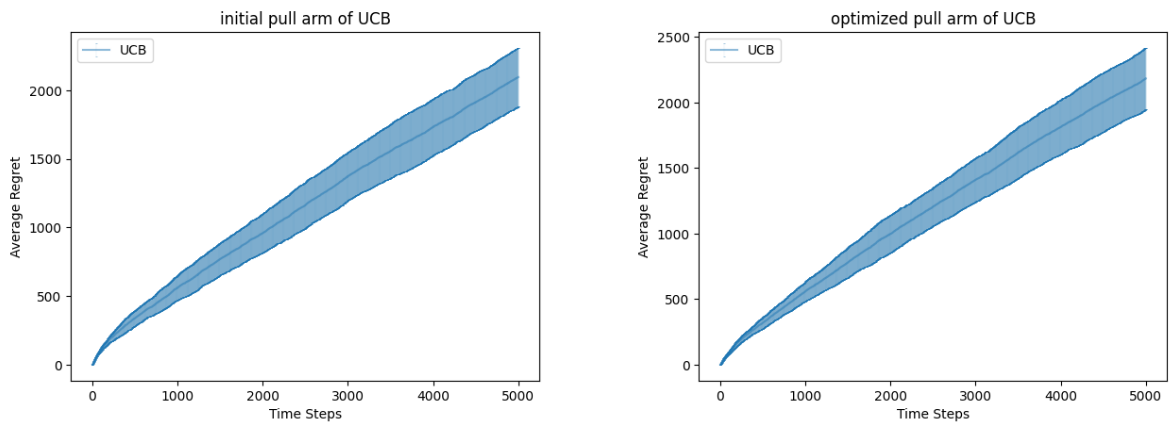

Figure 2. Cumulative Regret for initial and optimized pull arm of UCB with ***N*** = 5000 (Photo/Picture credit: Original).

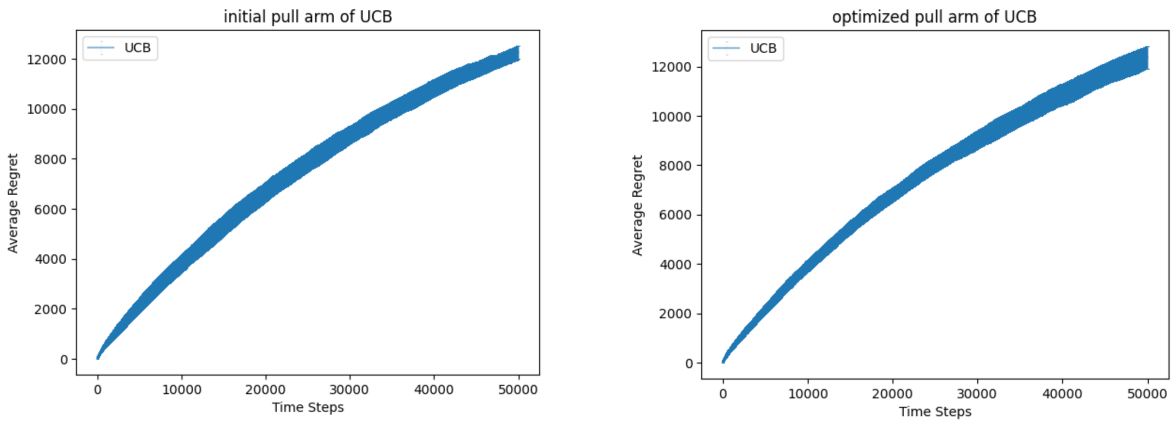

Figure 3. Cumulative Regret for initial and optimized pull arm of UCB with ***N*** = 50000 (Photo/Picture credit: Original).

