Multi-Segment Dynamic Pricing for Airline Tickets Using Model-Free Reinforcement Learning

Ross B. Alexander
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA 94305
rbalexan@stanford.edu

Jonathan S. Ling
Department of Management Science and Engineering
Stanford University
Stanford, CA 94305
jonling@stanford.edu

Abstract

Dynamic pricing in the airline industry demonstrates some of the most effective pricing schemes in business to maximize revenue based on customers' willingness to pay for particular goods at particular times. In this problem, the airline's goal is to set ticket prices to maximize its revenue over a finite time horizon. This paper considers the case of pricing a specific inventory item (e.g., economy class tickets) that is sold to different customer segments based on their inputs when purchasing the ticket (e.g., days to departure, one way/return, length of trip, etc.) and which thus has different price points. We review common approaches taken to solve the single-segment (i.e., single price point) dynamic pricing problem and formulate a multi-segment (i.e., multiple price points) dynamic pricing problem using a factored multi-agent Markov decision process (MMDP) framework. We solve the multi-segment dynamic pricing problem using modern reinforcement learning (RL) approaches including Sarsa and Sarsa-Lambda, and compare the performance of these approaches to various baseline policies.

Introduction

Dynamic pricing problems arise in many industries where goods and services are subject to a finite horizon. This finite horizon can be imposed both naturally or artificially, and often takes the form of perishability or expiration constraints of goods or services. In some cases, the inventory of goods or providers of services may also be finite. In attempting to maximize revenue, establishing a dynamic pricing policy can lead to increased revenue of a static pricing policy (Figure 1). A thorough survey of dynamic pricing reveals its pervasive presence in essentially all industries, and as a result, the sheer variety of approaches taken to date to solve the dynamic pricing problems under the given constraints (den Boer 2015).

In the airline industry, a natural dynamic pricing problem is evident – an airline must maximize revenue for a finite set of tickets for a given flight before the flight departs, since sales cannot be made after the flight has departed. The airline must balance the certainty of selling all tickets, the maximization of revenue, and a stochastic demand. While most customers purchasing airline tickets likely know that the price of a given ticket fluctuates with time-to-departure,

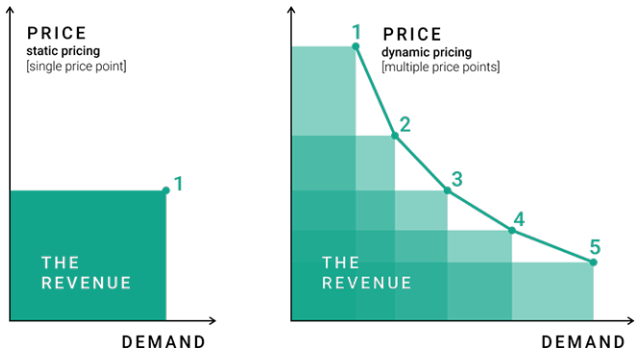

Figure 1: Maximization of revenue can be achieved using dynamic pricing policies instead of static pricing policies to exploit variation in demand. (Competitoor 2016)

seasonality, trip type, etc., customers may be unaware that airlines typically offer a number of fare classes for various customer segments for the same seats on a single flight (e.g. multiple fare classes for an economy ticket). For example, purchasing a one-way ticket, a return ticket spanning a weekend, or a return ticket spanning a week will mean different costs to the customer for the same leg of the journey, despite being for the same seat. The International Air Transport Association (IATA) has standardized a system of 26 different fare classes (formally called *Reservation Booking Designators*) that airlines use to characterize these different customer segments, which thus gives rise to 26 possible price points (Touraine and Coles 2018). As a result, a critical but often unseen dynamic pricing problem naturally arises in the airline industry for maximizing revenue over the different fare classes for a single flight.

Decision theory provides a foundation for formulating these types of dynamic pricing problems, and dynamic programming and reinforcement learning provide methods for solving these problems to generate an optimal or approximately optimal pricing policy. More detail is given in the Proposed Approach section, and a survey of decision-theoretic approaches for airline ticket dynamic pricing problems is given in Abdella et al. (2019).

Literature Review

In the 2000's, most dynamic pricing problems were solved using model-based reinforcement learning methods that explicitly described the interaction between demand and pricing (Rana and Oliveira 2014). While widely popular and theoretically sound, the challenge of accurate model representations led to reduced performance of these dynamic pricing policies. Model-free approaches are a relatively new area of research, and are proving to be more relevant given the explosion of big data applications, unknown and complex model structures, and situations with high uncertainty.

Here, we provide a brief survey of model-free approaches to the dynamic pricing problem in the literature. Carvalho and Puterman (2003) have studied parametric forms of the customer arrival distribution, and used reinforcement learning to estimate the parameters to predict demand and compute optimal dynamic policies, partially bridging the gap between model-based and model-free approaches. Rana and Oliveira (2014) have used reinforcement learning and Q-learning to compute optimal pricing policies under incomplete information and non-stationary demand. We extend this by considering multiple customer segments. Raju, Narahari, and Ravikumar (2006) have used Q-learning to dynamically price products with customer segmentation, but their segments are specially characterized and do not allow for generalizations to more segments, which we study.

This paper adds to the dynamic pricing reinforcement learning literature by considering how to price for multiple customer segments from a single pooled inventory source.

Proposed Approach

Multi-Agent Markov Decision Processes (MMDPs)

In the general hierarchy of decision-theoretic approaches for solving multi-agent sequential decision-making under uncertainty problems (Figure 2), the most general framework is the partially-observable stochastic game (POSG) where each agent maintains its own individually partially-observable state space, action space, observation space, and reward model (Hansen, Bernstein, and Zilberstein 2004). Decentralized partially-observable Markov decision processes (DEC-POMDPs) are a subclass of POSGs that have a single reward model rather than a reward model for each agent, though the agents' objectives may still be competing. Decentralized Markov decision processes (DEC-MDPs) are a subclass of DEC-POMDPs characterized by joint full-observability of the joint state, and multi-agent Markov decision processes (MMDPs) are a subclass of DEC-MDPs characterized by individual full-observability of the joint state. A detailed overview of DEC-POMDPs and their subclasses is given in Beynier et al. (2013).

An MMDP (Boutilier 1999) is defined by the 5-tuple $\langle n, \mathcal{S}, \mathcal{A}, T, R \rangle$ where n is the number of agents ($i \in \{1, ..., n\}$), $\mathcal{S}$ is the joint state space, $\mathcal{A}$ is the joint action space, T is the transition model, and R is the reward model.

In a factored MMDP, depicted in Figure 3, the joint state space can be factored into a representation composed of the environment state space $\mathcal{S}_0$ and the local state space $\mathcal{S}_i$ for agent i such that the joint state space is $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times ... \times \mathcal{S}_n$. The joint state s, which is jointly fully-observable, is thus the specific state given by the environmental state s_0 and sub-state s_i of all n agents ($s = \langle s_0, s_1, ..., s_n \rangle$). The joint action space can be represented similarly using the action space $\mathcal{A}_i$ for agent i such that the joint action space is $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$. The joint action a is given by the actions a_i of all n agents ($a = \langle a_1, ..., a_n \rangle$). The transition model gives the probability $T(s, a, s')$ of transitioning to the joint state s' from the joint state s after executing the joint action a. A general factored MMDP does not necessarily have transition independence; that is, in general, $T \neq T(s_i, a, s'_i)$. Additionally, the reward model $R(s, a, s')$ is the reward received for transitioning to the joint state s' from the joint state s after executing the joint action a.

Note that MMDPs are a superclass of Markov decision processes (MDPs), since all MDPs are degenerate 1-agent MMDPs (i.e. $\langle \mathcal{S}, \mathcal{A}, T, R \rangle \triangleq \langle 1, \mathcal{S}, \mathcal{A}, T, R \rangle$). The solution to an MMDP is given by a joint policy $\pi = \langle \pi_1, ..., \pi_n \rangle$. The complexity of solving an MMDP is known to be P-complete (Papadimitriou and Tsitsiklis 1987).

Dynamic Pricing Problem

The multi-fare dynamic pricing problem is the problem of maximizing revenue for a given flight with some finite number of seats, non-stationary demand across multiple categories, and a finite horizon induced by the flight departure date. Since we can utilize customer details provided in a trip query (e.g., one-way/return, length of trip, etc.), we can develop multiple fare classes (price classes) to exploit latent differences between such demand categories to maximize revenue. Thus, agency in our dynamic pricing problem is a representation of each fare class' independent pricing and sale of tickets. We consider the case where the seats are identical and fungible (e.g. all in economy), and thus the number of tickets available is from a common inventory pool.

It should also be noted that although the customer dynamics are stationary – customers arriving at each time step are defined by distributions that do not change with time – the customer arrival dynamics are non-stationary – fewer customers arrive as the departure date approaches – so the aggregate demand dynamics are non-stationary.

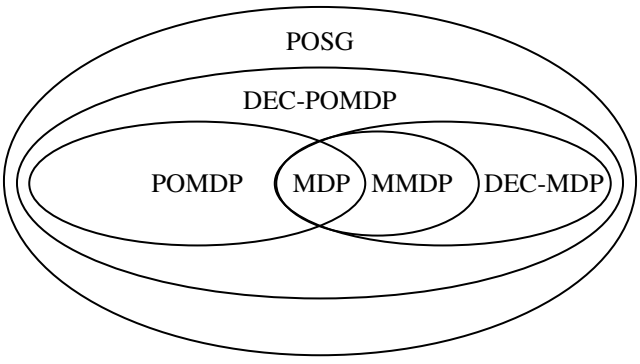

Figure 2: Hierarchy of decision-theoretic approaches for multi-agent sequential decision-making under uncertainty.

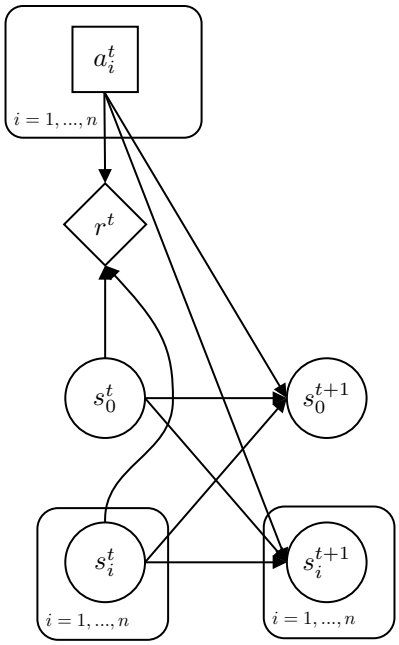

Figure 3: Structure of the factored multi-agent Markov decision process (MMDP) as a dynamic decision network.

For the dynamic pricing problem, we use a discretization of the joint state and action spaces to generate the joint state and action sets, respectively. Environmental state variables include the number of tickets available and a discretized time. Each agent represents a fare class, with their sub-state variable being the number of tickets sold, and action being a ticket price, for that fare class. The transition model is formulated as a generative model based on customer dynamics to compute the new joint state after taking the joint action from the previous joint state. The reward model, describing the total revenue, is exactly known, and is easily computed as the sumproduct of ticket prices and quantities sold across all fare classes.

Joint State Set In our dynamic pricing problem, the environment state s_0 is defined by the tuple $(v, t) \in \{0, ..., V\} \times \{0, ..., T\}$, representing the number of available tickets (or vacant seats) v given the number of initially available tickets V, and the time t given the finite horizon T. The sub-state s_i for agent i is defined as the number of tickets sold $u \in \{0, ..., V\}$ in the previous time step.

Joint Action Set The action a_i for agent i is defined as the ticket price $a_i \in \{\underline{p_i}, \underline{p_i} + \tilde{p_i}, \underline{p_i} + 2\tilde{p_i}, ..., \overline{p_i}\}$ given the agent's lower and upper pricing bounds, $\underline{p_i}$ and $\overline{p_i}$, and some spacing constant $\tilde{p_i}$, such that the cardinality of the agent's action set is fixed (e.g. $|\mathcal{A}_i| = 10$).

Transition Model The transition model that maps the current joint state s, the joint action a, and the next joint state s' to a probability $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ with probability constraints $T(s, a, s') \geq 0$ and $\sum_{s'} T(s, a, s') = 1 \, \forall \, s, a$ cannot be explicitly represented for our dynamic pricing problem without knowing the customer dynamics. Instead, we will formulate our own generative model $(s', u', r) \sim G_i(s, a)$ to simulate the transition from s to s' after executing a, for each agent i.

Since the dynamic pricing MMDP employs a factored state, we can describe the transition in terms of the environment state transition $s_0 \to s_0'$ and the sub-state transition for agent i, $s_i \to s_i'$.

For agent i, we define a set of customers seeking tickets C_i (initially empty). At each time step, we have n_i new customers arriving according to a Poisson distribution with a linearly time-varying parameter governed by the fare class parameters α_i and β_i, such that $n_i \sim \text{Pois}(n \mid \alpha_i t + \beta_i)$. The jth new customer $c_{i,j} \, \forall \, j \in \{1, ..., n_i\}$ has some willingness-to-pay (WTP) price threshold $w_j \sim \mathcal{N}(w_j \mid \mu_i, \sigma_i)$ and some willingness-to-pay (WTP) flexibility $k_j \sim \mathcal{U}(k_j \mid a_i, b_i)$. The customer purchase dynamics follow the given cases, with $b_j = 1$ if the customer purchases the ticket and $b_j = 0$ if the customer does not purchase the ticket:

$$\begin{cases} b_j \sim \text{Bernoulli}(2 \cdot (1 - \Phi(a_i))) & \text{if } a_i > w_j \\ b_j = 1 & \text{if } a_i \leq w_j \end{cases}$$

where $\Phi = \text{CDF}(\text{Logistic}(a_i \mid w_j, k_j))$, and the factor of 2 is used to rescale the probability to a $[0, 1]$ range (since this first case is valid only for $a_i > w_j$, which caps the value of Φ at 0.5). If the customer $c_{i,j}$ purchases a ticket, they are removed from the agent's set of customers $C_i \leftarrow C_i \setminus \{c_{i,j}\}$, where $c_{i,j}$ is characterized by the tuple (w_j, k_j).

