Multi-Armed Bandit Model for Factory Safety Service Robot

IOE 512, Winter 2022
University of Michigan, Ann Arbor
Nicole Campbell (nicoca@umich.edu)

I. Introduction

Mobile robots are becoming more commonly used to perform repetitive tasks that humans have traditionally performed. It is especially beneficial when these tasks put humans at risk of harm. One example is Hyundai deploying a quadruped robot made by Boston Dynamics to perform safety inspections at their factories.
In this work, we investigate the use of modeling the problem of visiting stations in a factory for safety checks as a Multi-Armed Bandit. We applied a modified Bernoulli Thompson Sampling Algorithm to determine a sequence of stations to visit that maximizes the estimated reward of catching the most safety updates in the factory as well as minimize travel cost between stations. We introduce a parameter which is used to balance the importance of maximizing safety and travel efficiency. We perform sensitivity analysis on this parameter and the number of stations to observe how they affect the balance of exploration and exploitation in the model.

II. Background

A. Multi-Armed Bandit

Multi-Armed Bandit (MAB) problems are typically described as a slot machine with N arms (bandits), each having its own probability distribution of success. The goal is to find an optimal sequential strategy for pulling arms that maximizes the total expected reward. Pulling any one of the arms gives you a stochastic reward of either 1 for success or 0 for failure [1]. Some common algorithmns used to solve MABs are ϵ-Greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS).

B. Epsilon-Greedy Algorithm

The ϵ-Greedy Algorithm, as the name suggests, is "greedy" because it chooses the best option in the short term. Exploration is uniform over all arms with an exploration probability parameter, ϵ. The algorithm will explore actions with probability ϵ and exploit the arm with the highest average reward so far with probability $1 - \epsilon$ [1].

C. Upper Confidence Bound Algorithm

The Upper Confidence Bound Algorithm simply selections the action that has the highest empirical reward estimate up to that point plus some term that's inversely proportional to the number of times the arm has been played.

D. Thompson Sampling

Thompson Sampling utilizes a Bayesian model to represent uncertainty. In the algorithm, actions, x_t, are taken sequentially for $t \ \epsilon \ T$ to balance exploiting arms that are known to maximize immediate reward, r_t, and explorating through investing to accumulate new information that may improve future rewards. It solves a problem specified as the Beta-Bernoulli Bandit where there are K actions each with a probability of receiving a reward represented by θ_k and a probability of no reward of $1 - \theta_k$. These actual probabilities are unknown and fixed for all periods. The model keeps a prior belief for each θ_k which is modeled by a beta distribution with parameters α_k and β_k as shown in Equation (1) where Γ is the Gamma function. At each time period, t, after an action is taken, the model observes a reward of 0 or 1 and according updates the parameters using Equation (2). The initial Beta distribution before any observations will be Beta(1,1), where $\alpha_k = \beta_k = 1$, giving a uniform distribution. The estimate for the true probability can be calculated as the mean of the beta distribution as shown in Equation (3) [2].

$$p(\theta_k) = \frac{\Gamma(\alpha_k + \beta_k)}{\Gamma(\alpha_k)\Gamma(\beta_k)} \theta_k^{\alpha_k - 1}(1 - \theta_k)^{\beta_k - 1} \quad (1)$$

$$(\alpha_k, \beta_k) = (\alpha_k, \beta_k) + (r_t, 1 - r_t) \quad (2)$$

$$\bar{\theta_k} = \frac{\alpha_k}{\alpha_k + \beta_k} \quad (3)$$

III. Related Works

A. Target Search

There have been many recent papers utilizing MAB to model target searches for robots, both with multi-robot scenarios [3] as well as multi-target scenarios [4]–[5]. In these problems, a robot or a fleet of robots are in an unexplored environment with targets in unknown locations. The goal is for the robot(s) to explore and exploit the areas where targets are frequently found. In [6], the authors implement a modified Bernoulli Thompson Sampling (BTS) Algorithm where the travel distance to the target is taken into account. This is done by not only rewarding the model for choosing an action with maximum probability of finding a target, but also rewarding the model for reducing travel time when targets are repeatedly found in current exploited sub-regions. Since the nature of the search is that targets are often clustered, they added a posterior filter to revise the beta distributions to ensure that the robot does not skip searching the targets near the boundaries of clusters.

