CS/STAT 184(0) Final Report

Christopher Lapop Salazar
Eric Peterson
Michelle Nhung Le
clapopsalazar@college.harvard.edu
peterson@g.harvard.edu
michellele@college.harvard.edu

Abstract

We developed a reinforcement learning model for hobby card collecting, focusing on optimizing card pulls and completing specific collections within a budget. Using modified Thompson Sampling and Proximal Policy Optimization, our approaches achieve sublinear regret and efficient policy learning. The results show robust performance for both diverse and rare card collections.

1 Project Phases

1.1 Phase 1: Unknown pack pull rates, minimize regret with different pack choices

In Phase 1, we adapt a Bernoulli Bandit problem to hobby card-collecting, where each pack pull generates cards of varying market values. Our goal is to maximize the value of collected cards by learning optimal pull strategies using a reinforcement learning agent.

We extend Thompson Sampling for multinomial outcomes, using Dirichlet priors to model pull probabilities. The algorithm iteratively updates these priors based on observed rewards.

1.2 Phase 2: Maximize probability of completing a specific deck of cards within set budget

Phase 2 introduces selling individual cards at market value, allowing the agent to extend the time horizon. This marks a change from an MAB problem towards an MDP problem, as the current budget and cards drawn so far are used to make decisions. The state space encodes the cards that have been pulled so far and the remaining budget for buying cards. The action space consists of the arms to pull from Phases 1-2, and additionally the option to sell cards to a merchant to recoup money for more card packs.

Actions that result in acquiring needed cards could generate rewards, while actions that don't result in acquiring needed cards would generate either no reward or smaller rewards if they give high-value cards that can be sold to buy more card packs. The distributions of cards granted by the different packs are encoded in transition probabilities of the MDP. The state space consists of all possible combinations of cards, which is quite high-dimensional, so we will look into different exploration and policy gradient methods for learning the MDP and finding an optimal policy. An additional ability to purchase cards at market value price is being considered, which would allow guaranteed (but possibly overly costly) steps towards completing the specific deck of cards.

We can optimize card collection utility by applying Proximal Policy Optimization. Environment settings are: **State Space:** $s_t = [\mathbf{x}_t; b_t]$, where:

$\mathbf{x}_t \in \mathbb{N}^N$ is current collection (count of each card) $b_t \in \mathbb{R}$ is current budget (N is total number of unique cards)

Action Space: $A = A_{\text{pack}} \cup A_{\text{trade}}$ where:

- Buy pack: $A_{\text{pack}} = a_{\text{pack}}(k) \mid k \in [K]$ for K different pack types
- Trade cards: $A_{\text{trade}} = a_{\text{buy}}(i); a_{\text{sell}}(i) \mid i \in [N]$ for buying/selling each card

Reward Function:

$$R(s_t; a_t; s_{t+1}) = R_{\text{collection}} + R_{\text{budget}}$$

where:

- $R_{\text{collection}} = 10.0 \quad \text{completion}$
- $R_{\text{budget}} = \begin{cases} 0 & \text{if} \quad b_t > 0 \\ 0.1 \quad b_t & \text{if} \quad b_t < 0 \end{cases}$
- completion is the change in number of needed cards collected
- b_t is the change in budget

This reward function effectively measures the progress made towards completing the collection by considering the reduction in the remaining cards needed after updating the collection with the drawn cards and cares less about cost because cost is scaled by 0.1.

2 Comparison to other works

The second phase of our project addresses the challenge of collecting a series of unique objects, each with an independent and unchanging probability of being obtained. This setup serves as a variation of the classical coupon collector problem. Our "completionist" test case functions in the same manner, with the objective of pulling a minimum of one of each unique trading card. When comparing our implementation with the report by Emma Brunskill and Lihong Li, the report develops an algorithm FORCEDEXP, designed for Lifelong Multitask Reinforcement Learning. FORCEDEXP tackles the problem by emphasizing the design of an exploration strategy that can identify and adapt to new tasks. In contrast, our work with a Bayesian policy model and PPO applies single-task reinforcement learning methods, prioritizing a specific task model. By refining and optimizing an existing policy, it focuses on the completion of a single predefined task - our specified collection.