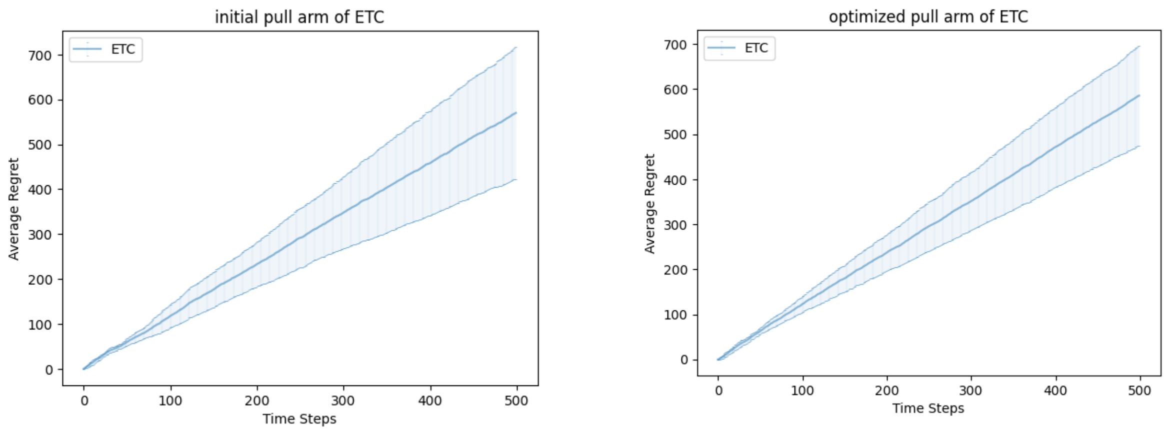

Figure 4. Cumulative Regret for initial and optimized pull arm of ETC with ***N*** = 500 (Photo/Picture credit: Original).

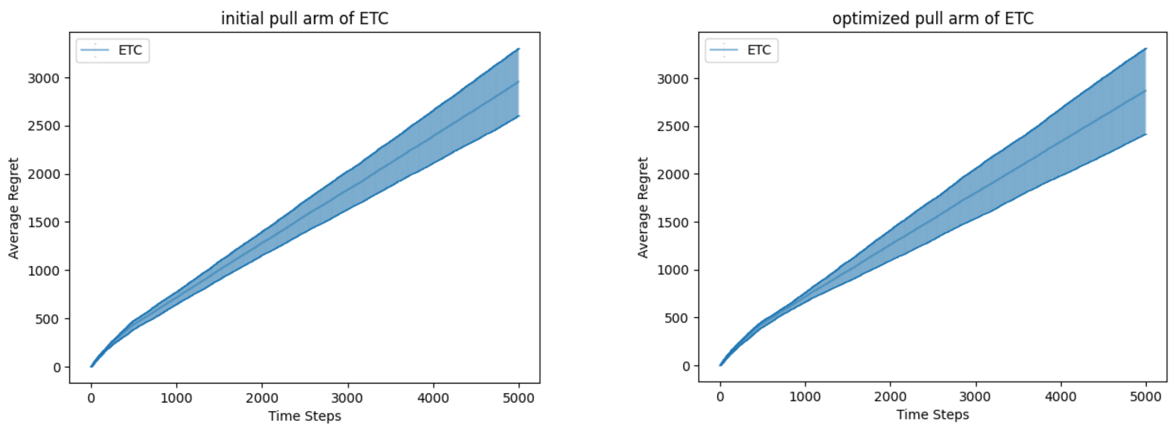

Figure 5. Cumulative Regret for initial and optimized pull arm of ETC with ***N*** = 5000 (Photo/Picture credit: Original).