B. *Spare-Time Service Robot*

The work in [7] introduces the idea of a service robot that needs to make decisions about offering un-requested services to users in the robot's spare time between scheduled services. The authors uses both TS and UCB algorithms to learn how likely different users in the environment are to be interested in un-requested services. To add a traveling cost component, the action is chosen from a subset of users that are within the traveling distance constraint given how much spare-time the robot has.

IV. Problem Description and Methodology

We consider a problem where we have a service robot that performs safety inspections at N stations in a factory and we need to determine which stations need to be visited more frequently for safety checks. Each station will have a probability, θ_k, at which it will have an safety-critical issue or update that needs to be notified to an engineer in the facility. If the robot chooses to visit a station and there was something to update to the engineer, the robot gets a reward of 1, otherwise the robot gets a reward of 0.

In this problem, we model the factory as a 10x10 unit grid with the stations fixed at specific locations and the robot always starting at location $X_{robot} = [5, 5]$. The robot will travel to each station and will get additional rewards for minimizing travel cost at each iteration. The method of incorporating distance is inspired by the method in [6]. In traditional BTS, the action is chosen by finding the action that corresponds to the maximum sampled posterior distribution, $\hat{\theta}_k$. In our method, we instead use a weighted sum of the sampled posterior and the Euclidean distance between the stations and the robot which is mathematically represented in Equations (4) and (5). We pre-define a discount factor, c, that determines how much to weigh the estimated reward probability. When the discount factor, c, has the value 1.0, Algorithm 1 follows the standard BTS algorithm by selecting the action with the highest posterior sample, $\hat{\theta}_k$. For all $c < 1.0$, there is a weight, $1 - c$,

Assumption 1 (Environment): We assume that the map of the factory is known by the robot and there are no obstacles so the Euclidean distance between two locations is always a viable path for the robot to travel.

Assumption 2 (Robot): We assume the robot has the ability to get to each location without a fail. This means that if the robot receives a reward of 0 from an action, it is because the station did not have an update, not because the robot was not able to reach the station. We also assume that at the beginning of each data collection period, the robot starts in the middle of the factory map.

Assumption 3 (Stations): We assume that stations are always static and the robot knows the location of these stations. We also assume the robot is able to accurately estimate the distance between its current location and all the stations.

$$Q = c\,\hat{\theta}_k + (1 - c)(1 - \delta) \tag{4}$$

$$\delta_k = ||X_{station_k} - X_{robot}|| \tag{5}$$

Algorithm 1 Distance-Aware Bernoulli TS Algorithm

```
for $t = 1, 2, \ldots, T$ do
    for $k = 1, 2, \ldots, K$ do
        Sample $\hat{\theta_k}$ from $beta(\alpha_k, \beta_k)$
        Calculate $\delta_k$
    $x_t \leftarrow argmax_k(Q)$
    Apply action,$x_t$, and observe reward, $r_t$
    $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
    Update robot's location to chosen station
```

V. Results

Using Algorithm 1, we simulated the environment with both 4 and 8 stations randomly scattered in the map as shown in Figures 1 and 5, respectively. We ran the simulation for four values of the discount factor, $c = [1.0, 0.9, 0.8, 0.7]$, each for $T = 100$ and $T = 200$ and iterations with $K = 4$ and $K = 8$ actions for the 4 station factory and 8 station factory, respectively. The full code files can be found at `https://github.com/n-campbell/DABTS`.

VI. Discussion

As shown in Table I and Figure 2 from the simulation results with 4 stations, the traditional TS algorithm with $c = 1.0$ performs the best at estimating all the

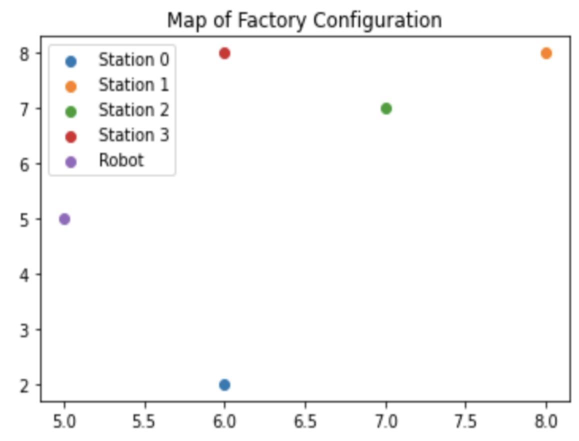

Fig. 1: Map of Factory with 4 Stations

TABLE I: Estimated and True Probabilities for 4 Stations

Probability	Station 0	Station 1	Station 2	Station 3
True P	0.9341	0.9769	0.4539	0.5368
Estimate with c = 1.0	0.9189	0.9649	0.6666	0.5000
Estimate with c = 0.9	0.5000	0.9484	0.6000	0.6666
Estimate with c = 0.8	0.5000	0.9696	0.3333	0.6666
Estimate with c = 0.7	0.5000	0.5000	0.5000	0.5148

