AeroAstro Work-Life Balance: How to Maximize Your AA 228 Quiz Scores

Alexandra Haraszti, Jusin Kao, and Yasmina Elmore
AA 228 Final Project Update

Simulated learning strategies are increasingly utilized in educational research to address challenges associated with optimizing academic performance under time constraints. In this project, we model the study behavior of students in Stanford's AA 228 course as a Markov Decision Process (MDP), focusing on how they allocate their time across chapters to prepare for critical quizzes. Using Monte Carlo Tree Search and Q-Learning, we evaluate the optimal study policies and their implications on academic success, discussing the tradeoff between exploration and exploitation in our exploration strategies and their resulting policies.

I. Introduction

EXCELLING in a challenging course like AA 228, "Decision Making under Uncertainty," is a top priority for Stanford students, especially given the heavy emphasis on three high-stakes quizzes throughout the term. These quizzes, strategically spaced across the 10-week course timeline, demand not only a strong grasp of approximately 20 chapters of material but also a well-structured and efficient approach to studying. Yet, the urgency to perform well on these quizzes often leaves students overwhelmed, leading to significant stress and suboptimal preparation strategies.

The crux of the problem lies in how students allocate their limited time to study while balancing other commitments and maintaining their well-being. Traditional methods of cramming or evenly distributing effort across topics often fail to account for the dynamic nature of learning. Without clear and evidence-based guidance, many students fall into ineffective patterns of preparation, leading to underperformance. To address this issue, this project models AA 228 as a sequential decision-making problem using a Markov Decision Process (MDP). Each day in the term is treated as a decision point where a student must choose how to allocate their study effort across the chapters. The objective is to find an optimal study policy that maximizes the probability of mastering the material necessary for success on the quizzes while also considering trade-offs such as preserving free time for recovery.

The model defines competency for each chapter as a binary state (mastered or not mastered) and incorporates probabilistic transitions that reflect real-world challenges. For instance, attempting to study too many chapters in one day reduces the probability of successfully mastering any single chapter, penalizing excessive workloads. By simulating different policies and evaluating their outcomes, we aim to provide actionable insights for students struggling to navigate the demanding timeline of AA 228.

A. Prior Work

1. Initial Inspiration

Our project builds on the work of Chen et al. [1], which modeled classrooms as Partially Observable Markov Decision Processes (POMDPs) to explore optimal policies for time allocation between students and teachers. The study effectively utilized simulated educational data to address challenges such as privacy concerns and logistical constraints inherent in real-world data collection. While the primary focus of Chen et al. was on the interactions between students and teachers, their methodology of using POMDPs inspired us to apply a similar approach to optimize study strategies for AA 228. The critique of quantitative over-reliance in their work also provided valuable insights into balancing model assumptions with real-world complexities.

2. Simulated Data and Educational Optimization

Simulated data has gained traction in educational research as a solution to the cost and accessibility barriers associated with real-world data collection. Rafferty et al. [2] applied POMDPs to teaching strategies, demonstrating

how adaptive methods could improve students' knowledge acquisition by predicting their learning states and tailoring lesson plans accordingly. Similarly, Chung et al. [3] introduced a reinforcement learning framework to optimize spaced repetition schedules, improving long-term knowledge retention through data-driven review intervals. These approaches illustrate the potential of simulated environments to provide actionable insights into learning dynamics, an essential aspect of our MDP-based study optimization model.

3. Reinforcement Learning in Education

The application of reinforcement learning (RL) in educational systems has shown great promise for personalized and adaptive learning. Matsuda et al. [4] presented RLTutor, an adaptive system that employs RL to optimize teaching strategies by modeling student behavior. This system emphasized the balance between exploration and exploitation, an approach mirrored in our use of Q-learning to navigate the trade-offs in study time allocation. More recently, Ye et al. [5] developed a Deep Q-Network framework for optimizing study schedules, providing a sophisticated example of RL's potential to enhance educational outcomes.

