Reinforcement learning applied to airline revenue management

Giovanni Gatti Pinheiro

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Apprentissage par Renforcement appliqué au Revenue Management des compagnies aériennes

Giovanni Gatti Pinheiro

Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis

Présentée en vue de l'obtention du grade de docteur en informatique **d'**Université Côte d'Azur
Dirigée par : Jean-Charles Régin
Co-encadrée par : Michel Defoin-Platel
Soutenue le : 8 septembre 2022

Devant le jury, composé de :
A. Dutech, Chargé des recherches, INRIA
B. Scherrer, Chargé des recherches, INRIA
G. Gallego, Professor, Hong Kong University of Science and Technology
J. Martinet, Professeur, Univ. Côte d'Azur
J.-C. Régin, Professeur, Univ. Côte d'Azur
M. Defoin-Platel, Head of MLI, Amadeus SAS

amadeus

Reinforcement Learning applied to airline Revenue Management

Apprentissage par Renforcement appliqué au Revenue Management des compagnies aériennes

Jury :

Président du jury :
Jean Martinet, Professeur des universités, Université Côte d'Azur

Rapporteurs :
Alain Dutech, Chargé des recherches, INRIA
Bruno Scherrer, Chargé des recherches, INRIA
Guillermo Gallego, Department Head & Chair Professor, Hong Kong University of Science and Technology

Examinateurs :
Michael Defoin-Platel, Doctor & Head of Machine Learning and Innovation, Amadeus SAS

Directeur :
Jean-Charles Régin, Professeur des universités, Université Côte d'Azur

Invités :
Michael D. Wittman, Doctor & Researcher, Amadeus IT Group
Thomas Fiig, Doctor & Chief Scientist, Amadeus IT Group

Résumé

Inspiré par les récentes réussites de l'apprentissage par renforcement (RL), telles que le contrôle des champs magnétiques d'un tokamak ou l'obtention de performances supérieures à l'humain au jeu de Go et aux échecs, ce travail étudie des moyens d'appliquer le RL au domaine de recherche du *revenue management* (RM).

Il existe de nombreux problèmes ouverts dans le domaine du RM qui reposent sur des heuristiques conçues par des experts. Cependant, ces heuristiques sont souvent difficiles à développer et à maintenir, et elles ne capturent qu'une fraction limitée des complexités inhérentes aux scénarios réels. Au lieu de cela, nous proposons d'aborder ces problèmes ouverts avec des méthodes RL génériques.

Afin d'illustrer l'argument central de ce travail, nous choisissons d'aborder le problème du *earning while learning* (EWL), qui est l'un des nombreux problèmes ouverts dans le domaine du RM. Pour résoudre le problème du EWL, le système doit maximiser les revenus à long terme en optimisant les prix des billets d'avion tout en faisant face à un comportement de demande inconnu. Autrement dit, le système n'a pas accès à la façon dont la demande future réagira aux changements de prix, et il doit estimer le comportement de la demande à partir des données de réservation passées.

Nous formalisons et concevons des interactions entre le système d'apprentissage et le comportement inconnu de la demande, puis un réseau de neurones artificiels (ANN) est entraîné avec un algorithme *actor–critic* pour résoudre le problème EWL. Le ANN entrainé, que nous appelons l'agent RL, réalise une meilleure performance de revenus que les méthodes heuristiques de pointe. Ensuite, nous discutons des moyens d'adapter l'agent RL pour résoudre d'autres problèmes pertinents dans RM, tels que la concurrence, l'auto-concurrence et le comportement non stationnaire de la demande. L'approche proposé met en évidence que la simplicité de l'agent RL, ainsi que sa capacité à résoudre des problèmes complexes, est un puissant atout inexploré.

Les découvertes présentées dans ce travail suggèrent que le RL peut profondément transformer le domaine de RM. Les méthodes RL peuvent remplacer les systèmes RM par un système d'apprentissage générique de bout en bout qui est entraîné en *offline* pour résoudre de nombreux problèmes complexes. Par conséquent, les chercheurs et les analystes du RM n'auraient qu'à décrire formellement le comportement de la demande utilisé pour l'entraînement et à déléguer la recherche de solutions à l'agent RL. Ceci modifie l'orientation de la recherche de comment résoudre les problèmes vers quels problèmes doivent être résolus. Nous croyons qu'un tel changement de perspective peut accélérer considérablement le processus de recherche dans le domaine du RM.

Mots clés— revenue management, apprentissage par renforcement, apprentissage profond, actor–critic, demand learning

Abstract

Inspired by the recent successful applications of reinforcement learning (RL) to real–world problems, such as controlling magnetic fields of a tokamak or achieving superhuman performance in board games, this work investigates ways to apply RL to the research field of revenue management (RM).

There are many open problems in the field of RM that rely on expert–designed heuristics. However, these heuristics are often hard to develop and maintain, and they only capture a limited fraction of the inherent complexities of real–world scenarios. Instead, we propose addressing these open problems with generic RL methods.

To illustrate the central argument of this work, we choose to address the challenging earning–while–learning (EWL) problem, which is one of the many open problems in the field of RM. When addressing the EWL problem, the system needs to maximize long–term revenue by optimizing the prices of airline tickets while facing an unknown demand behavior. In other words, the system does not have access to how future demand will react to price changes, and it must estimate the demand behavior from past booking data.

We formalize and design interactions between the learning system and the unknown demand behavior, then an artificial neural network (ANN) is trained with an actor–critic algorithm to solve the EWL problem. The trained ANN, which we refer to as the RL agent, achieves a better revenue performance than state–of–the–art heuristic methods. Following, we discuss ways to adapt the RL agent to address other relevant issues in RM, such as competition, self–competition, and non–stationary demand behavior. The proposed approach highlights that the simplicity of the RL agent, alongside its ability to address complex issues, is a powerful unexplored asset.