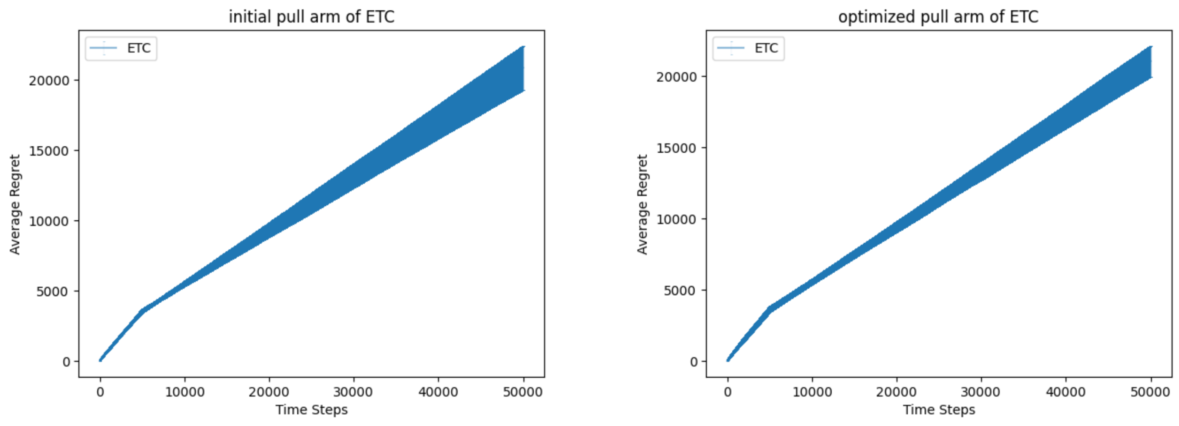

Figure 6. Cumulative Regret for initial and optimized pull arm of ETC with ***N*** = 50000 (Photo/Picture credit: Original).

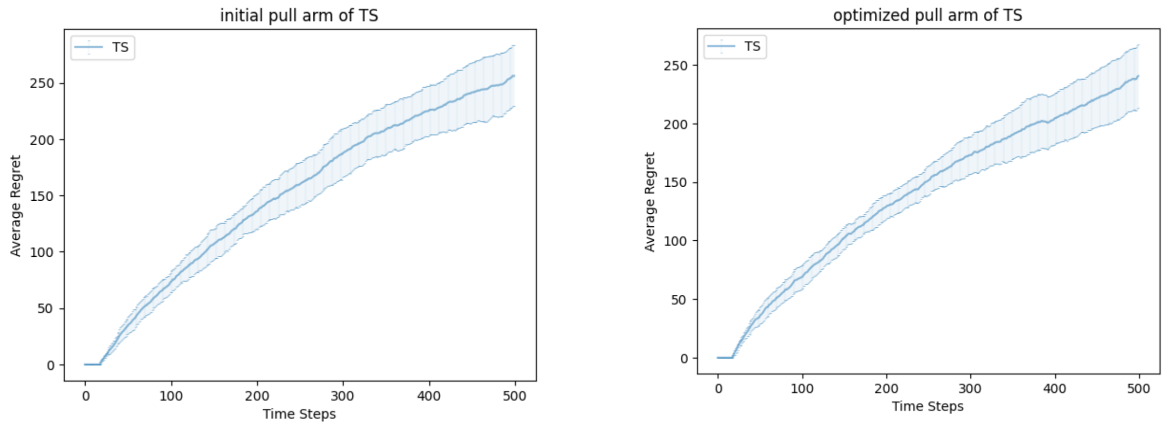

Figure 7. Cumulative Regret for initial and optimized pull arm of TS with ***N*** = 500 (Photo/Picture credit: Original).