4. Broader Applications of MDPs

Markov Decision Processes have been widely adopted in educational settings to model sequential decision-making under uncertainty. Ndukwe et al. [6] utilized simulation-based models to predict teaching outcomes, highlighting how controlled environments can inform practical applications. Shenfeld et al. [7] extended this work with teacher-guided reinforcement learning algorithms to create adaptive policies for diverse learning environments. These efforts align with our goal of employing MDPs to optimize learning strategies in the context of a high-stakes academic setting like AA 228.

5. Limitations and Challenges

While simulated data provides an efficient means to model educational scenarios, it has limitations. Gillborn et al. [8] critiqued the use of simulated data, arguing that such models often fail to capture the complexities and nuances of real-world educational settings. They also raised concerns about the potential biases embedded within quantitative frameworks, which could inadvertently shape outcomes in undesirable ways. These limitations are critical to consider as we design and interpret our MDP-based model, ensuring that it remains robust yet adaptable to the challenges students face in practice

II. Environment

We have set up a simplified AA 228 model with the following behavior:

Horizon: We have a finite horizon as we discretize the term into the approximately 60 days between start of class (Sept. 24) and Quiz 3 (Nov. 22). True to the current course schedule, Quiz 1 is on day 17, Quiz 2 is on day 38, and Quiz 3 is on day 60.

Observation: There are approximately 20 chapters in the textbook that an AA 228 student is responsible for understanding. Students may come in with some background knowledge on some chapters which will be accounted for by randomized initialization. For computational manageability of the large state and action spaces, we have reduced this to 10 chapters for this project.

State: For each chapter, a student either understands the chapter (comp = 1) or does not (comp = 0). We call this a competency score and keep it binary for now. Thus there are $2^{10} = 1024$ possible states each day.

Action: Each day, a student can either study a chapter or not. They can choose to study as many chapters as they wish (though too many chapters studied on one day will be less likely to increase competency). Choosing not to study is rewarded with free time. For each of the 10 chapters, the student either studies or does not so there are $60 * 2^{10} = 61440$ possible actions over the 60 days. However, we operate under the assumption that a student will not study material that has not yet been covered in lecture. Therefore, the action space is smaller early in term and a new chapter is added to the

study-able list every 6 days.

State Transition: Each day, for every chapter, there are 4 options:

1) Studied and comp = 1: The student will keep comp = 1 with probability 0.9.
2) Studied and comp = 0: the student will gain comp = 1 with probability

$$p(comp = 1) = \frac{1}{\text{number of chapters studied}}.$$

Thus we penalize studying too many chapters at once.

3) Not studied and comp = 1: the student loses understanding with probability

$$p(comp = 0) = 0.2.$$

4) Not studied and comp = 0: no change.

Reward: There are two sources of reward: free time, and quiz scores. For each action taken, studying receives a free time reward of -1 and not studying receives a reward of 0. This reward is received daily. Assuming each quiz has 6 questions and each question pertains to a single chapter, we randomly select 6 chapters from the course so far that will be tested. The quiz score is then given by

$$R_{quiz}(s_t, a) = \alpha \sum_{i=1}^{6} \text{comp}_i$$

where i is the randomly chosen chapter. This reward is given on quiz day for each of the three quizzes. We chose a scaling of $\alpha = 50$ arbitrarily to weight quiz grades higher than free time rewards

III. Approach and Implementation

We implement Monte Carlo Tree Search (MCTS) and Q-learning algorithms to find the optimal policy that maximizes quiz scores and free time under our problem definition above. We chose MCTS because we have a sequential decision-making process where we must make a decision on what chapter to study each day. We also know the explicit state transition and reward models. Then each day we can simulate the potential reward from each possible action and choose the best possible action. We compare this model-based method with the model-free base Q-learning. Q-learning works easily out-of-the-box but requires a more careful balance of exploration vs. exploitation, especially with our large state and action spaces. We use the same transition and reward functions for both algorithms for ease of comparison.

Since our states and actions are day-dependent, we choose to initialize our possible state-space and action-space for each day. On day one, we have two possible actions: study chapter 1 or not. On day 6, we have four possible actions: study chapters 1 and 2 or not. The pattern continues, until we have $2^{10} = 1024$ possible actions on day 60. Similarly, we define a 3D matrix of Q-values where we define $Q_i(s, a)$ to be the action value function specific to day i.

