New York University
Tandon School of Engineering

December 19, 2023

Analysis of Regret in Multi Armed Bandits with Extensions to ML in Regards to the One Dimensional Treasure

by

Kam, Henry Malik, Usaid

New York University – 2023

1 Introduction

In our project, we studied the paper *Improving Online Algorithms via Machine Learning Predictions* by Kumar, Purohit, and Svitkina. The paper outlined a framework for using Machine learning predictions to improve the performance of certain online algorithms. They then went on to define the competitive ratio, which is the ratio between the performance of an online algorithm, and the best offline algorithm. In our question, we initially began seeking to find a solution to the Online One-Dimensional treasure hunt problem.

We begin by defining the problem as follows: given a discrete one-dimensional line with only integers that range from $[-L, L]$ and an agent that starts at position 0 on the number line with nonoverlapping treasure that spawns randomly on any space once every time step on the number line, and a Time T which gives the number of time steps that the agent can execute three actions: stay still, move to the left, or move to the right, what is the maximum amount of treasure the agent can secure in the online setting.

Our project aimed to figure out the competitive ratio between the online version of this problem and the same offline version of this problem. we then wondered if machine learning predictions could be used to improve the competitive ratio. Upon working on the problem we discovered the class of problems known as Multi-armed Bandits (MABs), a class of online problems with many applications, reinforcement learning being one of them. We began to study algorithms relating to these classes of problems as we believed them to be similar to our original problem and we eventually discovered how we could instead improve the performance of algorithms used to solve the class of MAB problems using ML predictions.

2 Online 1D Treasure Hunt

As explained earlier the agent starts at position 0 on the number line. At each time step a single piece of treasure will spawn in on any of the positions on the line (including where the agent is). The agent may then choose to stay put, move to the left one space, or move to the right one space. If after the agent makes its decision it is at the position of some treasure, the treasure is marked collected and the agent updates their treasure count. The position where the agent was when they collected the treasure is marked open for collecting treasure again.

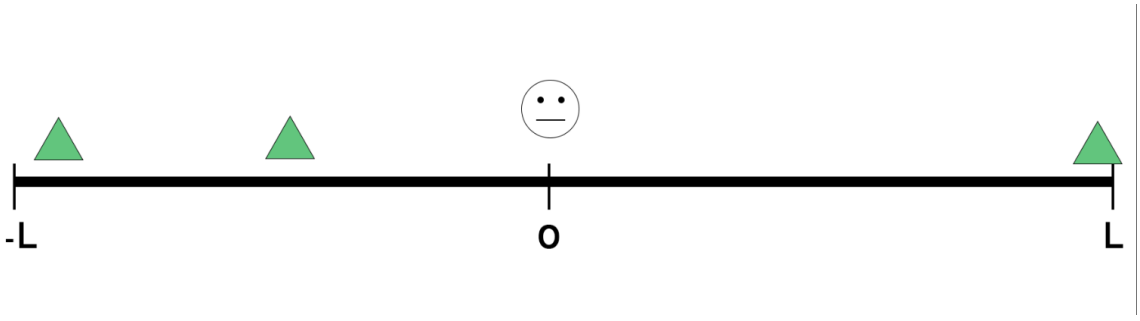

Figure 1: This agent chooses to remain at position zero despite three time steps having passed

We began with many variations through this problem, (Should we add some distribution to how treasures spawn?, Should there be weights on the treasure?, Should the agent be able to move more than two spaces ? etc.). We initially thought about allowing multiple treasures to spawn on the same positions, but we believed this may make the problem more difficult for us to get a good bound on the competitive ratio, so we stuck with the case where treasure may only spawn once on a single square. This however proved to be an issue later on.

In the offline algorithm we assume the same as the online algorithm except that now, all the treasure is already spawned in on the line, so the agent knows exactly where all the treasure is on the map and can go choose the best path to get the treasure. To find the best path, dynamic programming with a memoization-based approach can be used for all T, whereas in the dynamic programming-based approach the paths are examined and the max treasure after each time step can be found and then that treasure can be acquired.

