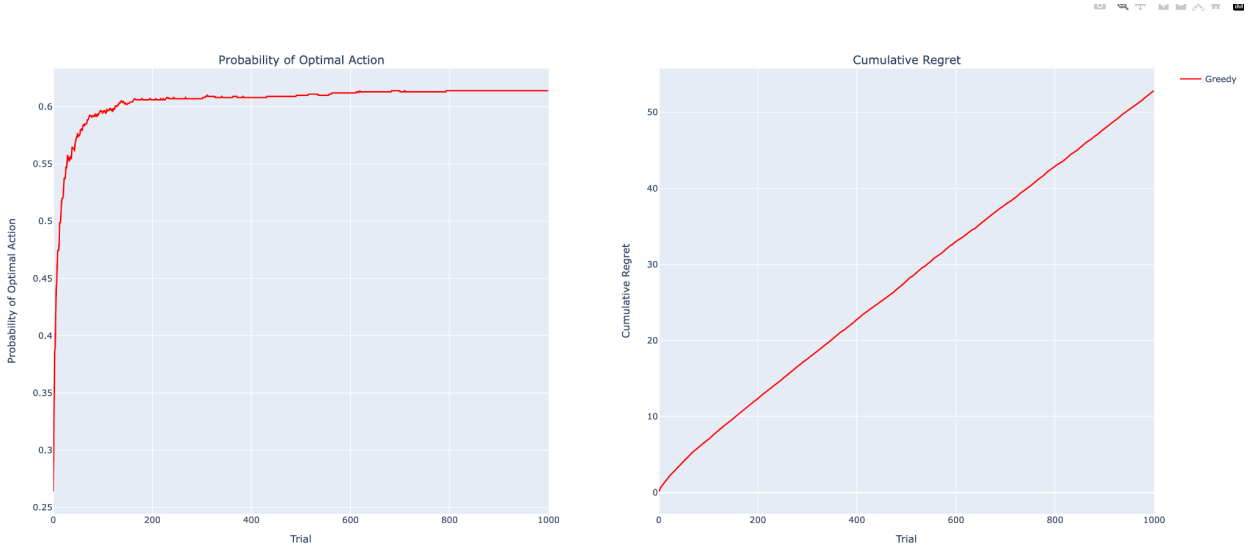
Assignment 1:

Problem One: Greedy Agent

a. Find the ad with the highest sampled Q value which consists of the number of clicks per interaction. If the greedy agent has multiple ads with the same Q value it would select randomly which causes optimality errors. This ad agent will perform the fastest however the least optimal. For example, if our agent were shown an ad 100 times and clicked it 30/100 times while another ad that was shown 20 times and clicked 15/20 it would select the second ad where it had more interactions per shown.

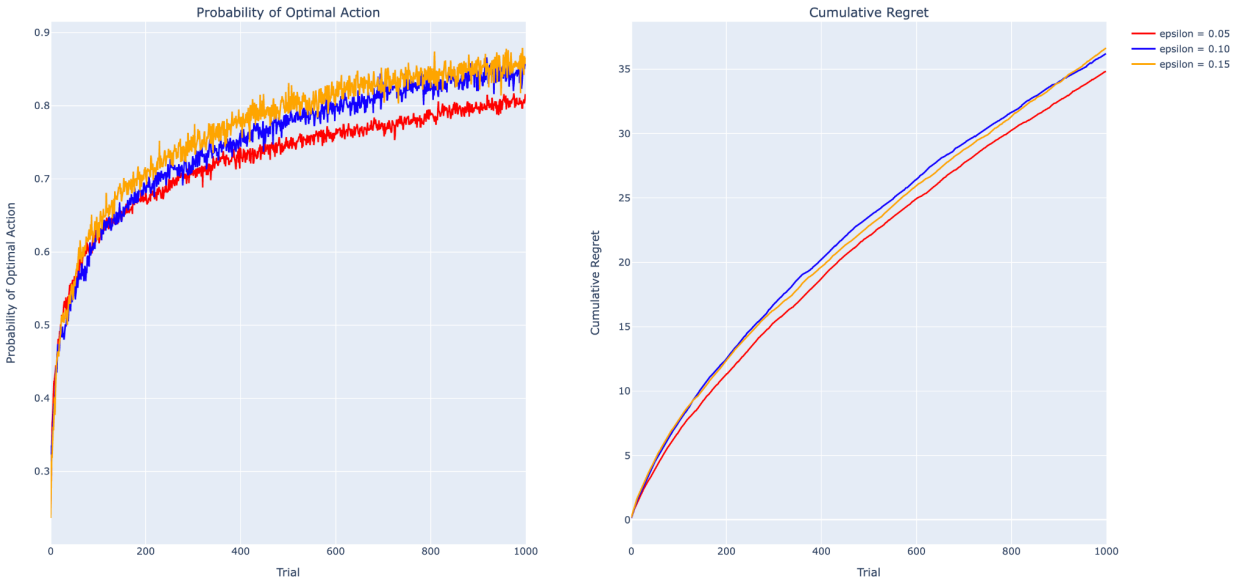
2. Sim Results: run a simulation script only through Greedy Gent and modify the agent list in the simulation configurations capture the resulting group and add to the report