Then the environment state $s_0 = (v, t)$ is updated by subtracting the number of tickets sold by all agents in the previous time step and incrementing t such that $s_0' = (v - \sum_i s_i', t + 1)$.

The pseudocode for this generative model is given in Algorithm 1. At each time step, we select some number of new customers from a Poisson distribution (with a linearly decreasing mean), randomly generate their WTP characteristics, and from this, determine how many tickets get sold.

Algorithm 1 GENERATIVEMODEL$_i(s(v, t), a)$

$u' \leftarrow 0$
$n_i \sim \text{Poisson}(\alpha_i t + \beta_i)$ for some constants α_i, β_i
for j in 1 to n_i **do**
  $w_j \sim \mathcal{N}(\mu_{w_i}, \sigma_{w_i})$
  $k_j \sim \mathcal{U}(\mu_{k_i}, \sigma_{k_i})$
  $C_i \leftarrow C_i \cup \{(w_j, k_j)\}$
end for
for $j \in C_i$ **do**
  if $a > w_j$ **then**
    $b_j \sim \text{Bernoulli}(2 \cdot (1 - \text{CDF}(\text{Logistic}(a \mid w_j, k_j))))$
  else
    $b_j = 1$
  end if
  if $b_j = 1$ **then**
    $u' \leftarrow u' + 1$
    $C_i \leftarrow C_i \setminus \{(w_j, k_j)\}$
  end if
end for
$r \leftarrow a \cdot u'$
return $((v - u', t + 1), u', r)$

We have chosen distributions over each of the random variables in a way that represents the problem dynamics. In some cases, such as the customer's WTP threshold and WTP flexibility, alternative distributions can also be justified. The customer's WTP is modeled as a complementary cumulative distribution function for the logistic distribution. The following figures depict the customer pricing dynamics and customer arrival dynamics.

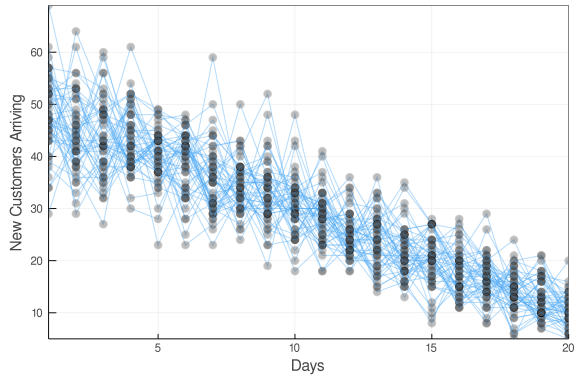

Figure 4: 50 samples of the arrival of customers over a 20-day horizon

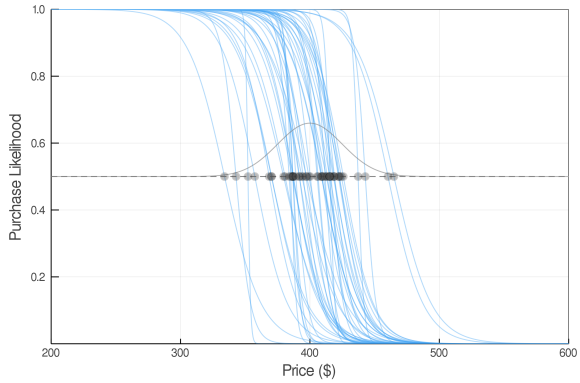

Figure 5: 50 samples of customer purchasing schedules

Reward Model The joint reward is the revenue r received from the sales of tickets at their respective prices, which enables an explicit representation of the reward function using agent-wise additive decomposition so that $r = \sum_{i=1}^{n} a_i s'_i$ or, using vector representations of the joint action $\mathbf{a}$ and new joint state excluding the new environment state $\mathbf{s}'_{\setminus 0}$, $r = \mathbf{a}^T \mathbf{s}'_{\setminus 0}$.

Model-Free Reinforcement Learning

Although the dynamic pricing MMDP is defined by the tuple $\langle 1, \mathcal{S}, \mathcal{A}, T, R \rangle$, the transition model cannot be explicitly represented. As a result, a model-free method for determining an optimal or approximately optimal policy must be used.