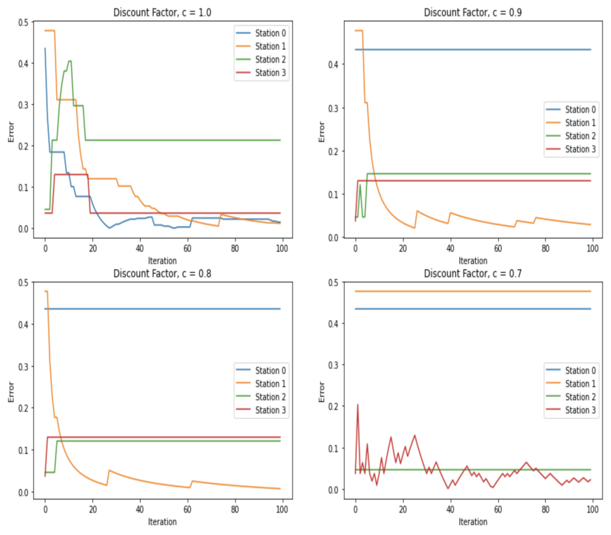

Fig. 2: Error Between True and Estimated Probability for 4 Stations

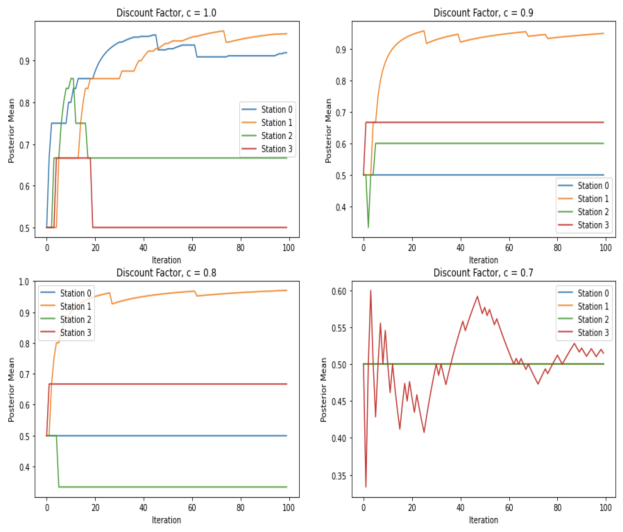

Fig. 3: Posterior Distribution Mean (Estimated Probability) for 4 Stations

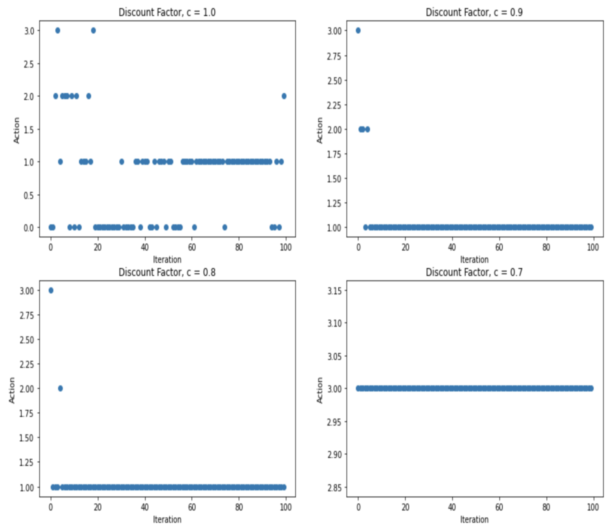

Fig. 4: Actions Taken By Robot for 4 Stations

true probabilities of each station and the performance consistently decreases as c decreases. This intuitively makes sense because as we decrease c, we are placing more importance on travel efficiency, therefore the robot does not have a lot of incentive to explore other stations that may be further away. Conversely, as we increase c, we are placing more importance on maximizing the expected reward from the stations, increasing incentive for exploration. This trend is observed in Figure 3 and 4 where we can see that with higher c values, the model is updating the estimate of the posterior means of more of the stations and has more variety in the actions. Once we get to $c = 0.7$, we can see in Figure 4 that the model only exploits one action, Station 4, for all iterations of