The discoveries presented in this work suggest that RL can profoundly transform the field of RM. RL methods may replace RM systems with a generic end–to–end learning system trained offline to address many complex issues. Consequently, RM researchers and analysts would need only to describe formally the demand behavior used for training, and delegate the search for solutions to the RL agent, changing the research focus from *how* to solve problems to *what* problems need to be solved. We believe that such a change of perspective can significantly speed up the research process in the field of RM.

Keywords— revenue management, reinforcement learning, deep learning, actor–critic, demand learning

Intelligence is the computational part of the ability to achieve goals. A goal achieving system is one that is more usefully understood in terms of outcomes than in terms of mechanisms.

— Richard S. Sutton*

* When discussing the John McCarthy's definition of intelligence.

Acknowledgments

I want to thank my PhD advisor, Prof. Jean Charles-Régin, for his guidance and support. I also would like to thank Alain Dutech, Bruno Scherrer, Guillermo Gallego, and Jean Martinet for agreeing to be on my thesis jury and for their interest in my work.

I gratefully acknowledge the funding provided by Amadeus over the many years of research through the French CIFRE program.

My special thanks to Dominique Pouchoulin, a non-RL and non-RM specialist, who bravely agreed to proofread the manuscript and provided essential feedback on the accessibility of the material.

Many people crossed my research over the years and inevitably impacted this work. I want to thank you all: Oksana Riou, Riccardo Jadanza, and Abderrahim Mehdaoui. I also give a warm thank you to all the Amadeus research team, especially to Mike Wittman, which whom I had fruitful discussions that led to insights into the many ideas presented in this thesis.

This research would not be possible without the support of my loving wife, Audrey, who encouraged me to pursue my dreams with no hesitation.

I save the last and likely the most important thank you to Michael. During these past years, Michel was to me what Obi-Wan was to Luke: a mentor, a supporter, and a friend. He made it all possible.

Contents

List of Figures

List of Tables

Preface

Reinforcement learning (RL) first came to the attention of Amadeus researchers after the great successes in Atari and Go, and the earliest experiments applying RL methods to revenue management (RM) performed by our team raised mixed feelings. Initially, the team believed that the only advantage of RL was its ability to optimize pricing policies without building explicit demand models, as is the current practice in modern revenue management systems. In principle, this alternative model–free approach to RM would enable the system to adapt to the complexities of human behavior. However, the team found that RL may not be suited for RM applications, mainly because it needs large amounts of data that are perhaps not available for real–world use cases. Thus, the most important conclusion of this early research was that more investigation was needed to evaluate the full potential of RL applied to RM.

RM and RL are both active fields where discoveries are made almost daily, but, to our surprise, the research on the intersection of these two fields is just in its infancy, and there are many unexplored opportunities. During the past years, we interacted with several researchers in RL and RM, and we experienced that many misunderstandings of each other's fields are slowing down progress. For this reason, much of our time was dedicated to clarifying the concerns of each research field and explaining what the open challenges are.

We see this dissertation as a consolidation of this work. Our goal is to create a bridge between these two research fields so that RM and RL researchers understand each other and work together. For this reason, when writing this dissertation, we assume that a typical reader may have little knowledge about RM or RL. Therefore, Chapters 1, 2, and 3 focus on clarifying the basics in a language that can be easily understood by researchers from any of these fields. We propose along the way short exercises to demystify and illustrate the mathematical concepts that, from our experience, can sometimes be a barrier to entry. Unfortunately, we cannot explain in detail the history and the mathematics of all components of our work. For this reason, we pay less attention to deep learning literature, presenting artificial neural network concepts only at the highest level to facilitate understanding for non–specialists.

After setting up the basic concepts, we present the core of our research. We illustrate how RL can change the way researchers think and solve problems by using it to address one of the most challenging open problems in RM. Finally, we conclude with our current vision of the impacts RL can bring to the field of RM.

Summary of Notation

Capital letters are used for random variables, whereas lower case letters are used for real–valued quantities and scalar functions. Real–valued vectors are written in bold and lower case.

$\doteq$	equality relationship that is true by definition
$\approx$	approximately equal
$\gg$	much greater than
$\leftarrow$	assignment
$x \mod y$	returns the remainder of a division x/y
$\arg\max_a f(a)$	value a at which $f(a)$ assumes the maximal value
$x!$	factorial operator
$\Pr\{X = x\}$	probability that random variable X assumes value x
$\Pr\{X\}$	probability of observing event X
$\Pr\{X \mid Y\}$	probability of observing event X given the condition Y
$\mathbb{E}[X]$	expectation of a random variable X
$\mathbb{E}[X \mid Y]$	expectation of a random variable X given the condition Y
$\mathrm{Var}[X]$	variance of a random variable X, i.e., $\mathbb{E}[(X - \mathbb{E}[X])^2]$
$\mathrm{U}(a, b)$	uniform distribution with closed interval $[a, b]$
$\mathbb{1}\{\cdot\}$	indicator function, i.e., returns one if the predicate is true, else zero
$\Gamma(x)$	Gamma function
$\Gamma(x, z)$	incomplete Gamma function
$O(\cdot)$	big O notation (limiting behavior of a function when the argument tends towards infinity)
$f : \mathcal{X} \to \mathcal{Y}$	function f from elements of set $\mathcal{X}$ to elements of set $\mathcal{Y}$
$f(x; \omega)$	function of x parameterized by ω
$\nabla f(x; w)$	gradient of function $f(x)$ with respect the parameter w
$X \sim p$	random variable X selected from distribution $p(x) \doteq \Pr\{X = x\}$
$X_0, \cdots, X_{n-1} \sim p$	n random variables selected from the distribution $p(x)$
$X_{t:t+n} \sim p$	$n+1$ random variables selected from the distribution $p(x)$ from time step t to time step $t+n$
$\mathbf{x} \cdot \mathbf{y}$, $\mathbf{x}^{\mathrm{T}}\mathbf{y}$	inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$
$\|\mathbf{x}\|$	Euclidean norm (or, L2 norm) of vector $\mathbf{x}$, i.e., $\sqrt{\mathbf{x} \cdot \mathbf{x}}$
$\mathbb{N}_0$	set of natural numbers, including zero
$\mathbb{R}$	set of real numbers
$\mathbb{R}_+$	set of positive real numbers including zero, i.e., $\{x \in \mathbb{R} \mid x \geq 0\}$