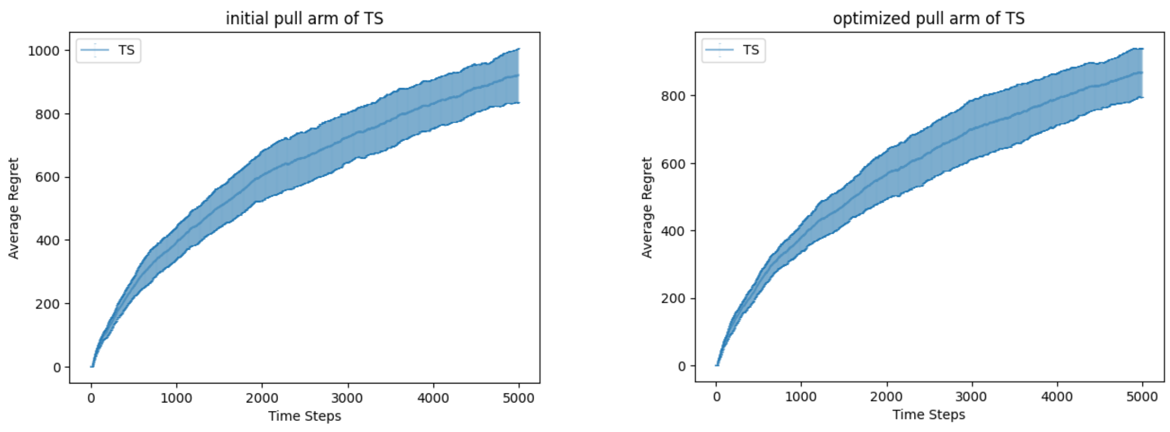

Figure 8. Cumulative Regret for initial and optimized pull arm of TS with ***N*** = 5000 (Photo/Picture credit: Original).

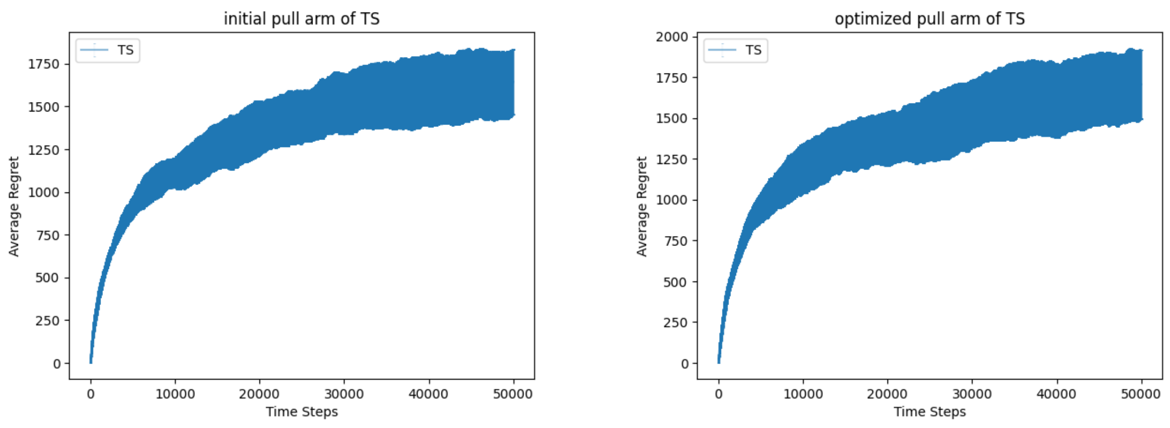

Figure 9. Cumulative Regret for initial and optimized pull arm of TS with ***N*** = 50000 (Photo/Picture credit: Original).

The initial observation derived from the results reveals that the TS algorithm consistently outperforms both the UCB, and ETC algorithms over time. Specifically, the TS algorithm exhibits the smallest cumulative regret across all three trial sizes Conversely, the ETC algorithm performs the worst, demonstrating the largest cumulative regret across all trial sizes. While the primary focus of this paper is not on comparing the performance of these three algorithms, this finding nonetheless presents an intriguing aspect of the results.

The results in Table 1 for ***N*** = 500, Table 2 for ***N*** = 5000, and Table 3 for ***N*** = 50000 clearly demonstrate the significant improvement in execution time achieved by optimizing the pull arm function across all three algorithms and for all values of ***N***. The execution time was reduced from several minutes (and even hours for ***N*** =50000) to mere seconds, representing a substantial efficiency gain. This improvement is particularly crucial for large-scale applications where the MAB problem needs to be solved repeatedly or in real-time.