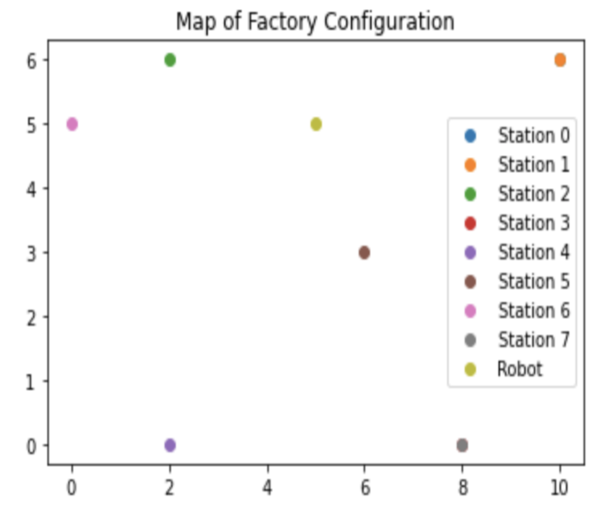

Fig. 5: Map of Factory with 8 Stations

TABLE II: Estimated and True Probabilities for Station 0 – Station 3

Probability	Station 0	Station 1	Station 2	Station 3
True P	0.9378	0.7604	0.2776	0.5682
Estimate with c = 1.0	0.8703	0.8421	0.3333	0.7142
Estimate with c = 0.9	0.5000	0.5000	0.3750	0.5000
Estimate with c = 0.8	0.5000	0.5000	0.3181	0.5000
Estimate with c = 0.7	0.5000	0.5000	0.5000	0.5000

TABLE III: Estimated and True Probabilities for Station 4 – Station 7

Probability	Station 4	Station 5	Station 6	Station 7
True P	0.8678	0.1838	0.4167	0.04079
Estimate with c = 1.0	0.8333	0.2500	0.2857	0.2000
Estimate with c = 0.9	0.5000	0.3333	0.5268	0.5000
Estimate with c = 0.8	0.5000	0.5000	0.4339	0.5000
Estimate with c = 0.7	0.5000	0.1890	0.5000	0.5000

the simulation even though that station does not have the highest true probability of reward.

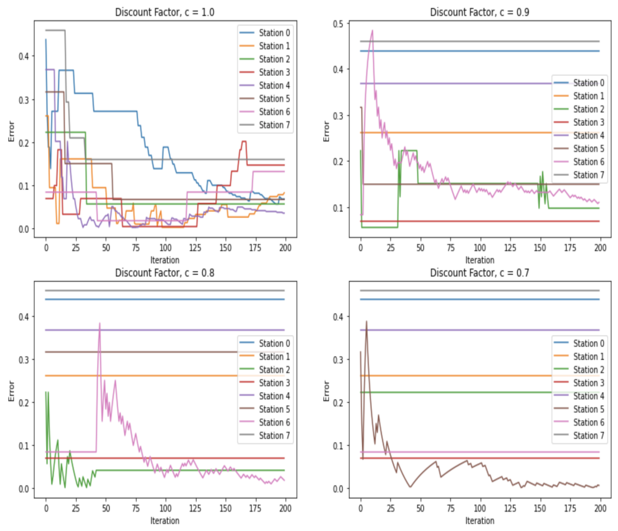

Fig. 6: Error Between True and Estimated Probability for 8 Stations

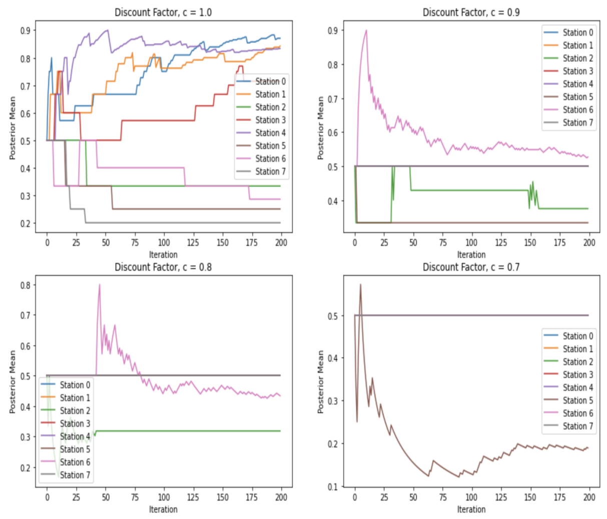

Fig. 7: Posterior Distribution Mean (Estimated Probability) for 8 Stations

Although $c = 1.0$ has the most accurate estimated probabilities when averaging all the stations, we can see that with $c = 0.7$ the most explored station, Station 3, has a more accurate estimate than it did for $c = 1.0$. This is not useful if the goal was the find the highest rewarding station. It is likely that the model consistently chose this station because it was closest to the the robot's initial location and since after each action the robot is moved to the selected station the selected station will always be the closest.