Problem Two:

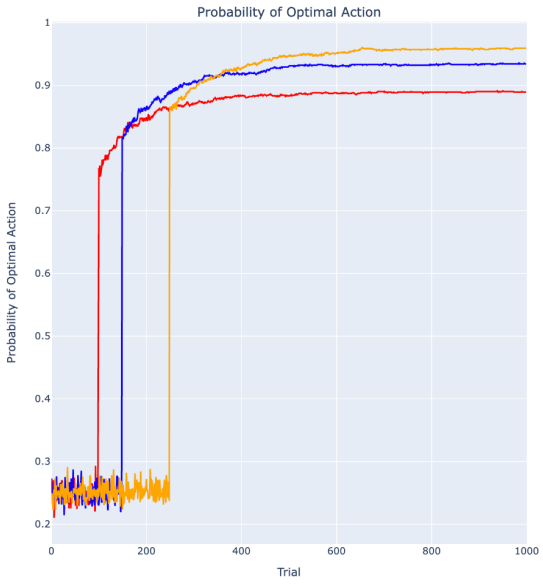
1. **Expectations:** in a sentence or two, describe how you expect this agent to perform with values of $\epsilon \in \{0.05, 0.10, 0.15\}$
   a. I expect the agent with the highest epsilon to have the highest probability of being optimal while the agent with the smallest epsilon will be the least optimal. This is because it will spend more time exploring but still balance that with time exploiting.

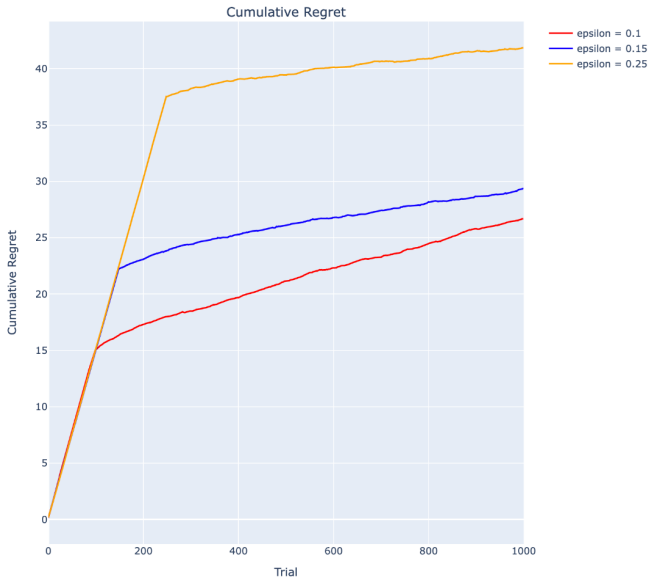
2. **Sim Results:** run the simulation script with 3 separate versions of the Epsilon_Greedy_Agent, parameterized by the 3 separate values of ϵ mentioned above, capture the resulting graph, and add it to your report



Problem Three:

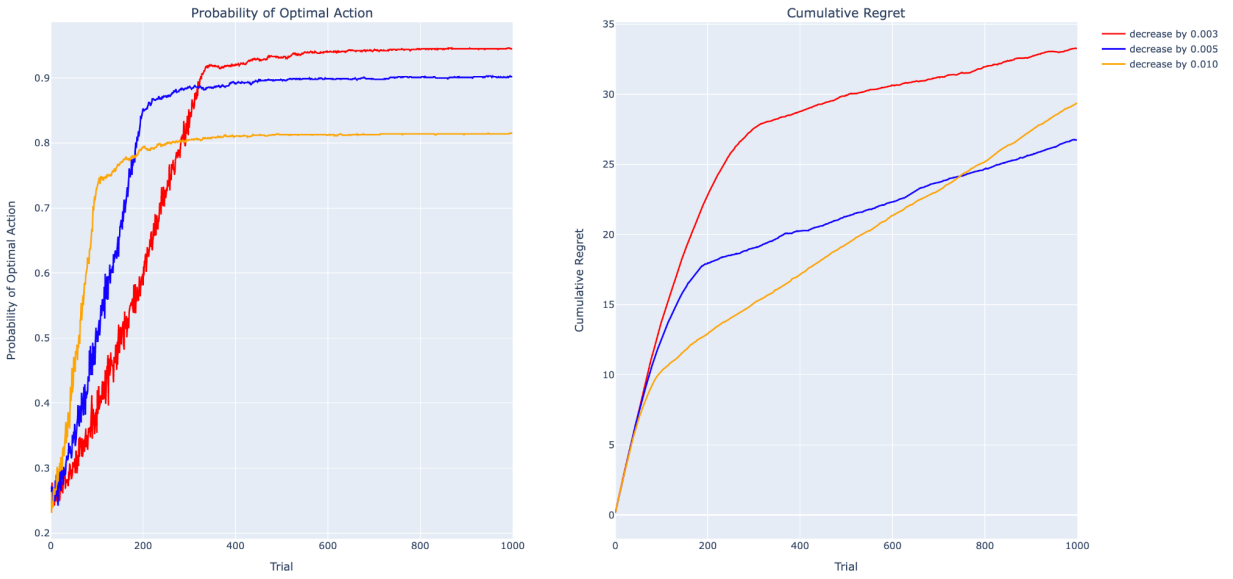
1. **Expectations**: in a sentence or two, describe how you used knowledge of a finite-time horizon T to choose and then test several values of ϵ
    a. For epsilon-first, considering $T = 1000$, we choose to test $\epsilon = .1, .15, .25, .33$ to explore for 100, 150, 250, and 330 trials respectively. Proportionally, we thought these would be adequate sample sizes to then exploit from.
2. **Sim Results:** run the simulation script with 3 separate versions of the Epsilon_First_Agent, parameterized by the 3 separate values of ϵ that you mentioned above, capture the resulting graph, and add it to your report.





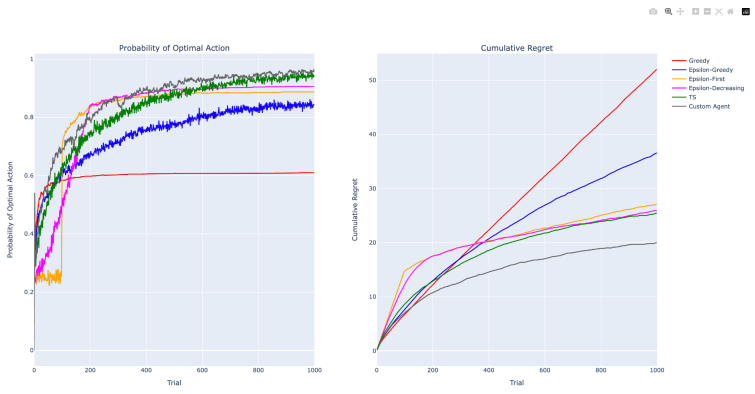
Problem Four:

1. **Expectations:** in a sentence or two, describe how you used simulated annealing to derive a cooling schedule for the decrease of ϵ over time (feel free to experiment here, and remark on your approach / findings).
    a. We simulated annealing by decreasing ϵ by 0.005 during every iteration. We decided that 0.005 was a good value to decrement after trying multiple other numbers. We found that the smaller the number, the higher the optimal probability, however, it took more trials to reach this level of optimality.
2. **Sim Results:** run the simulation script with 3 separate versions of the Epsilon_Decreasing_Agent, parameterized by the 3 separate cooling schedules that you mentioned above (these might be different by starting values of ϵ, rate of decrease, etc.), capture the resulting graph, and add it to your report.