We have a finite horizon problem so we will not use reward discounting ($\gamma = 1$) and will simulate all 60 days to the end of the term with each algorithm. This is especially important because our most important rewards are only received three times a term and we do not want to overly bias towards the free-time rewards on individual days if we do not simulate to enough depth.

A. Monte Carlo Tree Search

Monte Carlo Tree Search is a learning algorithm that estimates expected rewards by averaging outcomes from multiple episodes. Unlike Q-learning, which updates estimates after each step, Monte Carlo methods wait until an episode concludes before calculating and updating values. This makes it effective for modeling long-term dependencies, though, as we learned, it can be computationally intensive for environments with long or complex episodes.

The Monte Carlo Tree Search method builds a decision tree incrementally where the nodes are student understanding of each chapter s, and the edges are the actions a of which chapter to study that transition the model from one state to another. For each day of the term, we walk through four steps:

- **The exploration**: Our algorithm starts at the root node and explores the tree by following an exploration strategy that selects the action maximizing the Upper Confidence Bound (**UCB**) *exploration heuristic*. For each day i, the exploration is defined by the following formula :

$$\forall s, \quad a = argmax_a(Q_i(s,a) + c \times \sqrt{\frac{N_i(s,a)}{N_s}}) \tag{1}$$

  Where:
  1) $Q_i(s,a)$ is the action value estimate for action a and state s during day i
  2) c is a constant balancing between the exploitation and the exploration
  3) $\sqrt{\frac{N_i(s,a)}{Ns}}$ encourages the exploitation of less-visited actions during day i
  4) $N_i(s,a)$ visit count the edge (s,a) during day i
  5) N_s the total visit of state s during day i

- **The expansion :** For each day i, the expansion of the tree is given by the transition function that generates the next node (state s') for each node (state s) based on the action a taken. It also determines the immediate reward for taking the action. This transition function is the same as in the Q-learning model.

- **The simulation :** We define a function that performs a simulation of taking actions until reaching the depth of the tree $h = 60$ (since we have 60 days) and estimating the reward q from the simulation. The number of simulations realized in our code is $m = 50$.

- **The Backpropagation :** Once the reward q from the simulation is calculated using the previous step, it is propagated back up the tree, updating the value estimates $Q_i(s,a)$ and visit counts $N_i(s,a)$ along the path till we reach the root of the tree. The following formula gives the update of the value of Q:

$$Q_i(s,a) = Q_i(s,a) + \frac{q - Q_i(s,a)}{N_i(s,a)} \tag{2}$$

  1) $Q_i(s,a)$ is the action value estimate for action a and state s during day i
  2) $N_i(s,a)$ visit count the edge (s,a) during day i
  3) q the reward obtained at the end of the simulation before the back-propagation

In the case of the MCTS, the algorithm's efficiency also relies on the value of constant c (Eq (1)), which evaluates the contribution of exploration compared to exploitation. For a low c, the exploitation of already-seen actions is prioritized, whereas for a high c, the less-visited actions are prioritized, and the exploration is broadened. Since we want to prioritize the exploration by taking actions with fewer visits, the value of c needs to be high. By choosing a value of $c = 100$, we make sure that the exploration is broad enough, especially since we are dealing with an environment with many uncertainties and where the exploitation term $Q_i(s,a)$ is unreliable.

B. Q-Learning

We have a well-defined transition and reward model, but we wished to compare how the model-based MCTS would perform compared to a model-free algorithm so we implemented Q-learning. Unlike MCTS, Q-learning updates its estimates iteratively based on rewards received and the predicted value of future actions, making it particularly effective in environments with uncertainty or incomplete information. For our implementation, we applied Q-learning with an epsilon-greedy exploration strategy to balance exploration of new strategies with exploitation of known high-reward actions.

We have a finite horizon, so we do not use reward discounting ($\gamma = 1$) and our Q-value update function will be given by

$$Q_i(s,a) \leftarrow Q_i(s,a) + \alpha \left(r + \max_{a'} Q_{i+1}(s',a') - Q_i(s,a) \right)$$

We use the epsilon-greedy exploration algorithm. We train the action value function over 500,000 iterations; we simulate the entire 60-day period with each iteration since the largest rewards are sparsely distributed.

IV. Results

We train both algorithms from an initial state of understanding none of the chapters. We then test each resulting algorithm policy over 100 runs, also initializing with no student understanding. We compare the average reward across all of these runs with the random policy where a student chooses a studying action at random each day. The results are given in Table 1.

The Monte Carlo Tree Search Algorithm achieves a slightly higher average reward, reaching 374.35 as compared to 370.42 for the Q-learning Algorithm. We realize that the best policy for studying suggested by MCTS is to distribute the study of each chapter uniformly each day, in order to get the highest score in the quiz (don't leave your work till last minute). By comparison, Q-Learning suggests cramming in the days before the quizzes and only light studying in between. As we increase randomization in the ϵ-greedy function in the Q-learning algorithm, the result gets closer to the result obtained with the Monte Carlo Tree Search Algorithm with a constant $c = 100$.

Table 1 Both MCTS and Q-Learning beat the random policy averaged across 100 test iterations

Policy	Average Reward	Training Runtime
Random	272.50	N/A
MCTS	374.35	42 min
Q-Learning	370.42	104 min

A. Monte Carlo Tree Search : Analysis

The averaged policy and state-space results of 100 test runs of our trained MCTS algorithm is shown in Figure 1. The staircase nature of the plot is due to the fact that we restrict the chapters available for studying as the term progresses. We see that the policy derived from MCTS favors evenly distributed studying throughout the term with no major changes surrounding the quiz days. Since there are fewer chapters to study in the beginning of the term, these chapters are initially studying much more frequently and then the distribution thins out.

The difference between the two results and why the MCTS has a higher reward can be explained by how those two algorithms deal with exploration and exploitation. The Monte Carlo Tree Search uses UCB (Upper Confidence Bounds) to balance dynamically, whereas the Q-learning requires an explicit exploration defined by the ϵ-greedy function. In the first case (MCTS), the method ensures that even less obvious study strategies that can be beneficial are taken into account. In contrast, in the second case (Q-learning), the exploration relies on the performance of the ϵ-greedy algorithm.

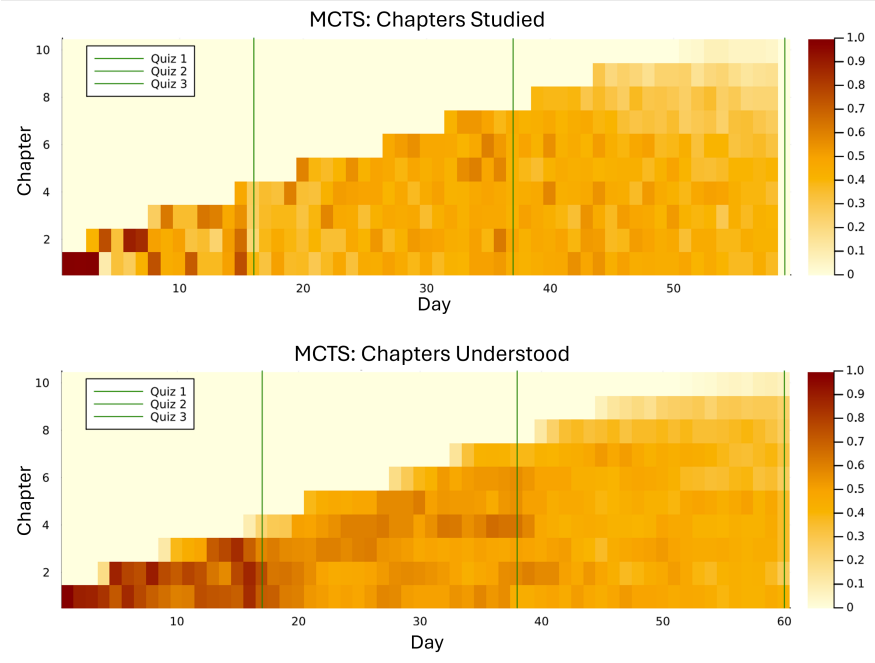


Fig. 1 Policy and state progression averaged over 100 test runs of the MCTS algorithm. Chapters can only be studied as they are introduced throughout the quarter.

B. Q-learning : Analysis

With some tuning, base Q-learning beat the random policy quite reliably. The policy is noticeably dependent on the inherent order of actions as they appear in our list of possible actions. This is because the argmax function in Julia chooses the first appearance of a maximum if there are multiple equivalent values. Thus, if actions with little studying are earlier in our list, the resulting policy will have less hours of studying. To counter this, we begin with a larger ϵ value of 0.3 and decay exponentially by $\alpha = 0.999997$ over 500,000 iterations. This helped the algorithm be less dependent on the inherent order of actions since it places higher value exploration over exploitation earlier in the training.

Figure 2 shows the averaged results of 1000 test runs using this exponential decay approach. The studying is much more evenly distributed and closer to the MCTS policy results than when we used a fixed $\epsilon = 0.1$ as shown in Figure 3. This suggests that more randomization and more training iterations are necessary for Q-Learning to perform reliably and without bias.

V. Conclusion

With the goal of improving AA 228 quiz grades, we have modeled the class as a sequential decision-making process where each day students choose which chapters to study in order to increase their understanding. Our two approaches: Monte Carlo Tree Search and Q-Learning yield vastly different policies but both beat the random policy. MCTS suggests distributing lighter studying across all the days in between quizzes while Q-Learning supports cramming in the days directly before a quiz. Both of our algorithms would greatly benefit from more training iterations and a better balance of exploration with exploitation.

Future improvements would largely involve training for longer and better balancing exploration with exploitation. In order to increase the fidelity of our work, some ideas include incorporating a non-binary competency score for each chapter and initializing training from different starting states to incorporate student background knowledge when entering the class. We could also include actual class data either from Carta or from quiz grades and student surveys to better understand study approaches.

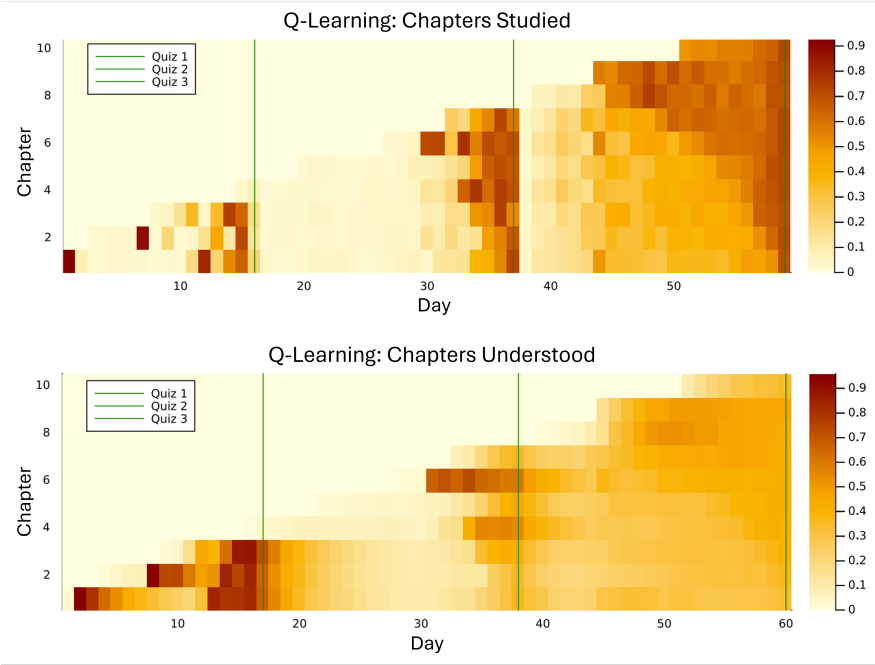
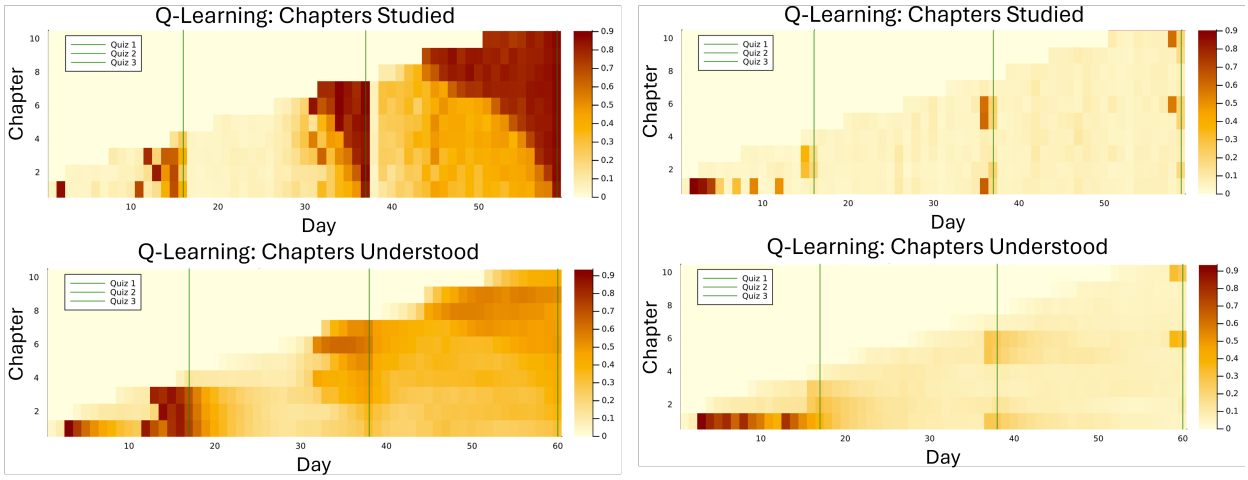


Fig. 2 Policy and state progression averaged over 1000 test runs of the Q-learning algorithm. Chapters can only be studied as they are introducing throughout the quarter.



(a) Studying prioritized

(b) Free time prioritized

Fig. 3 With decreased randomization ($\epsilon = 0.1$), Q-Learning favors ordering of the action-space during exploration. The two heatmaps above were generated with the same model parameters, but with the action-space reversed. Policy (b) did not beat the random policy.

VI. Team Member Contributions

Contribution of each member :

- Yasmina participated in defining the problem of the project, coded the Monte Carlo Tree Search algorithm and analyzed the MCTS results
- Alexandra participated in defining the problem of the project, worked primarily on Q-Learning and project scope
- Justin participated in defining the problem of the project, studied prior works, and aided with debugging and analysis

All team members cite equal contributions and worked together on the writeup.

References

[1] J. Chen, P. Sarin, and R. Ta. "How to 'do' school: Optimizing within injustice," *CS 238 Final Project*, Stanford University, 2023.

[2] A. Rafferty, E. Brunskill, T. Griffiths, and P. Shafto. "Faster Teaching via POMDP Planning," *Proceedings of AIED*, vol. 6738, pp. 280–287, 2011.

[3] K. Chung, D. Kim, S. Lee, and G. Jung. "Optimizing Spaced Repetition Schedule by Capturing the Dynamics of Memory," *IEEE Transactions on Learning Technologies*, 2023.

[4] Y. Matsuda, T. Matsubara, and H. G. Okuno. "RLTutor: Reinforcement Learning Based Adaptive Tutoring System," *arXiv preprint arXiv:2108.00268*, 2021.

[5] J. Ye, J. Su, and Y. Cao. "DRL-SRS: A Deep Reinforcement Learning Approach for Optimizing Spaced Repetition Schedules," *MDPI Applied Sciences*, vol. 14, no. 13, 2024.

[6] I. G. Ndukwe, B. K. Daniel, and R. J. Butson. "Data science approach for simulating educational data: Towards the development of teaching outcome models," *Big Data and Cognitive Computing*, vol. 2, no. 3, pp. 24, 2018.