$\mathbb{R}^n$	real coordinate space of dimension n, i.e., the set of n-tuples of real values
T	total time steps to departure of a flight (also the number of active flights in the horizon)
C	flight's total capacity
H	episode horizon
f, f_i	fare (e.g., \$50, \$70)
t	discrete time step
τ	time steps to departure
t	departure date
c	remaining capacity
$d(f;\boldsymbol{\psi})$	demand model that returns the expected number of customers willing to pay for fare f
$u(x)$	probability of observing x customer arrivals according to the Poisson distribution with constant mean
$z(f;\phi), z(f), z$	customer purchase probability
f_0	lowest fare which the purchase probability is one
$o(t,f)$	the number of times fare f is offered across all flights at sale date t
$b(t,f)$	the number of bookings made for fare f at sale date t across all flights
$S_t^{[i]}$	inventory state at time t for the i-th active flight in the horizon
$A_t^{[i]}$	action (fare) at time t for the i-th active flight in the horizon
$\boldsymbol{\psi}, \boldsymbol{\psi}_*$	generic set of parameters of the demand model
ν, ν_*	arrival rate parameter
ϕ, ϕ_*	arrival rate parameter
F_5, F_5^*	the fare ratio of the lowest fare at which the purchase probability is 50% (alternative representation of the price sensitivity parameter)
$\mathcal{S}$	set of all states (including the terminal state)
$\mathcal{A}$	set of all available actions
S_t	state at time t
A_t	action at time t
R_t	reward at time t
S_0	initial state
S_T	terminal state
$p(s' \mid s, a)$	probability of transition to state s' from state s taking action a
$r(s,a)$	expected immediate reward from state s after taking action a
G_t	return following time t
π	policy function (decision–making rule)

π_*	optimal policy function
$\pi(s)$	the action to be taken deterministically in state s
$\boldsymbol{\pi}$	pricing policy as a multinomial distribution
$\pi(a \mid s; \boldsymbol{\theta})$	probability of taking action a in state s given parameter vector $\boldsymbol{\theta}$
$v_\pi(s)$	value of state s under policy π (expected return)
$q_\pi(s, a)$	value of taking action a in state s under policy π
v_*	optimal value function
q_*	optimal action–value function
$\mathbb{E}_\pi[X]$	expectation of random variable X under policy π
$\hat{\mathbb{E}}_\pi[X]$	empirical average of random variable X under policy π
$V(s)$	tabular estimation of the value for state s
$Q(s, a)$	tabular estimation of the action–value for state–action pair s, a
δ_t	temporal–difference error at t (random variable)
$q(s, a; \mathbf{w})$	approximate value of state–action pair s, a given weight vector $\mathbf{w}$
$\mathbf{x}(s, a)$	the vector of features visible when in state taking action a
$h(s, a; \boldsymbol{\theta})$	preference of selecting action a in state s based on $\boldsymbol{\theta}$
$\mu(s)$	on–policy distribution over states
γ	discount rate parameter
α	learning rate parameter
ϵ	probability of taking a random action when following an ϵ-greedy policy
η	the trade–off parameter

Introduction 1

In this chapter, we briefly discuss what revenue management is, its many practical aspects, building blocks, and open problems. Among these open problems, we select one that will be the focus of our work, and we derive the problem assumptions used throughout the following chapters.

1.1 Welcome to revenue management

Anyone selling a product or a service has many decisions to make. A grocery store selling food needs to decide what products to sell (e.g., fruits and vegetables), how often to resupply these products to the store, and how to organize products on shelves. Likewise, an airline company selling tickets has to decide where and when to fly, how much to price for its services, how many seats should be available per cabin (e.g., first, business and economic classes), and which aircraft model to buy (e.g., Airbus A380), to name few. Some of these decisions relate to *supply–chain management*, i.e., how to deliver a product or a service to customers (e.g., resupply products in a grocery store). Other decisions, known as *demand–management* decisions, are concerned with planning and managing the demand for products and services (e.g., how to price products) [1].

[1]: Talluri and Van Ryzin (2004), *The theory and practice of revenue management*

A "rational" seller aims at maximizing profits, which can be achieved by maximizing revenue while minimizing costs. Minimizing costs is the goal behind supply–chain management while maximizing revenue is the goal behind demand management. Revenue management (RM) is the research field concerned with the methodology, the processes, and the technology necessary for making such demand–management decisions [1].

Even though the theory and practice of RM are, in some sense, as old as trade itself, RM is relatively new as a scientific field. In 1978, the United States enacted the Airlines Deregulation Act, later followed by European states, giving airline companies control over routes, schedules, and fares. The deregulation led to rapid changes in the industry and increased pressure for innovation, giving rise to the modern field of RM. Over the following years, the RM practices developed within the airline industry were adopted quickly by other industries such as retail, car rental, and hotel, among many others [1].

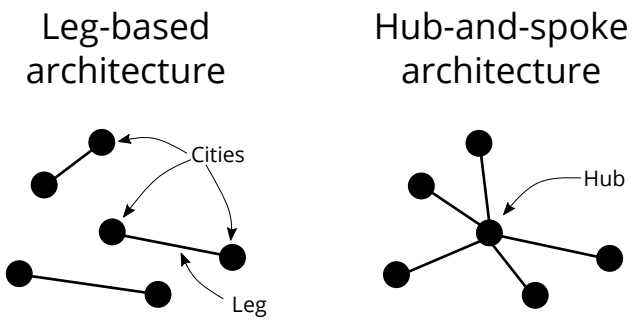


Figure 1.1: Leg–based vs. hub-and-spoke architectures. Each point represents a city (or, an airport), and each segment connecting two cities are legs. There may be many flights through each leg, often departing with a particular frequency (e.g., daily flights). **(Left)** The leg–based architecture uses a point–to–point approach. **(Right)** The hub–and–spoke architecture directs all traffic to a larger city when connecting to smaller cities.

The science and methods behind RM are constantly evolving to adapt to new technologies, customer behavior, and market conditions. The first revenue management systems (RMSs) emerged in the 1980s, and they performed revenue maximization by controlling the yield of each leg separately (also known as leg control; see Figure 1.1 left). By the 1990s, airlines started operating in hub–and–spoke models, in which flights between two smaller cities (spokes) connect through a large city (hub). Thus, demand for spoke–hub and spoke–spoke overlap, requiring RMSs to predict demand and optimize decisions at the network level (also called network control; see Figure 1.1 right). In the early 2000s, the popularization of the Internet brought customers much greater transparency over airlines' offers as they could compare offers from a wide range of options. At the same time, the Internet also enabled the possibility of updating prices worldwide almost instantaneously while accurately keeping track of sales and inventory, allowing airlines to react to market changes with unprecedented precision and speed. In the 2010s, the financial crisis and the consolidation of the Internet as a distribution channel boosted low–cost carriers that operated with the emphasis on minimizing costs by eliminating traditional services and amenities, causing a "commoditization" of the industry [2]. However, history never stops, and it is hard to foresight the next revolution. In response to the commoditization, will the airline industry move towards personalized offers with merchandising [3]? Or, will innovative ideas such as inspiring customers to their next travel destination change their relationship with airlines [4]? Or perhaps, as we discuss in the following chapters, the next revolution may come from within the RMS itself.

Throughout this work, we focus on a specific facet of RM, where the airline company's flights have a fixed *inventory capacity* (i.e., the number of seats available for bookings in each flight), and it seeks to control this capacity through pricing decisions to maximize revenue. The inventory is perishable (i.e., flight departs), and any unsold capacity at the expiration date (i.e., flight departure) is lost. The airline's prices are governed solely

[2]: Fiig, Cholak, et al. (2015), "What is the role of distribution in revenue management?–Past and future"

[3]: Dadoun, Defoin-Platel, et al. (2021), "How recommender systems can transform airline offer construction and retailing"

[4]: Dadoun, Troncy, et al. (2021), "Predicting your next trip: A knowledge graph-based multi-task learning approach for travel destination recommendation"

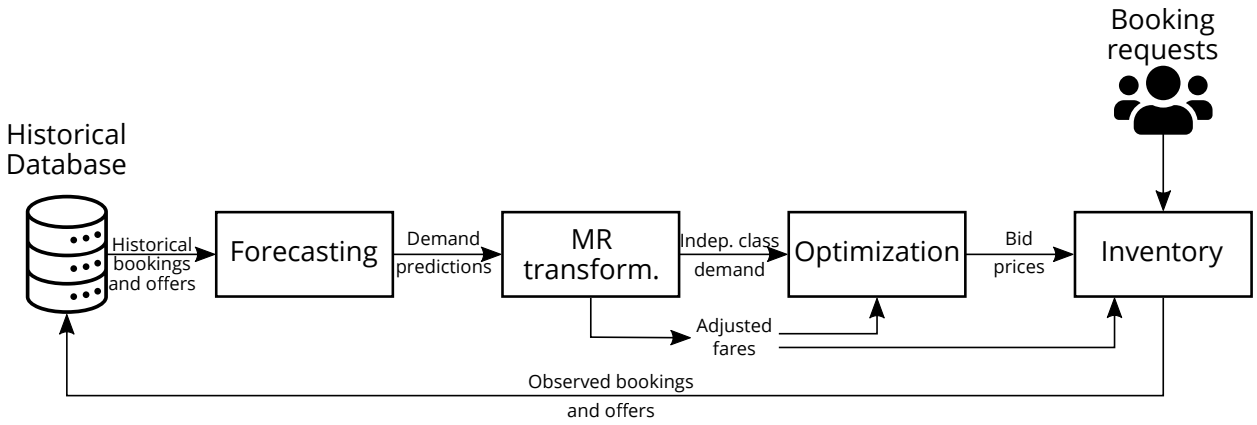


Figure 1.2: The revenue management system in a nutshell. "MR transform." abbreviates "Marginal revenue transformation."

by the forces of supply and demand: The airline must find a balance between pricing too high and losing potential buyers and pricing too low and losing potential profits.

1.2 Airline revenue management systems in a nutshell

The fundamental design of modern RMSs is divided into several distinct components, as illustrated in Figure 1.2 [5]. We refer to the first component as the *historical database*, which stores the booking data and the offers for every flight operated by the airline. This data contains the daily number of bookings observed for each offer made by the system. The historical booking data feeds the *forecasting* module, which predicts how future demand will respond to prices, i.e., how much more/less demand the system can expect to observe by increasing/decreasing prices. These forecasting predictions assume demand independence between classes of customers (e.g., business or leisure travelers), with no possibility for customers to choose a lower fare when a specific set of restrictions are in place (e.g., minimum stay requirements, cancellation penalties, refundability conditions). In practice, the assumption of independent demand is not entirely true, possibly causing a "spiral down" of the demand into lower fare classes, resulting in significant revenue losses [6]. To address this issue, the *marginal revenue transformation* module maps conditional demand models, where customers are free to purchase other itineraries or lower fares with a different set of restrictions, into independent demand and adjusted fares, where customers are assumed to be willing to purchase only a particular product [7]. Using the adjusted demand and fares, the fourth component, called the *optimization* module, is responsible for optimizing the pricing policy according to capacity constraints. The optimization

[5]: Fiig, Härdling, et al. (2014), "Demand forecasting and measuring forecast accuracy in general fare structures"

[6]: Cooper, Homem-de-Mello, et al. (2006), "Models of the spiral-down effect in revenue management"

[7]: Fiig, Isler, et al. (2010), "Optimization of mixed fare structures: Theory and applications"

module outputs bid prices used in the last and final component, the *inventory* module, to evaluate customer purchase requests in real–time.