Q-Learning Q-learning is a model-free reinforcement learning approach that focuses on learning a state-action value function Q that maps the state and action to the expected future reward $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The corresponding policy $\pi : \mathcal{S} \to \mathcal{A}$ for the given state-action value function is given by $\pi(s) = \arg\max_a Q(s, a) \, \forall \, s$. Most Q-learning methods utilize two parameters for learning, namely α, the learning rate, and γ, the discount factor. Q-learning employs incremental estimation of Bellman equation for dynamic programming such that in the limit, the equality is satisfied. The Bellman equation for Q-learning is:

$$\begin{aligned} Q(s,a) &= R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s') \\ &= R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a') \end{aligned}$$

The temporal difference (TD) update for Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

implying a complexity per state-action update of $\mathcal{O}(|\mathcal{A}|)$.

Sarsa Another prominent Q-learning algorithm is the Sarsa algorithm, which was first proposed by Rummery and Niranjan (1994) and derives its name from requiring the tuple (s, a, r, s', a') to compute the update. Sarsa uses the next state s' and the next action a' to remove the 'maximization over all actions' calculation in the Q-learning TD update. From this, it is straightforward to show that the TD update for Sarsa is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q(s',a') - Q(s,a) \right)$$

implying a complexity per state-action update of $\mathcal{O}(1)$.

Eligibility Traces Both Q-learning and Sarsa can be modified with eligibility traces (often called Q(λ) and Sarsa(λ)). Eligibility traces employ an additional exponential decay parameter λ to recursively assign partial credit proportional to λ^t backwards along the state-action history (the eligible trace), which can help speed learning in both complex and sparse-reward environments.

Model Pseudocode

The core Sarsa-based solving algorithm for the overarching MMDP (i.e., across all agents $i \in 1, ..., n$) used is illustrated in Algorithm 2. At each time step, we call the generative model for each fare class and add their contributions to get the next state (v', t') and current reward r, then generate the next action a' and use the Sarsa update rule.

In line 10, we check to see if we have oversold our stock of tickets. If so, we correct it in line 11 by simulating selling the remaining tickets to customers on a first-come, first-served basis. E.g., if 15 customers wanted to buy from a remaining pool of 5 tickets, we would randomly pick 5 of the customers, independent of fare class, to obtain those tickets.

Algorithm 2 SOLVEMMDP

$C_i \leftarrow \emptyset, \forall i \in 1, ..., n$
$(v, t) \leftarrow (V, 1)$
$r \leftarrow 0$
Initialize $Q(s, a) = 0, \forall s \in S, a \in A$
Select a using ϵ-greedy with Gaussian randomness
repeat
  for $i \in 1, ..., n$ **do**
    $(v', t'_i), u'_i, r_i \leftarrow \text{GENERATIVEMODEL}_i((v, t), a)$
  end for
  if $\sum_i u'_i > v$ **then**
    Randomly choose values for the u'_i so that $\sum_i u'_i = v$
  end if
  $(v', t') \leftarrow (v - \sum_i u'_i, t + 1)$
  $r \leftarrow \sum_i a_i \cdot u'_i$
  Select a' using ϵ-greedy with Gaussian randomness
  $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q((v', t'), a') - Q(s, a))$
  $s \leftarrow (v', t'), a \leftarrow a'$
until $t = T$ or $v' = 0$

Experimental Setup

We performed two sets of experiments, namely solving for the optimal policy for a single-segment case (just class 1) and a multi-segment case with two fare classes (class 1 & class 2). The class parameters were constant throughout the experiments and are listed in Table 1.

Table 1: Fare Class Parameters

Parameter – Description	Class 1	Class 2
α – cust. arr. rate slope	-1	-1
β – cust. arr. rate intercept	30	20
μ – WTP threshold mean	700	300
σ – WTP threshold SD	100	50
a – WTP flexibility LB	20	10
b – WTP flexibility UB	20.1	10.1
$\underline{p}$ – price LB	550	225
$\overline{p}$ – price UB	850	375
$\tilde{p}$ – price interval	75	37.5

The rationale for choosing these values was to create a small but realistic example. The customer arrival slope α is negative to indicate decreasing demand over time. The WTP values characterize two distinct price points that would represent significantly different customer segments. The prices that the airline would set (determined by $\underline{p}, \overline{p}$ and $\tilde{p}$) were centered around the WTP threshold mean value μ with five possible prices to choose from.

For each problem, both Sarsa and Sarsa(λ) were performed to compute optimal policies and were compared against three optimal policies learned with specific action policies: a static low policy (offering the lowest price at all times for all fare classes), a static high policy (offering the highest price at all times for all fare classes), and a random policy (always offering a random price from the action space). The parameters governing the two problems are given in Table 2. The problem is undiscounted ($\gamma = 1$) since the horizon is finite.

Table 2: Solver Parameters

Parameter	Single-Segment	Multi-Segment
Time horizon	20	20
Tickets available	300	500
Learning rate	0.1	0.2
Discount factor	1	1
Eligibility parameter	0.9	0.9
Training episodes	250,000	50,000
Exploration strategy	ϵ-greedy	ϵ-greedy
ϵ (for expl. strategy)	0.2	0.2

The values for the time horizon and number of tickets available were chosen to keep the example small but realistic, while the values used for running the algorithm were chosen from trial-and-error tuning.