We however, began by analyzing the case when $T \leq 2L + 1$ as this does not prove to be an issue in the offline setting. We took a look at the expected max treasure acquired in the offline case of the algorithm which we analyzed as follows: Since treasure can spawn on two sides, the left or the right side ignoring the zero position since it is currently relevant, a spawning oracle will spawn all the treasure onto the map. Treasure has a $\frac{1}{2}$ chance of being on either the left side or the right side. As such we would assume that half the treasure would be on the left side and the other half would be on the right side in expectation

the expected amount of treasure t on the left side or the right side of the map can further be analyzed by finding the probability of an item spawning in one space. Since the items cannot stack, this is similar to a probability question without replacement, so as the sides fill up, the probability of placing an item on that side decreases. The total amount of places an item can spawn is $2L + 1$, therefore, the probability that a single space gets a spot is $\frac{1}{2L+1}$ given that the iterations are at 0 as the iterations T increase the probability of a spot getting a space at time T is then $\frac{1}{2L+1-(T-1)}$ where the -1 term was added since the iterations start at $T = 1$.

We ended up realizing that that analysis wasn't necessary as we could reframe the problem once again, as instead of finding each probability individually, we could just find the proportion of the T treasures that were allocated to a length L portion of the line. As such the proportion of the line occupied by L is $\frac{L}{2L+1}$ then as such the expected number of treasures on that length L is then $E[t] = T\frac{L}{2L+1}$, where T is the number of time steps which is equivalent to the amount of treasure that is in the map in totality.

Similarly, the expected amount of treasure at the spawn point will be $E[t] = \frac{T}{2L+1}$ however this is irrelevant for the offline algorithm. Since we know that we start at spawn the two sides are symmetric, and $T \leq 2L + 1$ the max treasure we argue that the max treasure we could collect is $max(t) \leq E[t_l]$ where $E[t_l]$ is the expected number of treasure on the right or left side of spawn. We argue this is the case since an agent could look at the left or the right side, then just go left or just go right depending on which side has more treasure, and since $T \leq 2L + 1$ one side will have more treasure than the other side

We argue that $max(t)$ is less than the expected amount of treasure on a single side because if the agent goes to collect treasure on the left side and collects treasure on the right side, once the

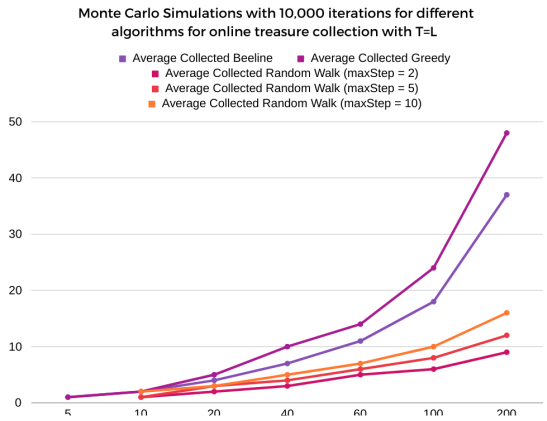

Figure 2: Graph of different simulations with T=L

treasure on that side runs out it takes the agent the same amount of steps to get back, and in the maximum case the time steps are $T = 2L + 1$ the agent would take L steps coming back and 1 extra step to get back to zero making them unable to explore the other side of the line. T

We recognize that this analysis could be improved by looking into the expectation of the events with dynamic programming for all T, but we decided to move on to analyzing the online version of this algorithm since we had begun to run into a problem in that, if $T \geq 2L + 1$ then our assumption of there not being more treasure on a single spot doesn't hold anymore as the extra treasure can't go anywhere, so we have to make it so that the treasure can overlap which proved to be very difficult to analyze since the best method would be the dynamic programming based approach to find the path that maximizes treasure.

So after being unable to solve that we attempted to then move on to analyzing different algorithms for solving the online version of this problem. We initially created some Monte carlo simulations for different algorithms and different sizes for T and L on the board. The simulations included a greedy simulation, where the agent goes for the closest treasure, a random walk simulation, where the agent goes left or right for some time k where k ranges between 1 and $maxStep$ where $maxStep$ is some maximum amount of steps the agent can take and then after depleting all its steps the agent chooses a new direction and step to take, or the agent stays still if a treasure spawns right on their head, a beeline simulation, where the agent will just go all the way left, or just go all the way right. After running the simulation the results were plotted below to try and figure out which online algorithm would work the best in this case and if ML predictions could be applied to what we found.

Analyzing the graph above the simulations showed that the greedy algorithm seemed to be doing the best for the case for $T = L$ and the beeline algorithm seemed to be close as well. The random walk algorithm did not do as well for different step sizes. Interestingly a somewhat exponential pattern was observed for the beeline algorithm and the greedy algorithm, hinting at a possible exponential relationship between the total treasure collected on average across all simulations and

the $T = L$ length and time case.

We also ran simulations for different combinations of T and L for different parameters and variations of the algorithm, the results of which we attach to this paper as CSV files.

After running the simulation we looked further into other possible relatives of this problem and other ways of solving it. We found that our problem resembled the MultiArmed Bandits problem especially its application in reinforcement learning, where an agent is given some actions to choose from and some rewards associated with those actions. We decided to further investigate MABs instead as a possible means to solving our problem since our previous attempts proved to not give us enough information.

3 Our work on MABs

We analyzed the problem of Multi-Armed Bandits (MABs) as a possible method for solving our original problem. MABs are a class of online learning problems involving finding the best strategy to maximize a given reward. MABs have a fundamental tradeoff between exploration and exploitation, by exploiting the agent may be able to acquire a higher reward for the remaining time steps, but by doing so they may risk losing out on more reward (exploitation) that they could exploit given a certain action. MABs have some parameters such as K the number of arms (actions) an agent may be able to choose from, and T the number of rounds a problem may be played for.

MABs may be visualized via gambling. An agent (gambler) may have access to a slot machine (bandit) with K levers (arms) which show the actions the gambler can take, he can pull any of the levers and see the reward the machine gives him. The agent fundamentally has to choose between exploring and exploiting, should they continue to exploit the lever they have now? Or try and explore a better lever in the hopes of winning it big.

With T rounds the gambler must maximize their reward and minimize their regret, regret ρ being the difference between the best arm with average reward $\mu^{(*)}$ and the average reward of all the arms agent chose $\sum^{T} \mu(a)$. MABs problems assume that for a given arm it has some probability distribution D_a unknown to the agent with some average reward $\mu(a)$ for that arm. MABs have many application domains and have been studied since 1930's as a problem in clinical testing. MABs further have many application domains such as in medicine (clinical research trials), computing (reinforcement learning), business (best ad to run), and many more. Our goal was to analyze and find the best algorithm for solving a MAB in the online case in hopes it would give us a solution to our original question. later one we wondered if we could use ML predictions to possibly improve the algorithm we chose for solving the MAB

Robustness and Consistency

In the paper we studied there is a notion of robustness and consistency which are then defined as follows

Robustness: An algorithm is considered robust if its performance does not degrade significantly under poor predictions. In the context of the epsilon-greedy algorithm, this would mean the

algorithm maintains reasonable performance even when the ML predictions about which arm to pull are inaccurate.

Consistency: An algorithm is consistent if it performs close to an optimal algorithm when the predictions are accurate. For an epsilon-greedy algorithm, this implies achieving near-optimal rewards when the ML predictions about the best arm to pull are correct.

It is important that we define these first before moving further,

Regret

Since different MAB problems may have different notions of reward we chose to analyze the regret as an unbiased indicator of the performance of some algorithm. Regret for an algorithm is defined as follows $\rho(T) = T * \mu^* - \sum^T \mu(a_i)$ where μ^* is the best possible mean from all actions the agent can choose, T is the number of times the agent operates in the scenario, and $\mu(a_i)$ is the expected reward of the action the agent chose at time i. This definition is called average or expected regret but we refer to it as regret.

4 Epsilon Greedy

Traditional Epsilon-Greedy Algorithm (Without ML Predictions)

Algorithm Basics The epsilon-greedy algorithm is a greedy algorithm using a parameter ϵ to regulate the trade-off between exploration and exploration. The algorithm sticks with the highest mean arm and chooses to exploit that arm with the highest estimated reward with probability $1-\epsilon$ and it does this for t rounds, and with a probability ϵ. the arm may choose to explore a different arm for t rounds in hopes of getting a higher reward. We set the number of arms N, and the mean reward of arm i be μ_i. The best arm has a mean reward μ^*.

Traditional Epsilon Greedy Algorithm

Regret Analysis The regret at time T, $R(T)$, is a measure of the performance difference between the algorithm and an optimal strategy.
Exploration Regret During the exploration phase, the algorithm chooses an arm at random. The expected regret from exploration is:

$$R_{explore}(T) = \epsilon \cdot \sum_{i=1}^{k} \frac{T}{k}(\mu^* - \mu_i)$$

where μ^* is the reward of the best arm, μ_i is the reward of arm i, and ϵ is the exploration probability.
Exploitation Regret In the exploitation phase, the algorithm selects the arm with the highest estimated reward. The regret from exploitation is:

$$R_{exploit}(T) = (1-\epsilon) \cdot \sum_{i=1}^{k} \delta_i(T) \cdot T(\mu^* - \mu_i)$$

where $\delta_i(T)$ represents the probability of incorrectly choosing arm i as the best arm at time T.
Total Expected Regret The total expected regret combines both exploration and exploitation regrets:

$$R(T) = R_{explore}(T) + R_{exploit}(T)$$

Consistency Analysis We examine the consistency of the Epsilon Greedy algorithm by looking at the convergence of the average regret. A consistent algorithm will have its average regret decreasing over time:

$$\lim_{T\to\infty} \frac{R(T)}{T} = 0$$

Epsilon Greedy Algorithm with ML Predictions

Regret Analysis with ML Predictions When the Epsilon Greedy algorithm has ML predictions, the regret calculation changes due to possibly inaccurate predictions. However, similar to Kumar's paper [1], we do not make any assumptions about the distributions of the prediction
ML-Influenced Exploration Regret Incorrect ML predictions can misdirect the exploration phase, affecting the regret:

$$R_{ML-explore}(T) = \sum_{i=1}^{k} P_i \cdot \frac{T}{k}(\mu^* - \mu_i)$$

where P_i is the probability of the ML model incorrectly predicting arm i as the best.
ML-Influenced Exploitation Regret The exploitation phase is also influenced by the ML predictions:

$$R_{ML-exploit}(T) = \sum_{i=1}^{k}(1 - P_i) \cdot \delta_i(T) \cdot T(\mu^* - \mu_i)$$

Total Expected Regret with ML Combining the exploration and exploitation regrets under ML predictions:

$$R_{ML}(T) = R_{ML-explore}(T) + R_{ML-exploit}(T)$$

Consistency Analysis with ML The consistency of the algorithm, when augmented with ML predictions:

$$\lim_{T\to\infty} \frac{R_{ML}(T)}{T}$$

The ideal consistent algorithm will have limit approach zero.

Numerical Analysis of Robustness and Consistency

Traditional Epsilon Greedy Algorithm
Robustness Metric (R_b) Robustness is measured by the variance in total regret across the different scenarios:

$$R_b = Var_{s\in S}[R_s(T)]$$

Consistency Metric (C_s) Consistency related to the inverse of the area under the regret curve:

$$C_s = \left(\int_0^T R_{traditional}(t)\, dt \right)^{-1}$$

Epsilon Greedy Algorithm with ML Predictions
Robustness Metric with ML (R_b^{ML}) Robustness in the ML-augmented scenario considers the variance in regret due to ML predictions:

$$R_b^{ML} = Var_{s \in S}[R_{s,ML}(T)]$$

Consistency Metric with ML (C_s^{ML}) Consistency with ML is measured by the inverse of the area under the ML-influenced regret curve:

$$C_s^{ML} = \left(\int_0^T R_{ML}(t)\, dt \right)^{-1}$$

This shows that the accuracy ML predictions strongly determine that accuracy of the online algorithm.

5 Conclusion

For our report, we first did a quick analysis on the Online 1-D Treasure Hunt problem, where we looked into the competitive ratio in both online and offline settings. This analysis led us to explore the problem of Multi-Armed Bandits (MABs), which is an online problem that has implications to many things, including reinforcement learning. We wanted to find ways to analyze the performances of online algorithms through machine learning predictions.

For our focus on MABs, we looked into the trade-offs between exploration and exploitation of the greedy approaches, where we analyzed the Epsilon-Greedy algorithm in both traditional forms and with Machine Learning predictions. We also added small robustness and consistency metrics.

We found that while ML predictions could significantly impact the algorithm's performance, we realized that its effectiveness is extremely dependent on the accuracy of these predictions.

In conclusion, our research highlights the intricate balance between algorithmic design and ML integration in solving online problems like the One-Dimensional Treasure Hunt and MABs.

While ML predictions could offer an exciting way to optimize these online algorithm performances, their effectiveness is highly dependent on the prediction accuracy.

5.1 References

Kumar R, Purohit M. Improving Online Algorithms via ML Predictions. [accessed 2023 Dec 17]. https://proceedings.neurips.cc/paper_files/paper/2018/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf.

Senno'tt L. Stochastic Dynamic Programming and the Control of Queueing Systems Chichester Weinheim Brisbane Singapore Toronto. [accessed 2023 Dec 17]. https://download.e-bookshelf.de/download/0000/5714/G-0000571462-0015280642.pdf.

Kuleshov V, Precup D. 2014 Feb 24. Algorithms for multi-armed bandit problems. arXiv:14026028 [cs]. https://arxiv.org/abs/1402.6028.

Slivkins A. 2020. Book announcement: Introduction to Multi-Armed Bandits. ACM SIGecom Exchanges. 18(1):28–30. doi:https://doi.org/10.1145/3440959.3440965.