1.3 Open problems in revenue management

RM is an active area of research yet in expansion in many industries, such as cargo, restaurants, and special events [8–10]. There are still many problems to be solved, sometimes industry–oriented, and it is far beyond our scope to develop the complete anatomy of the field and its research prospects. Instead, we focus on a short list of issues, often seen as critical, that we believe could be addressed with the use of artificial intelligence (AI). These issues are pricing under competition, self–competition, earning while learning, model–free optimization, and non–stationarity.

[8]: Meng, Zhao, et al. (2019), "Revenue management for container liner shipping services: Critical review and future research directions"
[9]: Kimes and Ho (2019), "Implementing Revenue Management in Your Restaurants: A Case Study with Fairmont Raffles Hotels International"
[10]: Bouchet, Troilo, et al. (2016), "Dynamic pricing usage in sports for revenue management"

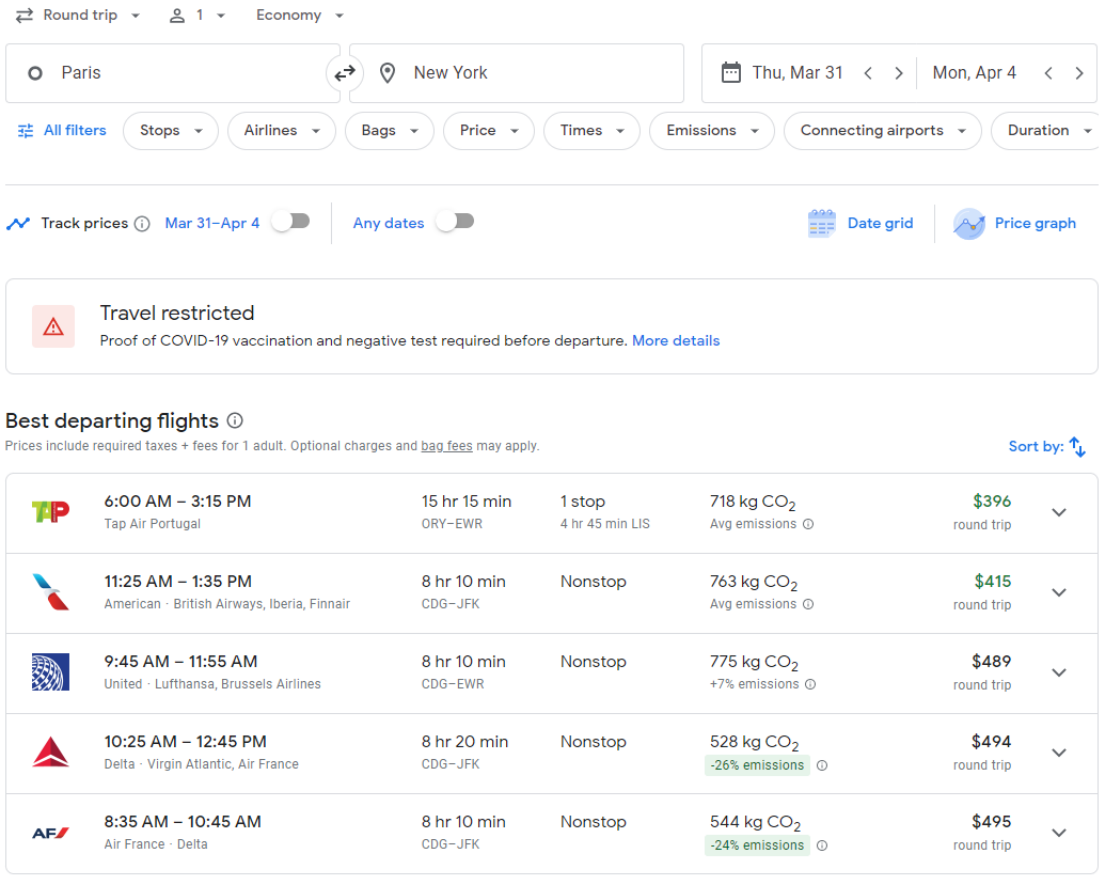
One of the most important open problems in airline RM is *pricing under competition* [11, 12]. When traveling between two cities, customers can choose from several options often offered by different airlines (see Figure 1.3 (a)). For example, the busiest route in the world in 2019 was between Seoul to Jeju, deploying about 17 million seats over the year with flights operated by ten airlines [13]. Therefore, airlines compete for the same demand by offering advantages over the others (e.g., price discounts). The most noticeable impact of competition is that the airline may lose potential customers and thus profits. Every unsold capacity eventually perishes, i.e., flight departs, and a revenue opportunity is lost. Even though decreasing prices may attract more demand, it may impact revenue negatively because less revenue is produced for each booking. A more subtle impact relates to data collection. In practice, as airlines only have access to their booking data (which does not contain no-purchase information), estimating the total demand volume is challenging. With the presence of a competitor, correctly estimating the demand behavior is even more difficult because the observed demand is now dependent on the competitor's prices (which may vary in time).