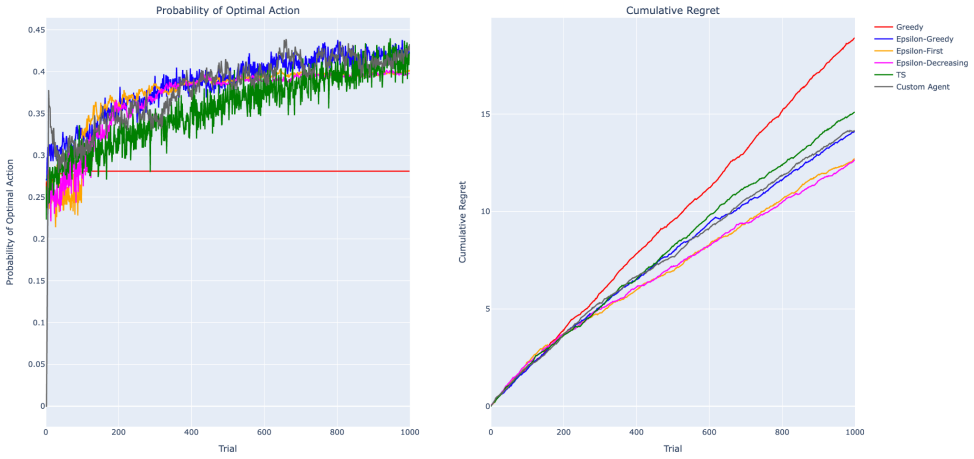


Problem Five: The Thompson Sampling Player

1. **Summary:** The Thompson Sampling Bandit player implements a "self-correcting" agent by sorting through the "exploitation vs. exploration" by taking a mass sample of the wins and losses built from our agent's history. We implemented this by making a list that organizes and samples our wins (interactions) , losses (non-interactions with a view) and then selecting the highest sampled distribution.
   a. Encoding wins and losses = np.random.beta(wins,losses) = np.random.beta(interaction, non-interaction with a view) then collect all the information from the sample then pick the largest
2. **Sim Results:**Record the graph and results in your report.



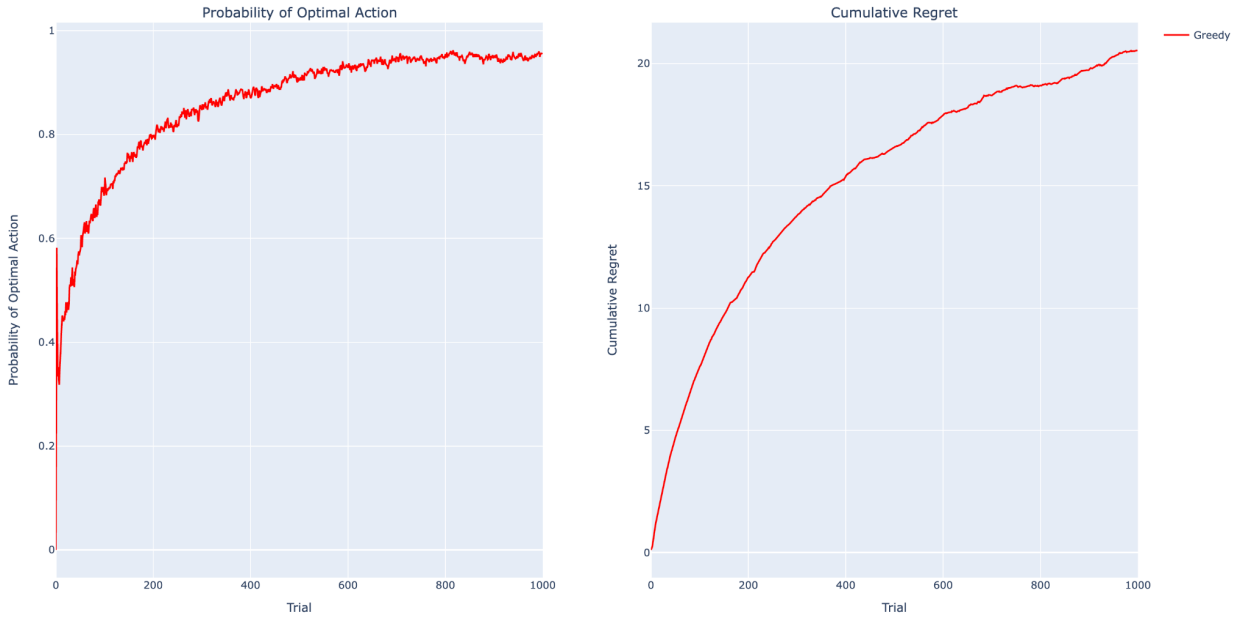
3. Compare: **Reflect:** When values are close together it doesn't do much better however when the values are spread out it does perform better.



Problem Six: Custom Agent - Upper-bound Confidence

1. For UCB, we found the sqrt of the log of the current number of iterations then dicided that by the amount of times action a had been chosen. We multiplied that by c which was our confidence variable which we kept relatively low. We then added the current q value (clicks/total choices) to that product.

2.



Problem seven: Exploration

One variant of a MAB problem is the adversarial bandit problem. In this problem, one agent chooses an arm while another, aka the adversary, chooses the reward for that agent's arm choice. It is more of a specified bandit problem rather than a generalized one. One strategy that approaches this problem is the EXP3 algorithm. This algorithm chooses an arm randomly based on a probability assigned to them. It prefers arms with higher weights in order to exploit before exploring more. The weights are then updated through the rewards, allowing an exponential increase of weight on the best arms. An example that fits within this problem is the iterated prisoner's dilemma. EXP3 theoretically approaches this problem by more heavily weighting combinations in which the two adversaries would cooperate instead of defect.