Results

The average rewards for executing the baseline optimal policies and the optimal policies for each of the fare classes is tabulated in Table 3.

Table 3: Fare Class Optimal Policy Average Rewards

Policy	Single-Segment	Multi-Segment
Static low	165,000	217,684
Static high	75,424	92,776
Random	188,729	250,003
Sarsa	190,758	239,400
Sarsa(λ)	196,876	235,700

In general, the random policy outperforms all of the other baseline optimal policies. For the single-segment case, Sarsa and Sarsa(λ) both outperform a random policy, and moreover, Sarsa(λ) outperforms Sarsa for the same number of training episodes, indicating a more efficient learning procedure.

The following figures show the optimal pricing policies (blue-to-white scale) or optimal value functions (yellow-to-purple scale) over the state space (the Cartesian product of tickets available and time). A more comprehensive collection of figures (illustrating all combinations of graphs and cases) is given in the appendix.

For the single-segment case, we observe in figures 6 and 7 that the algorithms learn optimal policies that hold out selling tickets right to the end of the time horizon (here, 20 days), as would be expected to maximize the opportunities to sell tickets at high prices. We also observe that the left side of the curves are darker (indicating higher prices) than their right sides, which show that if tickets are selling well, the price should be increased to exploit the demand and customers' willingness to pay, which is also the pricing behavior we would expect. Also, figure 7 has a broader footprint in the state space than figure 6, showing that Sarsa(λ) was able to learn from its explorations more quickly than Sarsa in this case.

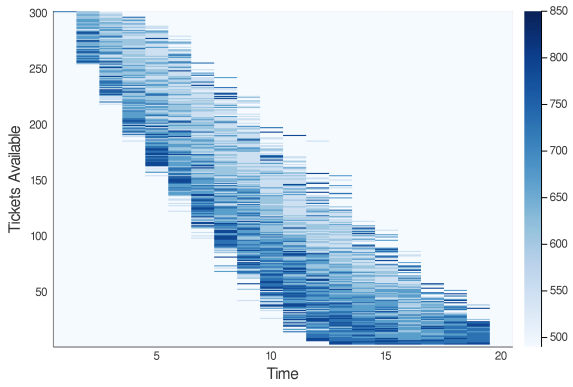

Figure 6: Optimal pricing policies in dollars ($) for the single-segment case (fare class 1) trained using Sarsa.

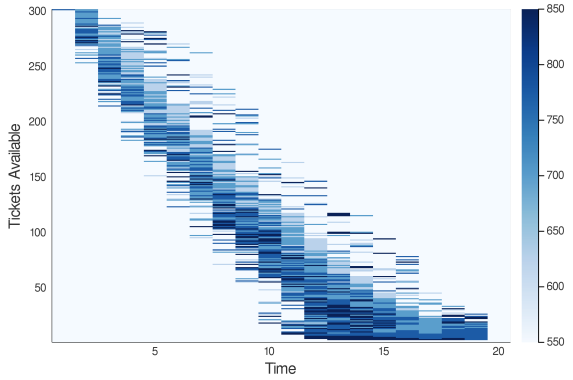

Figure 7: Optimal pricing policies in dollars ($) for the single-segment case (fare class 1) trained using Sarsa(λ).

Figure 8 shows the optimal value functions of the static low, random and static high benchmark action policies. These clearly indicate the portion of the state space explored due to constraints on their action policies. The random policy performed the best as it was able to explore the most pertinent part of the state space, while the static low policy performed better than the static high policy as the simulated demand was more consistent in purchasing at the lower price to the extent that it resulted in a higher total revenue. Figure 9 shows how the random policy also learned to price higher on the left side of its curve like in the Sarsa and Sarsa(λ) cases.

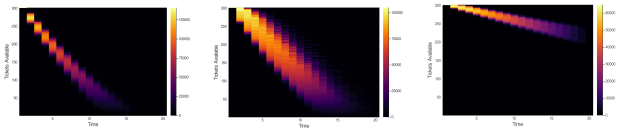

Figure 8: Optimal value functions for the single-segment case (fare class 1) using the static low (*left*), random (*center*), and static high (*right*) policies.

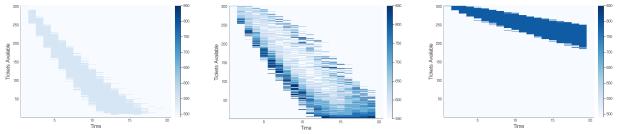

Figure 9: Optimal policies learned for the single-segment case (fare class 1) using the static low (*left*), random (*center*), and static high (*right*) policies. The static low graph appears uniform due to the lowest price being selected throughout the whole state space.