3 Phase 1

3.1 Environment

We wrote an environment class called `PACKPOOL` from scratch, which will also serve as a core for the environment in the later phases of our project. The class contains `num_packs` "packs" objects, which correspond to the card packs with randomly-sampled cards. In order to make the environment correspond to some realistic card-collecting scenarios, there are 3 sub-categories of cards ("common," "uncommon," and "foil") which each have different ranges of values. The class stores the lists of cards that each pack can possibili contain, the probabilities of drawing them, and each of their individual values. The `PACKPOOL` class also includes methods to open packs (effectively, sample according to the card probabilities within each respective pack) and calculate the values of sets of cards.

3.2 Algorithm

The problem of trying to maximize the value of collected cards by opening packs with a priori unknown card distributions is conceptually similar to the multi-armed bandit problem that we studied in class. The most effective approach we learned for this kind of problem was Thompson sampling, but in order to apply this algorithm to our problem we need to extend it to deal with an important distinction from Bernoulli bandits: the arms now output a vector drawn from a multinomial distribution, with the reward being a weighted sum of the categories drawn (here, the categories are the cards and the weights are their individual values).

Since the essence of the Thompson sampling algorithm is to maintain Bayesian priors on the arm distributions and sample them according to which ones seem most promising given all existing information, this suggests that we can straightforwardly adapt it if we have a conjugate prior for the multinomial distribution and some way of ranking arms to compute the $\arg\max_k$ over packs. It turns out that the Dirichlet distribution Dir(α_k) is a conjugate prior of Mult(K, p) and has a similar Bayesian update rule to that of the Beta distribution, which is just to increment α_k^j by one each time pack k outputs card j.

Finally, we define the reward of a pack pull n_t as the total value of all the cards drawn, i.e. $r_t := n_t \cdot v$, where $v \in \mathbb{R}^J$ is a vector containing all of the individual card values. Then, the regret is just the cumulative sum of differences between the expected value of the pulled packs and the pack with the highest expected value.

In summary, our new algorithm can be written as:

Algorithm 1 Modified Thompson Sampling for Packs

Require: Number of packs K, number of cards J, horizon $T \geq K$

Initialize $\alpha_k^{(j)} = 1 \in \mathbb{R}^{J,K}$

for $t = 0, 1, \ldots, T-1$ **do**

  for $k = 1, 2, \ldots, K$ **do**

    Sample $\hat{\theta}_k \sim$ Dir(α_k)

    Compute reward map $\hat{r}_k = r(\hat{\theta}_k)$, where r is the function that maps the hand to its value, $\mathbb{R}^J \to \mathbb{R}$

  end for

  Open pack $a_t = \arg\max_{k \in \{1,\ldots,K\}} \hat{r}_k$, where ties are broken uniformly at random

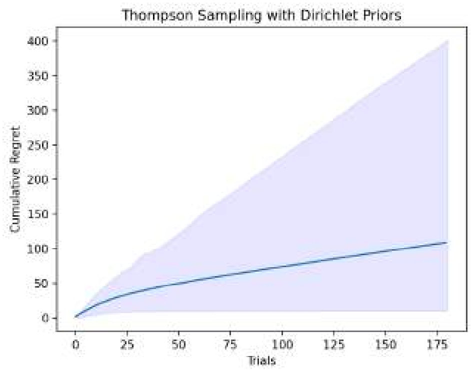
  Update the belief for pack a_t using the drawn cards n_t:

$$(\alpha_{a_t}^{(1)}, \alpha_{a_t}^{(2)}, \ldots, \alpha_{a_t}^{(J)}) \leftarrow (\alpha_{a_t}^{(1)} + n_t^{(1)}, \alpha_{a_t}^{(2)} + n_t^{(2)}, \ldots, \alpha_{a_t}^{(J)} + n_t^{(J)})$$

