Optimizing task scheduling for small businesses using reinforcement learning

Case study: Tennis Town & Country

Kento Perera, Timothy Sah, Dean Stratakos

Stanford Computer Science

Introduction

Tennis Town & Country is a local tennis shop that sells all things tennis-related and provides racquet stringing services. With a large inflow of racquets daily and different stringing request types, it is challenging to determine the best order in which to fulfill these requests. As members of the Stanford Varsity Men's Tennis Team who use their services, we created a reinforcement learning algorithm to find an optimal prioritization for their stringing service that maximizes revenue and minimizes implied costs. We believe the model can be adapted to fit other small businesses with similar models.

Data Collection

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

After speaking with the owners of Tennis Town & Country to learn about their pricing model and average frequency of different jobs requested, we used the distribution shown above to create a **data generator** with the following properties:

- Determine the quantity of requests for a given day using a Poisson random variable (requests per hour) over 9 hours
- Categorize each request based on weighted random choice with the above distribution

Markov Decision Process

We formulated the problem as a Markov Decision Process with the following characteristics:

Start State: (racquets, day); Start state is a tuple of racquet requests on Day 1.
End State: The end of a single time frame (6 days)
Current State: (tuple of unstrung racquets from previous and current days, current day number)
Actions: Subsets of racquets that can be strung on the current day (e.g. if there are 20 racquets to be strung with the capacity to string 15 per day, there will be 20 choose 15 possible actions)
Succ(s, a): The new set of incomplete requests given the subset of racquets.
Reward(s, a, s'): the revenue of all racquets to be strung in the day minus the implicit costs of missing any deadlines

Q-Learning (best overall):

- Balances exploring new states with exploiting previously learned policies
- Greatly improved runtime, especially on larger datasets since it does not have to repeatedly iterate over every possible state
- Continuously updates Q* (optimal policies) with every episode over a dataset

Value Iteration (good on small datasets):

- Iterates over every possible state and backsolves to find the optimal policy
- High compute -> memory expensive and time consuming, especially on big datasets
- Guaranteed to find optimal policy

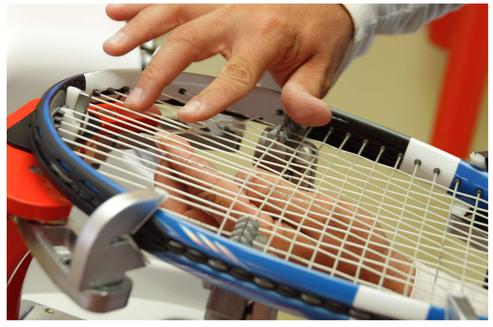
FIFO (first-in, first-out; baseline model):

- String racquets in the order that they were brought to the shop

Approach

Specific considerations to our MDP included

- Assigning the rewards for stringing a set of racquets based on the dollar revenue of fulfilling the corresponding request-type
- Enforcing implicit penalties (negative rewards) for failing to fulfill requests on time
- Creating hand-crafted features:
  - Give slight priority to older requests
  - Penalize unstrung racquets as its due date approaches
- Extracting the list of racquets from the state to compare states and reuse information across days



Challenges

Value iteration

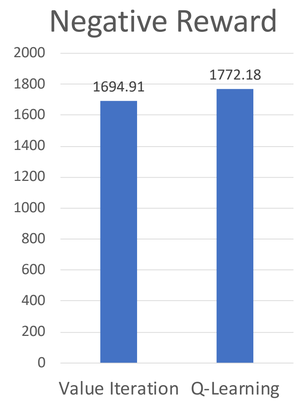
- Fine tuning the penalty for overdue requests against the rewards of fulfilling current requests

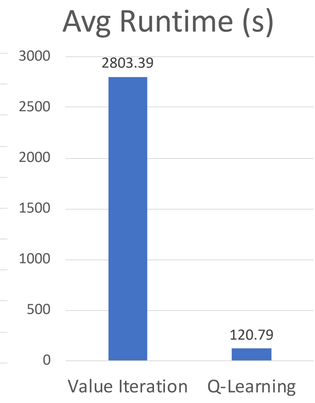
Q-Learning

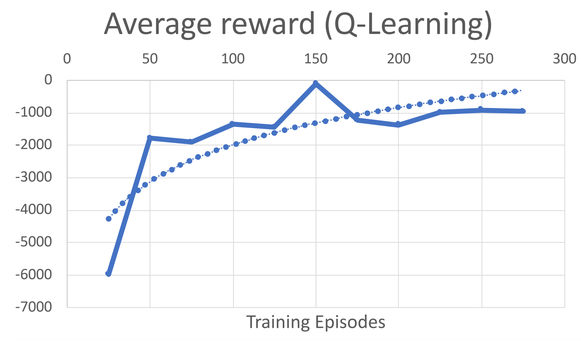
- Determining the time frame over which to optimize
- Minimizing the state space
  - Condensing the number of features in each state

Results & Error Analysis

We tested both models on various datasets. These are results for 75 requests over 6 days with the capacity to string 13 racquets per day.







Conclusion

- Q-Learning has great speed improvements at the price of a small reduction in reward
- Could be applied to other businesses and tasks
- In the future, we could
  - Run on larger datasets with more compute power
  - Implement Deep Q-Learning / Neural Networks to achieve different results
  - Add more features to make our model more representative of reality