We see similar trends in Figures 6 – 8 for the simulation with 8 stations as we did with 4 stations. The major notable difference when doubling the number

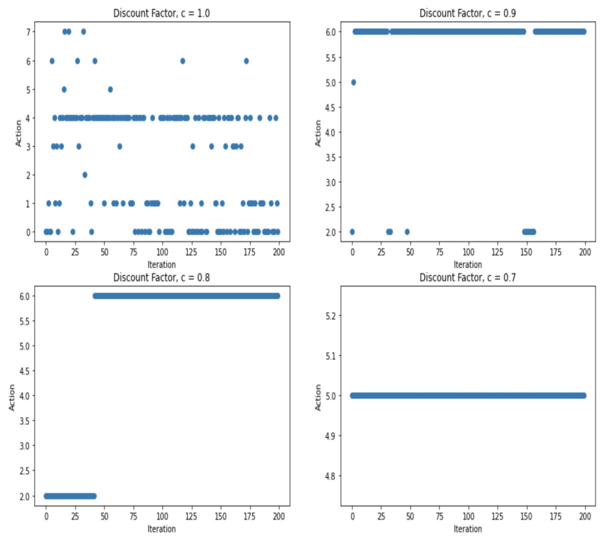

Fig. 8: Actions Taken By Robot for 8 Stations

of stations can be seen in Tables II and III. Even with $c = 1.0$, the model is unable to accurately estimate the true probability of each station. It explores, and therefore estimates better for some stations more than others. For example, Table III shows that Station 7 has an error of 0.159 whereas Station 4 has an error of 0.0345. Another thing to note is that Station 3 in Table II has an error of 0.146 which is similar to the error of Station 7 even though it has a much higher true probability which the model would have learned if it explored that action more.

VII. Conclusion

Through running sensitivity analysis on the c parameter, we observed that increasing c decreases exploration and accuracy of the reward probabilities. Similarly, we observed through sensitivity analysis on the number of stations that increasing the number of stations decreases exploration and accuracy of the reward probabilities.

An issue we came across in the results is that when the c is lower, it tends to get stuck in one station. As previously discussed, this is most likely because more importance is put on reducing travel cost and once the robot visits a station, the cheapest decision is to stay there. One way to combat this for future work is to put more weight on choosing other stations than the one it is currently visiting.

The BTS algorithm does well with accurately predicting the reward probability of the station that it explores the most, but if the goal is to accurately predict the probabilities of all the stations, another method might be more ideal. Instead, the algorithm is a good fit if you want to exploit the station with the most estimated reward.

References

[1] A. Slivkins, "Introduction to multi-armed bandits," 2019.
[2] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on thompson sampling," 2017.
[3] X. Chen and J. Liu, "Multi-robot object finding using thompson sampling," in *2022 International Symposium on Control Engineering and Robotics (ISCER)*, (Los Alamitos, CA, USA), pp. 105–110, IEEE Computer Society, feb 2022.
[4] Z. Wang, Y. Gao, Q. Xiao, and Q. Zhang, "Application of Thompson sampling on multi-target search," in *2nd International Conference on Applied Mathematics, Modelling, and Intelligent Computing (CAMMIC 2022)* (H. M. Srivastava and C.-H. Chen, eds.), vol. 12259, p. 122591Q, International Society for Optics and Photonics, SPIE, 2022.
[5] J. Chen and P. Dames, "Active multi-target search using distributed thompson sampling." 2022.
[6] J. Chen and S. Park, "Bernoulli thompson sampling-based target search algorithm for mobile robots." 2022.
[7] M. Korein and M. Veloso, "Multi-armed bandit algorithms for spare time planning of a mobile service robot," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, (Richland, SC), p. 2195–2197, International Foundation for Autonomous Agents and Multiagent Systems, 2018.