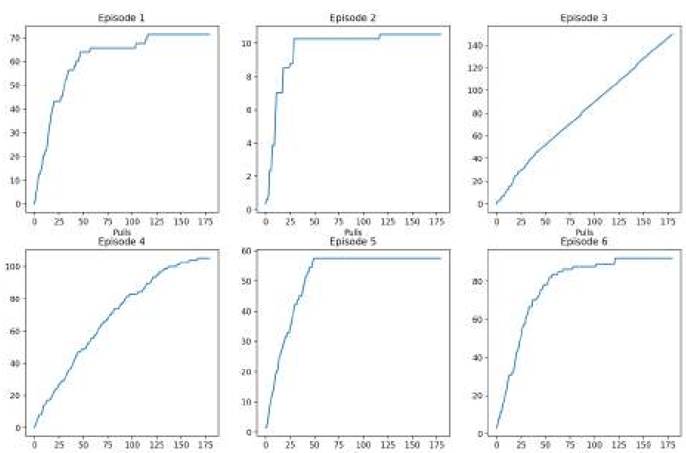
end for

3.3 Data generation

We ran our algorithm on the `PACKPOOL` class for 100 episodes and a horizon of $T = 180$ pack pulls. The mean regret with 95 percent confidence intervals is plotted below:



The algorithm is evidently able to achieve sublinear regret, although with a fairly large variance. Plotting the first 6 episodes suggests that algorithm's behavior is fairly "streaky:" it is often able to quickly identify the optimal pack and almost completely stops exploring (Episodes 1, 2, 5 and 6) but occasionally stops exploring with a high-confidence prior on the incorrect optimal pack (Episode 3):



4 Phase 2

4.1 Environment

In approaching Phase 2, the environment makes use of the `PACKPOOL` class with modifications to the open_pack_list function to provide the specific cards produced from pack openings rather than only producing the card values. Most notably, Phase 2 involves the creation of a class `DefinedCollection` whose reward function is built to provide a full reward in cases where a pulled card is found and then removed in a living `target` list and partial rewards based on card values above a certain threshold. The number of pack openings will also be modified, based on the subtraction of a consistent pack price from a budget, allowing cards to contribute toward opening an additional pack.

We set the variable `cost_per_pack` to be sufficiently high that it would ensure that our MDP is finite (if `cost_per_pack` were lower than the expected value of any of the packs, then it would be theoretically possible for the budget to grow indefinitely without any of the `target` cards being collected, resulting in an infinite-horizon MDP). This was done so as not to preclude some of the dynamic programming-based methods that we learned in class (such as UCB-VI), though in the end we didn't take advantage of dynamic programming (we discuss this further in the Conclusions section).

While it is easy to write down the states and actions of this MDP, it is very high-dimensional, with the state space consisting of all possible combinations of budget and drawn cards within the constraints of the initial budget and target collection. The transition probabilities $P(s'|s, a)$, related to the pack distributions, are also unknown at the beginning of the algorithm execution, so it is necessary to explore the MDP to converge towards an optimal policy. In the next section, we describe how we adapt the Bayesian method of Phase 1 to overcome these challenges.

4.2 Algorithm

The problem of maximizing probability to complete the specified collection initially led to us considering Proximal Policy Evaluation in the Project Proposal. However, UCB-VI is being highly considered due to how sparse the rewards provided from the packs may be, allowing UCB-VI's exploration to shine in making sure card packs are not only chosen with confidence but are selected without neglecting the potential of less explored packs.

In addition to the reward function described in Section 1.2, we also experimented with an exploration bonus that would encourage exploration. Rather than base the exploration bonus on a Hoeffding bound (as was done in the UCB-VI algorithm covered in class), we formulated a new bonus based on our generalization of Thompson sampling from Phase 2 but still based on the principle of "optimization in the face of uncertainty" that underlies the UCB algorithm. In particular, we wanted the policy to give extra weight to packs that still may have a good chance to yield the card that is still most needed at time t, c'_t = `argmax(target)`, within the degree of confidence granted by the prior pack openings. To do so, we maintained a set of Bayesian priors on the pack distributions that we updated after each pack opening as in Phase 1. Then, we calculated the upper credible interval of $p(c'_t)$ for each pack, weighted by a hyperparameter C that determines how greedy the policy should be. A higher value of C encourages exploration; typically a value of between 10 and 20 was needed to make the reward bonus comparable to the baseline rewards of the environment, as the large number of cards in environment led to individual card draw probabilities that were quite low.