[11]: Fiig, Wittman, et al. (2019), "Towards a competitor-aware RMS"
[12]: Kumar, Wang, et al. (2021), "Competitive revenue management models with loyal and fully flexible customers"

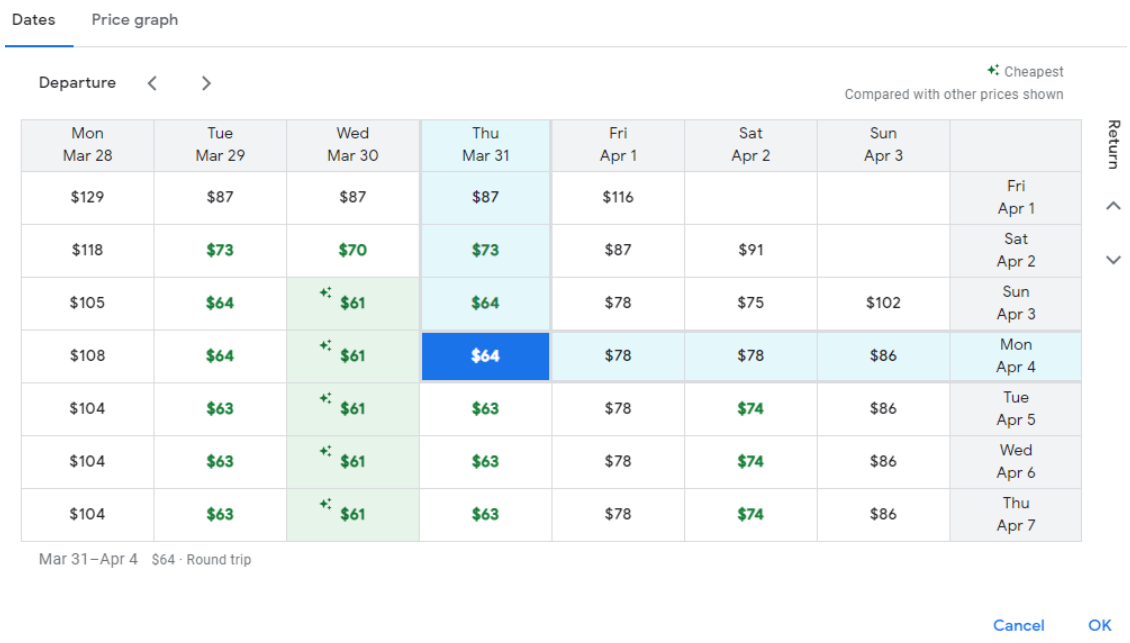
[13]: Pande (2020), *What are the world's busiest air routes right now?*

Perhaps another issue as important as the competition, but much less evident is the *self-competition* problem, sometimes referred to as *cannibalization* [14]. When booking, customers may adapt their preferences to the airline's offers (see Figure 1.3 (b)). For example, customers may prefer to travel at inconvenient hours, delay their departure, shorten their stay, or choose a longer itinerary to save money. As an airline usually provides several flights between two destinations, it may observe losses in revenue because its flights compete with each other for the same demand. This issue

[14]: Gallego and Phillips (2004), "Revenue management of flexible products"



(a) Example of competition: Many competing airlines operate from Paris to New York.



(b) Example of self–competition: A single airline operating flights from Nice to Paris, and, customers may decide to delay or anticipate their trip.

Figure 1.3: Screenshot of a flight search with Google Flights engine.

is only not challenging from the modeling perspective but also from the optimization perspective because optimizing the prices of many flights in a combined way is a complex task due to the large search spaces involved.

Another intriguing issue calling attention is known as the *earning–while–learning* problem [15, 16]. When optimizing pricing strategies, airlines do not know the true demand behavior, which must be estimated from historical booking data. Moreover, airlines' current pricing decisions impact the collected data and thus the quality of future estimations. In practice, the RMSs may need to compromise immediate revenue (earning) by performing price experiments hoping that the information gained about the demand behavior (learning) will lead to better future pricing decisions.

[15]: McLennan (1984), "Price dispersion and incomplete learning in the long run"

[16]: Chen and Gallego (2022), "A primal-dual learning algorithm for personalized dynamic pricing with an inventory constraint"

Furthermore, RMSs usually rely on expert–designed models describing how the demand behaves when facing choices (also known as customer choice models). However, human behavior is not necessarily logical and rarely deterministic, and therefore difficult to understand and model. One could wonder if we could eliminate the need for such demand models, discovering the pricing strategy exclusively from data without any explicit help from experts. In theory, the general nature of reinforcement learning allows the optimization of prices from direct interactions with the demand, i.e., *model–free optimization*. In practice, it is currently unclear if such an approach is feasible because interactions with the customers are expensive and limited, requiring systems to find near-optimal solutions from little data [17]. This *data efficiency* problem is central to applying AI techniques to RM.

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"

Lastly, the demand behavior is not static, as RM pricing optimization techniques often assume [18]. The general economy may present growth or crisis, new airlines may join or leave markets, and customer interests may shift from some destinations to others, impacting how much customers are willing–to–pay for the airline's services. Such changes in demand behavior may occur over short or long periods, or they may impact airlines locally (such as a new competitor in a segment) or globally (such as a pandemic). The airline's ability to quickly adapt pricing strategies to changing market conditions, or *non–stationary* demand behavior, is fundamental to its success (and survival).

[18]: Keskin and Zeevi (2017), "Chasing demand: Learning and earning in a changing environment"

1.4 Goals and assumptions

There are many open issues in airline RM, and we believe that AI has the potential to solve them. Because of their general nature, AI methods bring many advantages over classical techniques, mostly changing how we approach problems where the solutions are centered less on expert design and intuition and more on formulation and representation. Therefore, our primary goal is to demonstrate the power of AI and discuss how to use it to address the main problems of RM.

We cannot hope to address so many issues in detail, and thus we concentrate on one representative problem. Among the several candidate problems presented earlier, we prefer the earning while learning (EWL) one because many real–world complexities, such as competition, cannibalization, and non-stationarity, can be ignored. Indeed, our primary motivation for choosing this problem is that we can make simplifying assumptions that facilitate the implementation and debugging. Moreover, when considering only the EWL problem, we can compare the final learned solution to the optimal performance obtained when the true demand behavior is known at all times, providing an upper bound of performance improvement, easing the search for the learning system's design and interpretation of the results.