For the multi-segment case, the random policy (figure 10) outperforms the optimal policies generated by Sarsa and Sarsa(λ), which appear noisy either due to unstable learning or a lack of convergence. The random policy in this case performs similarly to the single-segment case, learning a conveniently optimal policy for the agent that can sell the most expensive tickets for class 1, and learning a truly random policy for class 2.

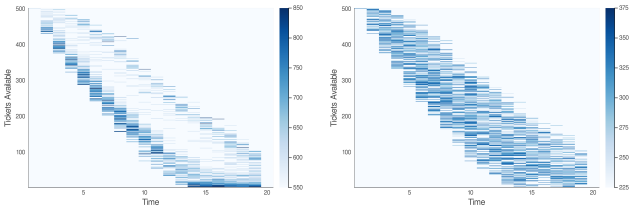

Figure 10: Optimal policies learned in the multi-segment case with the random policy for class 1 (*left*) and class 2 (*right*).

Conclusion

We introduced a factored multi-agent Markov decision process (MMDP) as a decision-theoretic approach to solving a multi-fare dynamic pricing problem, and utilized model-free reinforcement learning algorithms to determine an approximately optimal policy for a specified set of fare classes. We demonstrated improvement over static and random default policies for a single-agent case and demonstrated improvement over static and random default policies in only some cases due to limited training time.

Further work could involve modifications to both the customer dynamics and learning frameworks. For example, the mean of the customers' willingness-to-pay threshold could be modeled to increase with time. Also, more complex learning frameworks could be used such as deep Q-learning (DQN) to globally approximate the state-action value function and thus improve generalization of the joint policy. Also, the set of prices employed could be expanded to the 26 price points used in the industry, or for a more generalized case, a fine-resolution discrete action set or continuous action space could be used. Another modification could be to implement learning rate and epsilon-greedy annealing to stabilize the learning process and make the computation more efficient.

Additionally, the problem could be framed as a decentralized Markov decision process (DEC-MDP), where agents cannot communicate for free and the joint state is only jointly fully-observable, or the problem could be framed as a decentralized partially-observable Markov decision process (DEC-POMDP), where the individual state is partially-observable. These frameworks can improve the generalization of the joint policy to a wider variety of scenarios.

References

[2019] Abdella, J. A.; Zaki, N.; Shuaib, K.; and Khan, F. 2019. Airline ticket price and demand prediction: A survey. *Journal of King Saud University - Computer and Information Sciences*.

[2013] Beynier, A.; Charpillet, F.; Szer, D.; and Mouaddib, A.-I. 2013. DEC-MDP/POMDP. In *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons, Ltd. chapter 9, 277–318.

[1999] Boutilier, C. 1999. Sequential optimality and coordination in multiagent systems. In *IJCAI International Joint Conference on Artificial Intelligence*.

[2003] Carvalho, A. X., and Puterman, M. L. 2003. Dynamic Pricing and Reinforcement Learning. In *Proceedings of the International Joint Conference on Neural Networks*.

[2016] Competitoor. 2016. Dynamic Pricing: How It Works and How Much It's Used in 2016.

[2015] den Boer, A. V. 2015. Dynamic pricing and learning: Historical origins, current research, and new directions.

[2004] Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. *AAAI Workshop - Technical Report* WS-04-08(2000):25–30.

[1987] Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*.

[2006] Raju, C. V. L.; Narahari, Y.; and Ravikumar, K. 2006. Learning dynamic prices in electronic retail markets with customer segmentation. *Annals of Operations Research* 143(1):59–75.

[2014] Rana, R., and Oliveira, F. S. 2014. Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning. *Omega (United Kingdom)* 47:116–126.

[1994] Rummery, G. A., and Niranjan, M. 1994. *Online Q-Learning Using Connectionist Systems*. Ph.D. Dissertation, Cambridge University.

[2018] Touraine, S., and Coles, H. 2018. Dynamic Offer Creation White Paper. Technical Report October, International Air Transport Association, Geneva.

Appendix

Source Code

The source code used for this paper is available at https://github.com/rbalexan/aa-228/tree/master/final-project.

Graphs

The following graphs show the optimal value functions and optimal pricing policies calculated as an extension to the Results section.

Single-Segment Case

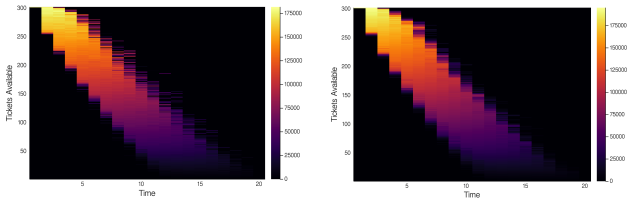

Figure 11: Optimal value functions for the single-segment case (fare class 1) trained using Sarsa (*left*) and Sarsa(λ) (*right*).