In terms of average regret, the results are more nuanced. For UCB and ETC, the average regret slightly increased after optimization, but the difference is relatively small and within the range of standard deviation. In terms of the TS, the average regret decreased after optimization, indicating that the optimization not only sped up the algorithm but also slightly improved its performance. The standard deviation of regret also remained relatively stable after optimization, suggesting that the optimization did not introduce additional variability in the algorithm's performance. In addition, some more advanced algorithms e.g. neural networks may be considered for further improving the performance in optimizing average regret due to their satisfactory performance in other tasks [9-10].

Table 1. Summary of execution time, average regret, and standard deviation for ***N*** = 500.

	Initial UCB	Optimized UCB	Initial ETC	Optimized ETC	Initial TS	Optimized ETC
Time (Second)	346.7	0.1650	387	0.0080	747.9	0.1001
Average Regret	341.78	307.35	569.95	585.4	256.12	240.52
Standard Deviation	42.63	25.43	146.97	110.92	27.01	27.17

Table 2. Summary of execution time, average regret, and standard deviation for ***N*** = 5000.

	Initial UCB	Optimized UCB	Initial ETC	Optimized ETC	Initial TS	Optimized ETC
Time (Second)	3587.1	1.2837	4765	0.0872	4941.8	1.0366
Average Regret	2096.49	2183.66	2954.07	2867.02	920.59	867.75
Standard Deviation	213.26	233.7	350	448.52	85.14	71.75

Table 3. Summary of execution time, average regret, and standard deviation for ***N*** = 50000.

	Initial UCB	Optimized UCB	Initial ETC	Optimized ETC	Initial TS	Optimized ETC
Time (Second)	40901.4	12.2340	34989.2	0.8136	33800.2	10.3172
Average Regret	12253.52	12376.31	20862.61	21064.44	1643.16	1706.77
Standard Deviation	264.17	450.78	1561.36	1075.11	187.3	209.58

4. Conclusion

This paper introduced a method to optimize the performance of three MAB algorithms. The optimization of the pull arm function has been demonstrated to be highly effective in improving the efficiency of MAB algorithms without significantly affecting their performance. This optimization is particularly beneficial for large-scale applications where computational resources and time are critical. However, the optimization may not always lead to lower regret, as observed in the UCB and ETC results. Therefore, when applying this optimization, it is necessary to consider the trade-off between computational efficiency and algorithm performance. Future work could explore other optimization techniques the current method to achieve both faster execution time and lower regret.

References

[1] Robbins H 1952 Some aspects of the sequential design of experiments.
[2] Kuleshov V and Doina P 2014 Algorithms for multi-armed bandit problems
[3] Kawale J and Elliot C 2018 A MultiArmed Bandit Framework for Recommendations at Netflix
[4] Wang Y 2023 A Review of Multi-Armed Bandits Applications at Lyft
[5] Auer P Cesa-Bianchi N and Fischer P 2002 Finite-time analysis of the multiarmed bandit problem (Machine learning) 47 235-256.
[6] Perchet V Rigollet P Chassang S and Snowberg E 2016 Batched bandit problems
[7] Garivier A Lattimore T and Kaufmann E 2016 On explore-then-commit strategies (Advances in Neural Information Processing Systems) p 29

[8] Agrawal S and Goyal N 2012 Analysis of thompson sampling for the multi-armed bandit problem (JMLR Workshop and Conference Proceedings) pp 39-1
[9] Yu Q Wang J Jin Z et al 2022 Pose-guided matching based on deep learning for assessing quality of action on rehabilitation training Biomedical Signal Processing and Control 72: 103323
[10] Beltratti A Margarita S Terna P 1996 Neural networks for economic and financial modelling London UK: International Thomson Computer Press