When studying the EWL problem, we assume that the company offers a single flight a day with a fare structure with no restrictions and a *monopoly* where the airline is the exclusive service provider. Furthermore, the airline provides the service over a *single leg* from point A to point B, meaning connections and round–trips are not possible. In other words, customers can only travel to a unique destination, and they can only decide to book or not based on the price chosen by the airline. We refer to such settings as the *single–leg problem*, which we develop further in Section 2.1. Even though such assumptions are not realistic, they are sufficient to display the problem properties we aim at. Yet, we believe that most of our work can be adapted to real–world systems, but we recognize that further research is needed to do so.

Under such assumptions, there is no need to represent explicitly many of the RMS components presented in Section 1.2. We consider a simplified theoretical view of RMS illustrated in Figure 1.4. This simplified structure consists of the historical database that stores historical bookings and offers for every flight from point A to point B, the forecasting module that predicts the future demand response to changes in prices, and the optimization module that computes the prices that maximize revenue.

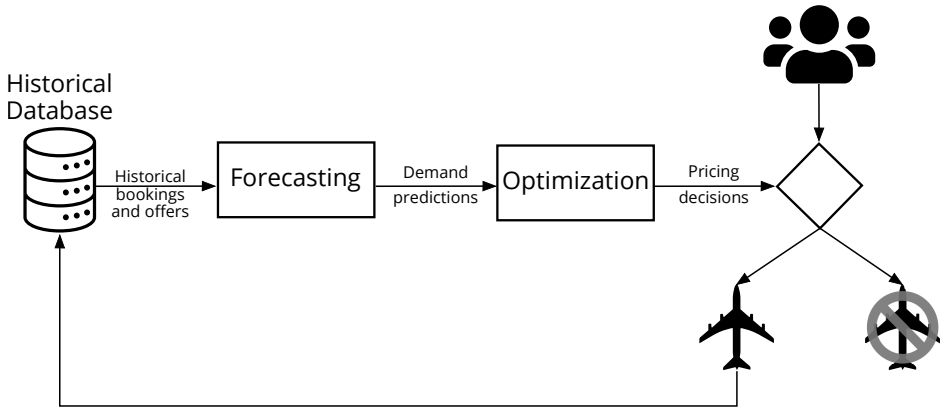


Figure 1.4: A simplified revenue management system. Image from [19] under license CC BY 4.0.

Customers interact with the offered prices by either choosing to book or not. The resulting bookings are stored in the historical database for future use (only purchase decisions are available to the system).

1.5 Dissertation structure

This work is organized as follows. Chapter 2 introduces the state–of–the–art RMS when considering the problem assumptions developed earlier. We describe in detail how each module of the simplified RMS works, i.e., how booking data are stored, how the parameters of a demand model are estimated from the historical data, and how the calibrated demand model is used for price optimization.

Following, Chapter 3 presents the basic concepts behind reinforcement learning (RL), which is a field in AI concerned with learning through interactions with an environment. Then, we expose breakthroughs and some popular methods, such as Q-Learning and Deep Q-Networks. We also introduce a brief history of how the field of RM has used RL and some potential pitfalls.

Chapter 4 presents the problem of earning while learning and how researchers addressed this problem in the past. We show how to adapt one of the most promising heuristic methods to airline RM constraints and how to modify the problem formulation to use RL to solve this problem without human supervision. Then, we compare the experimental results between human–designed heuristic methods and the solution obtained through RL. The contents of this chapter represent the major contributions of our work, and it was first presented in [19].

[19]: Gatti Pinheiro, Defoin-Platel, et al. (2022), "Outsmarting human design in airline revenue management"

Chapter 5 discusses adaptations of the proposed framework to address several other open issues in airline RM, such as competition, cannibalization, and non–stationarity. The contents of this chapter are also a contribution, and they are original to this dissertation.

Finally, in Chapter 6, we close our work with a summary of our findings, some last thoughts, and a conclusion about the potential consequences for the practice and science of RM.

1.6 Summary

Businesses have many decisions to make, and such decisions are either related to supply–chain management, where the goal is to minimize costs, or demand management, where the goal is to maximize revenue [1]. The research field of revenue management (RM) addresses theoretical and practical questions of demand–management decisions, such as, but not limited to, pricing decisions. In RM, the goal is to maximize revenue, which can be achieved by controlling supply and demand through prices. If the prices are set too high, the airline can lose customers, and if excessively low, it may lose potential profits. The system's objective is to find the right balance.

[1]: Talluri and Van Ryzin (2004), *The theory and practice of revenue management*

There are many open problems in airline RM, such as pricing under competition, cannibalization, earning while learning, model–free optimization, and non–stationarity, to cite a few. These problems are very challenging, many of them have been investigated for years by researchers, but no definitive solutions have been found as far as we know. We do not expect to address all these problems in this work, but in Chapter 4, we show how artificial intelligence (AI) can tackle the earning–while–learning problem. Chapter 5 discusses how this same technique could address the other issues. We close this dissertation in Chapter 6, where we discuss the consequences of AI on the practice and research of RM.

The revenue management systems (RMSs) have several building blocks, but only some of these blocks are required for the studies we develop. For this reason, we assume a simplified structure of RMS, which consists of the historical database, forecasting, and optimization modules. The historical database stores the data from offers and bookings for every flight, which the forecasting module uses to predict how future demand will answer to prices. Finally, the optimization module computes the pricing policy

that adjusts prices according to the flight's current capacity constraints. Then, customers interact with such prices by choosing whether to book or not.

Revenue Management Systems 2

This chapter reviews the basic forecasting and pricing optimization techniques employed in revenue management systems. We present the RM problem in its simplest form, ignoring complexities, such as seasonality and network optimization, that are unnecessary for studying the earning–while–learning and other issues we seek to address in this work.