Mathematically, the marginal distribution over the Dirichlet priors for c'_t winds up being $p^{(a)}_{c'_t} \sim$ Beta(α', β') where in terms of the vector α parameterizing the Dirichlet distribution

$$\alpha' = \alpha_{c'_t}$$
$$\beta' = \sum_{c \neq c'_t} \alpha_c$$

Then, the exploration bonus assigned to each pack a is given by $b_t(s, a) := C \cdot U_a$, where U_a is calculated from

$$P(p^{(a)}_{c'_t} \leq U_a) := 1 - \delta \tag{1}$$

We set $\delta = 0.95$ and didn't use it as a separate hyperparameter.

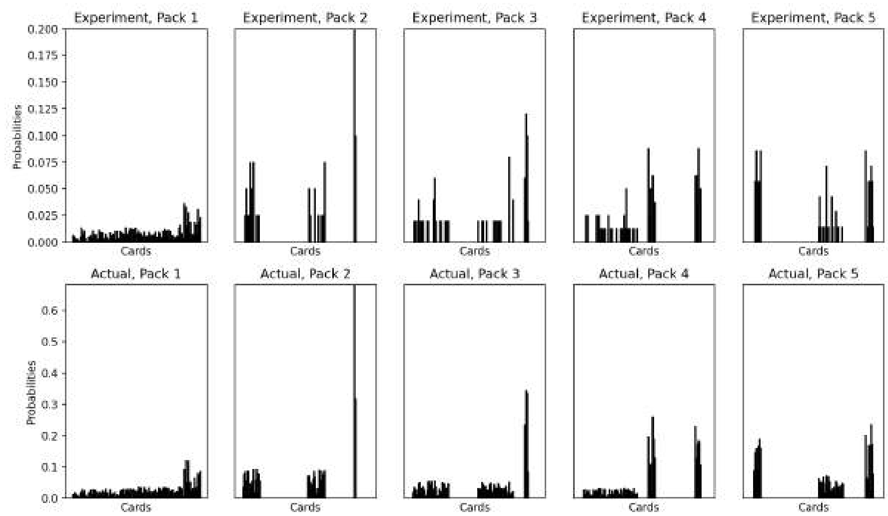
We also used the Bayesian idea to parameterize our policy. As an aside, one of the themes of the course was the generality of MDPs, given that past history can be incorporated into the current state to make the environment Markovian again. Incorporating all information about the past transitions into a Bayesian prior is a particularly memory-efficient way of doing so. It also provides one way of overcoming the large dimensionality of our state space S in designing policies $\pi(s)$, by parameterizing the policies in terms of the priors α. We used a policy based on the expected reward $r(s, a)$ of pulling pack a in state s, which represents all information about of the cards collected so far as well as the current budget. Including the reward bonus discussed previously, our policy is

$$\pi_{\alpha, C}(s) := \arg\max_a \left[E_{\hat{P}(\cdot|s,a)=\mathrm{Dir}(\alpha_a)}[r(s, a)] + C \cdot b_t(s, a) \right] \tag{2}$$

The two terms in this policy complement each other – the first prioritizes greedy actions, while the second encourages exploration.

4.3 Results

We designed the card packs to guarantee that all cards existed in at least one pack. In particular, any card could be drawn from `Pack 1` with low probability, while other packs had probabilities more concentrated around specific subsets of cards. In each case, we tracked all cards collected throughout the episode. Both of these points are illustrated in the following figure, where the empirical pack distributions after running the algorithm once are plotted above their actual distributions:



We measured the performance of our algorithm in several different ways. First, we measured the cumulative reward throughout the trajectory. We plotted this alongside the cumulative budget and fraction of `target` cards collected. Additionally, since we are interested in measuring how well the algorithm learns the MDP, we plotted a measure of the statistical distance between the Bayesian prior distributions based on the opened pack data and the "true" distributions of the packs. We initially considered using the KL-divergence (discussed in class in the context of PG methods) as a statistical distance measure, but eventually opted for a simpler metric that directly compares the prior parameters α to the true pack parameters p, as we expect that $\bar{\alpha} \to p$ asymptotically as more samples are collected. Mathematically, our distance measure is written

$$d := |\bar{\alpha} - p| \tag{3}$$

where $\bar{\alpha}_i := \alpha_i / \sum_j \alpha_j$. We computed how d evolved for each pack separately throughout the trajectory.

We tested our algorithm on two different `target` collections. In the first case, `target` consisted of one of each possible card (`completionist`). This test was somewhat easy, and the policy sampled more-or-less evenly from each pack even for a small value of C: