Optimizing task scheduling for small businesses using reinforcement learning

Kento Perera, Timothy Sah, and Dean Stratakos
{kperera1, tsah, dstratak}@stanford.edu
13 December 2019

code.zip: https://drive.google.com/file/d/1M9rlD93UPqS9yCjl7fuibZf_fZU_JeF_/view
data.zip: https://drive.google.com/file/d/1-_j2XkdUo4CGtCWHzLVInGfiuxZuCA8m/view

***Abstract*—**Small businesses in the service industry face an abundance of challenges. Oftentimes, their success or failure is closely correlated with their ability (or lack thereof) to efficiently prioritize tasks in order to maximize profits on their services. Motivated by a problem we see in our everyday lives and inspired by the success of Markov Decision Process formulations on problems such as Blackjack, we developed a system that learns to optimize scheduling through experience. We present an example solution to a problem of prioritizing tasks with varying rewards and time constraints by translating it into a learning problem. Through the course of the project, we settled on two main AI algorithms: value iteration and Q-learning. These algorithms were compared to our naive baseline algorithms: random and FIFO. Our results showed that the AI algorithms outperformed the baselines; both of our AI algorithms adopt policies that are strategic in hindsight, resulting in a much higher total reward when compared to the naive algorithms. Furthermore, with respect to value iteration, Q-learning performs comparably and converges exponentially quicker.

I. INTRODUCTION

To ground our idea while incorporating personal significance, we designed a system with a specific local business in mind: Tennis Town & Country (http://tennistownandcountry.com/). Tennis Town & Country is a small tennis shop that provides a tennis racquet stringing service (in addition to selling tennis-related retail merchandise). This shop has a special deal with the Stanford Varsity Men's Tennis Team to string all of their players' racquets for a reduced labor rate. This poses a challenge to Tennis Town & Country, as they have to balance their demand from the Stanford Men's Tennis Team with their demand from their regular customers. With a large inflow of racquets daily, it grows increasingly difficult to account for all of the factors and optimally prioritize stringing orders. For all of these reasons, we felt that this shop, in particular, would greatly benefit from having an algorithm to help them optimize labor allocation when stringing racquets. We created a system to model their stringing service that maximizes the revenue of stringing racquets and minimizes the implicit costs of missing deadlines.

II. TASK DEFINITION

Given the number of racquets the shop can string in one day, our system constructs a

lookup table of the optimal set of racquets to string. In the real world, Tennis Town & Country would provide the set of racquets for a given day in addition to the leftover unstrung racquets from the previous day as input, and our system would return the optimal set of racquets for them to string that day.

The performance of our AI models was measured by comparing their net rewards to the net rewards obtained by our baseline models. Net rewards were calculated by our reward function which considered not only the true revenue from completing a request, but also the implicit costs (hand-crafted negative rewards) of failing to complete a request on time. Although these net rewards do not reflect the actual difference in profits, it is a consistent metric that enabled us to compare our results relative to one another. Lastly, we compared the algorithms' runtimes to determine which one performs most optimally when operating on larger data sets.

III. INFRASTRUCTURE

Data collection. In building our system, we first collected preliminary data from Tennis Town and Country and created a program to generate a sufficient amount of data in a clean, workable format. Since our project is specific to Tennis Town & Country, there was no online database to download; we had to collect data based on their own historical sales. To do so, we visited the shop and spoke with the owners, who were able to provide us with information for their sales for the whole year to date. We saw that not every string job costs the same depending on the requested time of completion, which translates into varying rewards in our model. Furthermore, the frequency of jobs requested varied. Specifically, the distribution is as follows:

Type	Price	% of sales
Speedy (while you wait)	$40	0.855%
Express (1 day)	$30	7.695%
Standard (3 days)	$20	76.95%
Stanford Men's Tennis - Speedy (while you wait)	$18	1.45%
Stanford Men's Tennis - Express (1 day)	$18	5.075%
Stanford Men's Tennis - Standard (3 days)	$18	7.975%

Data Generation. To generate realistic data for our model to use, we wrote a program that writes data into CSV files, using maximum likelihood estimation over the statistics and probabilities that we collected from Tennis Town & Country. The data generator program takes in up to four command-line arguments:

1. the number of hours per day to generate requests
2. the number of days to generate requests
3. the start date for the first day's requests
4. a boolean signifying whether or not the Stanford Men's Tennis team is in-season.

The final command-line argument is important because when the team is in season, there is a higher probability that a given request is one that is affiliated with Stanford Men's Tennis.

When run, this program writes data into a single CSV file containing five columns of data, specifically:

1. *Identification number of the request.* This allows us to keep track of the requests by their ID numbers.
2. *Boolean that is true if the request came from Stanford Men's Tennis.*

This is used to determine which set of probabilities to draw probabilities from with regards to the frequency of observing various types of requests.

3. *Type of request ∈ [Speedy, Express, Standard].*
   This allows us to determine the time frame for which a request needs to be fulfilled by in order to obtain the full reward and incur no penalties. It also allows us to determine how much to reward the fulfillment of a given request.
4. *Date (year, month, day concatenated) of the request.*
   Along with the time, this allows us to keep track of when each request was made and allows us to determine when the request must be fulfilled by.
5. *Time (hour and minute concatenated) of the request.*
   Along with the date, this allows us to keep track of when each request was made and allows us to determine when the request must be fulfilled by.

IV. APPROACH

Model: Markov Decision Process. Our problem falls under the category of a Markov Decision Process (MDP). The Markov property states, "The future is independent of the past given the present." Formally, an MDP is a tuple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma})$, where $\boldsymbol{S}$ is our state space, $\boldsymbol{A}$ is a finite set of actions, $\boldsymbol{P}$ is the state transition probability function, $\boldsymbol{R}$ is the reward function, and $\boldsymbol{\gamma}$ is a discount factor $\gamma \in [0, 1]$.

When formulating our MDP, we made the assumption that racquets are all dropped off at the start of the business day, so we created a state to capture the environment at this point. The state consists of two elements:

- a set containing all of the racquets dropped off that morning in addition to the racquets that were rolled over from the previous day
- the day number

Each racquet contains all of the necessary information as described in the data generation section above. Our MDP is defined as follows:

Start State: empty list of racquets to string on Day 1 of the time frame

End State: the end of one time frame (e.g. start of Day 7 when using a 6-day time frame)

Current State: a tuple consisting of

1. the set of all unstrung racquets with specifications for each request
2. the day within the time frame

Actions: a list of the possible subsets of racquets that should be strung on the current day (eg. if there are 20 racquets to be strung with the capacity to string 15 a day, there will be 20 choose 15 possible actions)

Succ(s, a): the new set of unstrung requests for the next day given the subset of racquets to string from the action

Reward(s, a, s'): the revenue of all racquets to be strung in the day minus the implicit cost of missing a deadline

- We are using the exact dollar amounts as positive revenue.
- Our implicit costs are calculated as a negative revenue of $20 times the number of days the racquet is overdue. Furthermore, there is a one day lookahead which penalizes unstrung racquets that are about to be overdue.

Throughout the process of formulating our MDP, we faced an array of challenges that

caused us to cycle through dozens of attempts. A large source of complications stemmed from handling a large state space, which was initially nearly infinite. Many of our code iterations revolved around reducing parameters in the state to prune it into a reasonable state space.

The second main challenge was finding an appropriate reward function. Simply modeling our rewards based on the dollar-value of fulfilling each request would lead to a model that greedily chooses the combination of racquets that yield the highest-dollar-value rewards. To prevent this, we incorporated a variation of a discount factor. Our strategy, in addition to considering the immediate rewards for each racquet strung on a given day, was to assign single-day lookahead penalties (negative rewards) to each unstrung racquet such that the penalty scales based on how close to being overdue and/or how far overdue each unstrung racquet is on the current day, and will be on the following day.

Furthermore, our reward function did not have a way to distinguish between requests of the same type with different deadlines. To account for this, we created a hand-crafted feature within our reward function to give a slight reward by a factor of .01 to prioritize requests that have closer deadlines.

Algorithm: Value Iteration. For our preliminary model, we used value iteration. Value iteration is simpler than Q-learning and is not too noticeably inefficient when being run on a small data set. We do this by calculating the reward over every possible action from a given state and choosing the action that maximizes the reward for that day. The reason that it is only applicable to smaller data sets is because it must run iterate through every possible state, which grows exponentially as the number of requests increases. However, because value iteration visits every possible state, it is guaranteed to converge to optimal values resulting in the maximum possible net profit.

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.
For iteration $t = 1, \ldots, t_{\text{VI}}$:
  For each state s:
    $$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s,a)}$$

Algorithm: Q-learning. In order to accommodate more realistic sized datasets, we implemented a Q-Learning algorithm to solve the MDP.

Q-Learning works by learning an action-value function that expresses the net reward of taking a given action in a given state.

In each training cycle, the agent explores the environment and accumulates the rewards until it reaches the end state, in our case the end of the time frame. The purpose of the training is to increase the knowledge of the agent represented by the Q-values. As the algorithm runs more training cycles, the Q-Values will be updated so that the agent can take the optimal action.

One critical component for the Q-Learning algorithm is the exploration probability represented by epsilon, ϵ. A high exploration probability means that the agent will mostly choose an action at random, exploring new states that may or may not be the optimal policy. Meanwhile, a low exploration policy will exploit the optimal policies learned from previous iterations and is less likely to explore a new state. We decided to use a dynamic epsilon function to decrease the exploration probability as the number of episodes increases. By implementing this epsilon-decreasing function, we were able to minimize exploration as our model becomes more knowledgeable and effectively converge at an optimal total reward.

Baselines and Oracle. We created two baseline models to provide a point of comparison against our preliminary model. Specifically, we implemented a random choice selector and a first-in-first-out (FIFO) model.

The random choice selector imitates the scenario where a racquet stringer grabs random racquets from the set of racquets to be strung, where each of these racquets is a request that needs to be fulfilled. Intuitively, this is a highly inefficient approach since it blindly selects racquets without considering any information on the request date or type.

The FIFO baseline model imitates a situation where the racquet stringer fulfills stringing requests in the order of the date and time that they were received. This model performs better than the first random choice baseline. However, this approach is still rudimentary in that it fails to take into account the type of request or other aspects of reward optimization.

An oracle would be an algorithm that could predict the future or know the outcome of probabilistic events deterministically. In our case, an oracle might know exactly what the maximum capacity of racquets that can be strung in a day is, assuming it is non-static.

V. LITERATURE REVIEW

Because our model is tailored to fit the business model of Tennis Town & Country, one challenge has been finding research that is comparable to our project's specifications. That being said, one research paper titled *Reinforcement-Learning-based Foresighted Task Scheduling in Cloud Computing* [1] has provided some useful information regarding the overarching topic of task scheduling using reinforcement learning. The purpose of this research was to find an optimized scheduling approach for mapping tasks to the computer processors' resources with the goal of decreasing response time and increasing system utilization rate. Similar to our project, their algorithms consisted of random scheduling and order scheduling (FIFO) as their baseline with Q-learning as their main algorithm.

One approach that was consistent between our projects was addressing the issue of an infinite state space. We both adjusted our model to discretize continuous values in the state space. Specifically, in our case, we reduced the state space by assuming that all the racquets arrived in the morning as opposed to a continuous inflow of racquets throughout the day. Additionally, we set an upper bound to the number of racquet requests that can be considered in one day, ensuring that the number of possible states is always finite.

One aspect of their algorithm that was effective was updating their epsilon value over many cycles. Specifically, they used the following epsilon equation:

$$\varepsilon = \varepsilon - \frac{\varepsilon}{total\,cycle} \times cycle\,number$$

This equation allows the Q-learning algorithm to have more exploration in the initial stages and then gradually decide more greedily, giving more weight to the policies learned from previous cycles. As for our project, we had initially applied a constant epsilon value for each test. However, we adopted this method to update our epsilon value automatically within each test run. After implementing this dynamic epsilon value, we noticed that our total reward converged much more quickly.

As for the results from their research, their model found that for a test set of 60 tasks, Random Choice had an average response time of 298 seconds, FIFO had an average response time of 254 seconds, and Q-Learning had an

average response time of 172 seconds. This was consistent with our findings when comparing the results of Q-Learning with the baseline algorithms.

Another source that was useful to our research in creating our model was titled, *A Reinforcement Learning Approach For Scheduling Problems* [2]. In this paper, the researchers used Q-Learning to optimize the allocation of limited manufacturing resources over a number of parallel manufacturing tasks. Similar to the previously analyzed paper, this project focused on the allocation of several resources to parallel tasks whereas our project focused on the optimal scheduling for one resource, the racquet stringer. One way that we could expand our problem to fit their model in future works is by considering the number of racquet stringers (resources) at Tennis Town & Country and individualize the number of racquets and the type of requests each stringer can complete. This would alter the optimal schedule because the requests will be allocated over multiple resources with different constraints.

VI. ERROR ANALYSIS

We tested our models on various datasets.

Small dataset. In our first iteration prior to the poster session, we tested on a small dataset of 75 requests over 6 days with the capacity to string 13 racquets per day. We ran the tests on value iteration and Q-learning and produced Fig. 1 and Fig. 2.

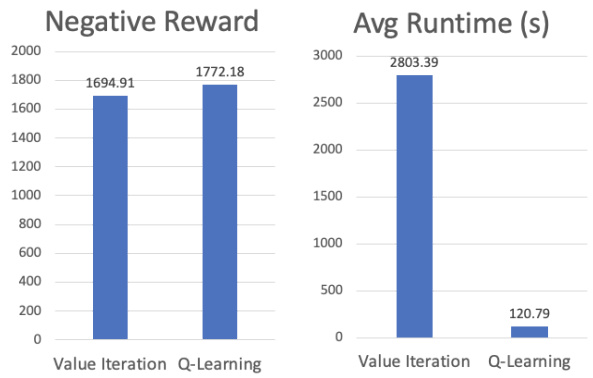

Fig. 1: Value Iteration vs. Q-learning; negative reward (left) and average runtime (right) on small dataset.

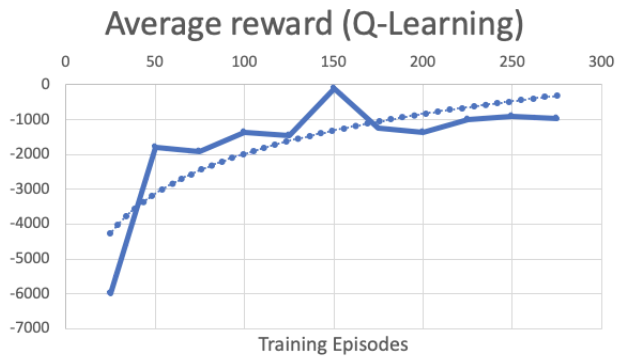

Fig. 2: Average reward over 300 training episodes for Q-Learning with constant exploration probability $\epsilon = 0.2$ on small dataset.

These tests showed that value iteration produced slightly less negative reward, meaning that it was better from an economical standpoint. However, it came at a huge price in time - over 46 minutes for value iteration in comparison to just over 2 minutes for Q-Learning. Furthermore, Fig. 2 shows how the Q-Learning algorithm improves over time. Yet we see that improvement is minimal after around 100 iterations. We will address the root of this observation when we discuss varying the exploration probability.

Large dataset. In our second major iteration, we had two main adjustments.

First, we included a bound on the number of racquets that the shop could accept, which prevented a large buildup of negative rewards. The main effect was that our rewards were now largely positive rather than largely negative.

Second, we increased the size of our dataset. This set of experiments was conducted on a larger dataset of 88 racquets over 6 days. Even though there are only 13 more racquets than the previous test set, the number of possible combinations in the state space increases exponentially, rendering value iteration useless due to an exponentially longer runtime.

Due to this fact, we only compared the results of Q-Learning from this larger data set to the results of FIFO and our random baseline algorithms. From Fig. 3, we see that Q-Learning performs slightly better than FIFO and performs nearly twice as well as random. One reason that our Q-Learning result may be comparable to FIFO is that on the data sets that we generated, for most cases, the best policy was to string the oldest racquets to avoid incurring a penalty for being late. One way that may have allowed Q-Learning to outperform FIFO more significantly was if we had requests that had a larger variability in the number of days it could take. For example, if we included another type of request that could take 10 days, we would see Q-Learning would take a smarter approach and save those requests for later, whereas FIFO would string those requests in the same order.

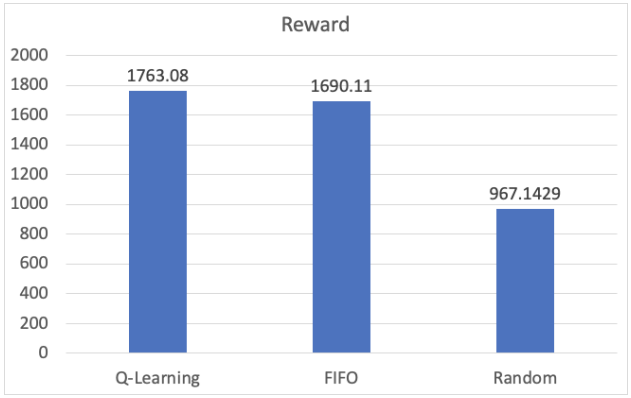

Fig. 3: Rewards for Q-Learning, FIFO, and random on large dataset.

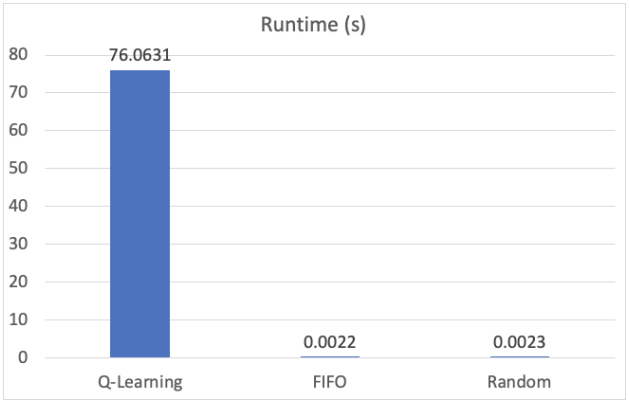

Fig. 4: Rewards over training episodes for Q-Learning with constant exploration probability $\epsilon = 0.2$ on large dataset.

Another key difference between Q-Learning and the baselines was the stark difference in runtime, as shown by Fig. 4. This is due to the simplicity of the FIO and Random algorithms. This was expected, however, as the point of the Q-Learning algorithm was to implement a much more sophisticated algorithm to improve reward, which it succeeded at doing even though the runtime was significantly higher. Note that the Q-Learning runtime is still drastically better than value iteration would have taken.

Variable exploration probability. Another major area we experimented on was a varying exploration probability.

As we see in Fig. 2, when calculating the average reward over buckets of 25 training episodes, we notice drastic improvement of average reward over the first few training episodes, but then the average reward appears to plateau, presumably due to the constant probability for exploration. Furthermore, when analyzing the average reward, it fails to reach the optimal point of convergence as seen in Fig. 6, Fig. 7, and Fig. 8. For a more detailed version of this graph with further explanation, proceed to Fig. 5.

We also examined what happens when varying our exploration probability, ϵ. We considered four different scenarios—constant $\epsilon = 0.2$, as well as an epsilon-decreasing policy with $\epsilon_0 = 1.0$, $\epsilon_0 = 0.5$, and $\epsilon_0 = 0.2$, and plotted the results over 500 training episodes with our small data set containing 75 racquets over 6 days with the capacity to string 13 racquets per day.

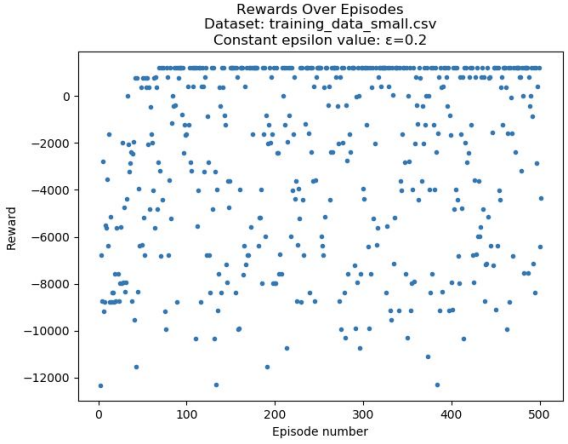

Fig. 5: Rewards over training episodes for Q-Learning with constant exploration probability $\epsilon = 0.2$.

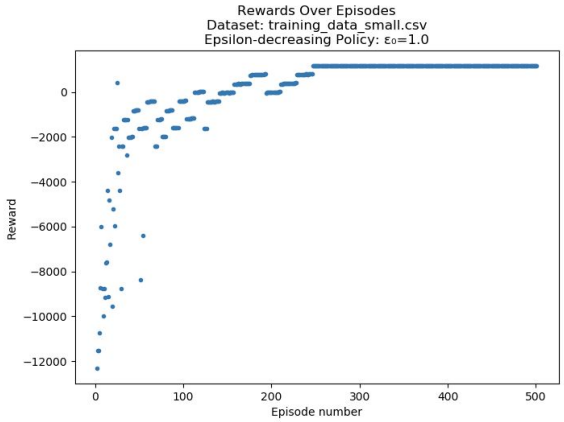

Fig. 6: Rewards over training episodes for Q-Learning with epsilon-decreasing policy with initial exploration probability ϵ_0=1.0.

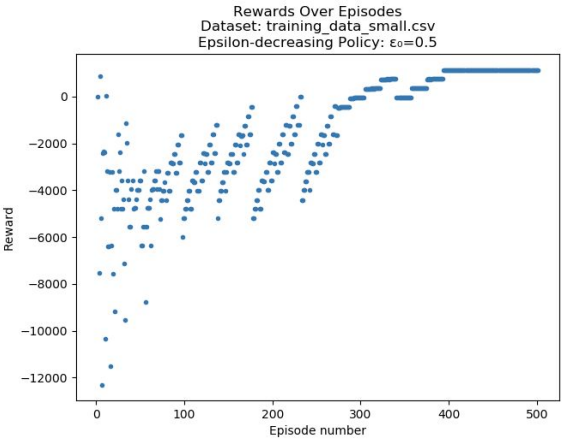

Fig. 7: Rewards over training episodes for Q-Learning with epsilon-decreasing policy with initial exploration probability ϵ_0=0.5.

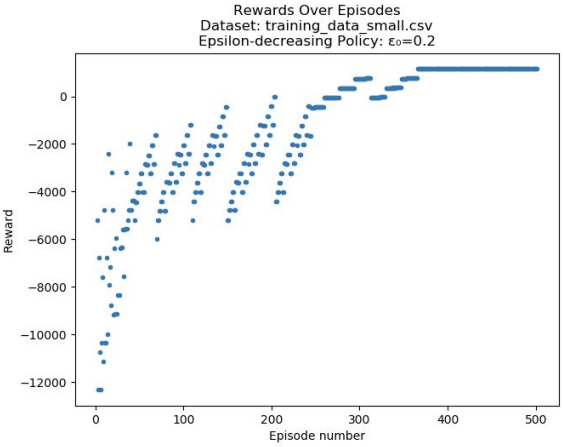

Fig. 8: Rewards over training episodes for Q-Learning with epsilon-decreasing policy with initial exploration probability ϵ_0=0.2.

It is clear that although a constant exploration probability seems to produce favorable results with respect to learning and rewards-production when considering the average rewards over multiple episodes (as shown in Fig. 2), we can clearly see in Fig. 5 that the per-episode rewards are, overall, quite volatile. Further, we can see why the average reward for episodes 100+ failed to converge to the optimal reward value of 1199.16—the constant chance of exploration severely brought down the average reward. With this in mind, we realized that in order to reach a state of convergence (which, after all, is the goal of Q-Learning), we should consider implementing an epsilon-decreasing policy. Inspired by our findings from [1], we adapted an epsilon-decreasing policy where the exploration probability ϵ decreases from $\epsilon=\epsilon_0$ at the first episode, to $\epsilon = 0$ by the last episode, using the following equation:

$$\varepsilon = \varepsilon - \frac{\varepsilon}{total\ cycle} \times cycle\ number$$

With this in place, we examined three different starting exploration probability values ϵ_0: $\epsilon_0 = 1.0$ (Fig. 6), $\epsilon_0 = 0.5$ (Fig. 7), and $\epsilon_0 = 0.2$ (Fig. 8), over which our results matched our predicted outcome of achieving convergence. We note that among the epsilon-decreasing cases, $\epsilon_0 = 1.0$ converged to the optimal reward value 1199.16, while $\epsilon_0 = 0.2$ took the most number of episodes to do so. This detail was unexpected because our epsilon-decreasing function should assert that lower starting ϵ_0 values approach $\epsilon = 0$ over less episodes.

VII. FUTURE WORK

Through this project, we captured a large essence of the problem faced by Tennis Town & Country. However, we have identified a handful of features that we could have tackled with more time. These features would serve two main purposes: to increase the robustness of the model to handle edge cases and to make the overall system even more representative of reality.

The first obvious change we could make would be to run our system on even larger datasets. Due to the immense size of our state space, tests were slow with Q-learning and painfully drawn out with value iteration. With more time, we could run more tests and continue to fine-tune our models. These are general strategies to improve upon our work, which would simply come with time.

In our model, we have a set number of racquets that the shop can string on any given day. In general, they typically string the same number of racquets from day to day with high consistency. Nonetheless, there is a slight variation in their daily capacity for various reasons. Thus, we could include random probabilities for the capacity of racquets that can be strung in a day in our transition function. By doing this, we account for the varying levels of store activity in the week.

Furthermore, a significant assumption we made was that racquet requests all come at the beginning of the workday. In reality, racquets are dropped off throughout the day. To model this, we would have to adjust the granularity of our states from a per-day basis to a per-racquet basis. We did not use a per-racquet approach in our implementation because with a per-day granularity, we were able to limit the state space to a workable size. Had we used a per-racquet approach, we would have quickly found ourselves facing an infinitely large state space, since we would also

have to keep track of the exact drop-off time for each racquet.

In order to counteract the vast state space required by a problem of this nature, we could also implement function approximators. Function approximators could be used to represent policies with the intent that the agent should follow similar policies for states that are relatively similar. This could be achieved using a deep neural network in our model, but this would undoubtedly require more time.

Above all, the problem of task optimization can be extended to countless applications ranging in social impact. Ideally, our implementation would be generalized to allow for widespread expansion to a wide variety of task scheduling for businesses in the service industry. Some potential applications of our research include optimizing scheduling for repair-based businesses such as car mechanics and electronics stores. Furthermore, it could be used to find an optimal queue of appointments for things such as doctor visits and office hours, factoring in the specifications of each request and creating an appropriate reward function.

VIII. REFERENCES

[1] S. Mostafavi, F. Ahmadi, and M. Sarram. Reinforcement-Learning-based Foresighted Task Scheduling in Cloud Computing (https://arxiv.org/pdf/1810.04718.pdf).

[2] Yunior César Fonseca Reyna, Yailen Martínez Jiménez, Juan Manuel Bermúdez Cabrera, Beatriz M. Méndez Hernández. A Reinforcement Learning Approach For Scheduling Problems (https://pdfs.semanticscholar.org/aa39/8b10ebfddaece6889a49e5ec77620ed69ee5.pdf)

[3] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, Srikanth Kandula. Resource Management with Deep Reinforcement Learning (https://people.csail.mit.edu/alizadeh/papers/deeprm-hotnets16.pdf)

[4] Xu Yuan, Lucian Busoniu, Robert Babuska. Reinforcement Learning for Elevator Control (http://busoniu.net/files/papers/ifac08-elevators.pdf)

[5] Atman Rathod. Machine Learning for Small Businesses: Unearthing the Revolutionary Potential (https://www.dataversity.net/machine-learning-for-small-businesses-unearthing-the-revolutionary-potential/)