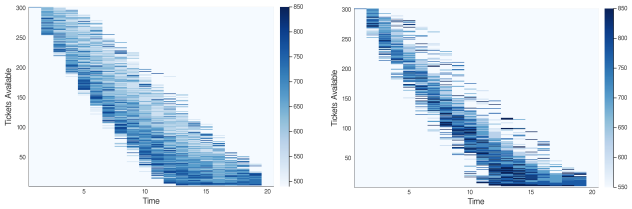

Figure 12: Optimal pricing policies in dollars ($) for the single-segment case (fare class 1) trained using Sarsa (*left*) and Sarsa(λ) (*right*).

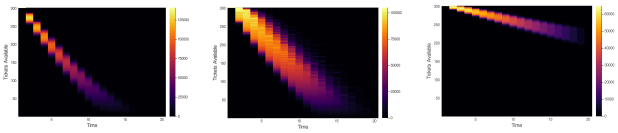

Figure 13: Optimal value functions for the single-segment case (fare class 1) using the static low (*left*), random (*center*), and static high (*right*) policies.

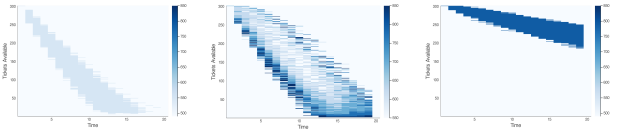

Figure 14: Optimal policies learned for the single-segment case (fare class 1) using the static low (*left*), random (*center*), and static high (*right*) policies.

Multi-Segment Case (for Two Classes)

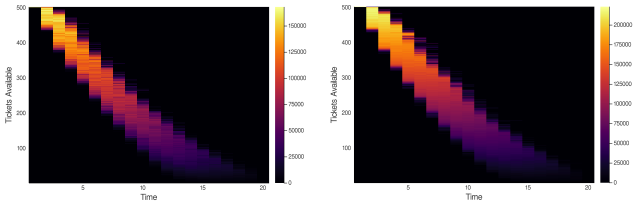

Figure 15: Optimal value functions for the multi-segment case trained using Sarsa (*left*) and Sarsa(λ) (*right*).

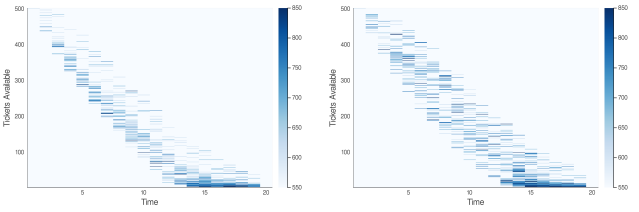

Figure 16: Optimal pricing policies for the first fare class in dollars ($) for the multi-segment case trained using Sarsa (*left*) and Sarsa(λ) (*right*).

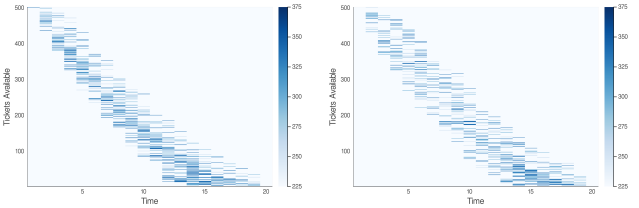

Figure 17: Optimal pricing policies for the second fare class in dollars ($) for the multi-segment case trained using Sarsa (*left*) and Sarsa(λ) (*right*).

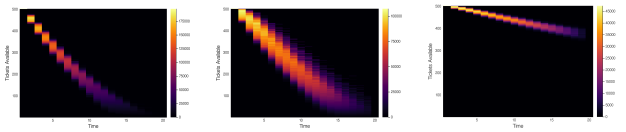

Figure 18: Optimal value functions for the multi-segment case using the static low (*left*), random (*center*), and static high (*right*) policies.

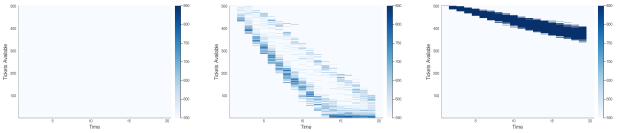

Figure 19: Optimal policies learned for the first fare class in the multi-segment case using the static low (*left*), random (*center*), and static high (*right*) policies.

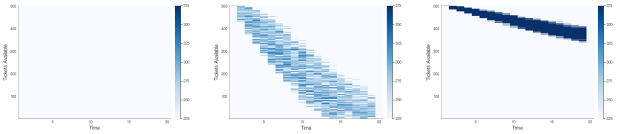

Figure 20: Optimal policies learned for the second fare class in the multi-segment case using the static low (*left*), random (*center*), and static high (*right*) policies.