2.1 The single–leg problem

One important property of airline revenue management is that airlines manage several flights at the same time. Even if we consider that the airline operates only one leg from point A to point B, the flights of this leg depart with a certain regularity (e.g., once a day or once a week). We consider that a new flight is open for sale every day, and another flight departs closing for new bookings. Each flight is open for sale T days before departure (also known as the booking horizon), and the airline must manage T *active* flights *simultaneously*. For the sake of generality, from now on, we call "a day" "a time step" (because, in principle, it could be any unit of time).

We assume that the customers are only willing to book the flight departing at a specific time step and either choose to book or not to book at all, meaning that customers are not willing to anticipate or delay their travel according to prices. The RMS's goal is to select the prices *for every* active flight to maximize the airline's long–term revenue. The system can only change the price decisions for active flights at each time step. For simplicity, we assume that overbooking and cancellations are not possible and that the flight has a single cabin class (e.g., first–class, business, or economic), being C the flight's capacity for this unique cabin.

Arguably one of the simplest demand models[1], known as the exponential model [21], considers that, for each time step, customers willing to travel arrive (e.g., they query the airline's website) according to a Poisson distribution with mean ν, named the arrival rate parameter, and each arriving customer purchases according to an exponential decay probability for the selected fare f defined as

$$z(f;\phi) = \Pr\{\text{purchase} \mid f, \phi\} \doteq e^{-\phi(f/f_0-1)}, \qquad (2.1)$$

1: In literature, the linear demand model is often used as well [20].

[21]: Gallego and Van Ryzin (1994), "Optimal dynamic pricing of inventories with stochastic demand over finite horizons"

where ϕ is the price sensitivity parameter and the constant f_0 is the lowest fare such that the customers' purchase probability is one ($\Pr\{\text{purchase} \mid f_0\} = 1$).

As mentioned earlier, the simple demand model presented here does not assume any dependency on time, in particular, no seasonality and nor on departure's day–of–week (e.g., Sunday, Monday, etc.). In practice, the customer arrival rate is often chosen to be a function of time $\nu(t)$ to accommodate such customer preferences, and the customer price sensitivity is also chosen to be a function of time $\phi(t)$ to allow the system to model the increasing customers' willingness–to–pay as the time approaches the departure date (e.g., last–minute buyers). Even though these complexities can be ignored without loss of generality for most problems considered in this work, we illustrate, in Chapter 5, possible ways to adapt the proposed algorithms to more complex demand behaviors.

The arrival rate and price sensitivity parameters drive the customer behavior at the leg level, i.e., customers for all active flights of the single–leg share the same behavior. This is aligned with reality because RMSs usually estimate an independent set of demand model parameters for every origin and destination operated by the airline (seasonality, day–of–week, and other customer preferences are represented by a subset of parameters in the demand model). As we assume that the demand model is the same for every active flight in the single–leg problem, one could wonder why the modeling of parallel flights is necessary. When computing forecasts, real–world systems aggregate the data of many flights with the same origin and destination, and we seek to reproduce this behavior. The importance of such a choice will be evident when presenting the earning–while–learning problem in Chapter 4.

2.2 Forecasting

When optimizing the pricing strategy, RMS first needs to *predict* demand for different prices. The forecasting module is responsible for such predictions and is crucial for the airline's profitability. In the airline industry, researchers calculated that a bias of ±20% in the estimation of the demand price sensitivity (i.e., how the expected demand increases or decreases according to the offered price) can reduce revenue by up to 4% [22]. In fact, much of the RM research is dedicated to improving forecasting accuracy [23].

[22]: Fiig, Weatherford, et al. (2019), "Can demand forecast accuracy be linked to airline revenue?"

[23]: Weatherford (2016), "The history of forecasting models in revenue management"

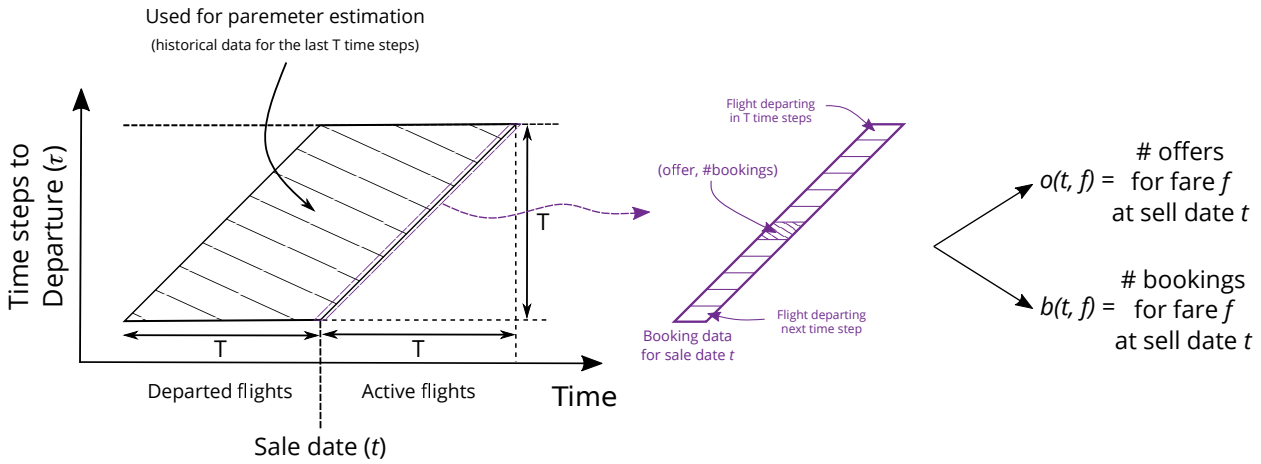


Figure 2.1: The historical database scheme. The historical database keeps records for the last T time steps. At the end of every time step, the booking data collected for that step are immediately added to the historical data and the oldest record is erased. The booking data of each sale date can be further separated by individual entries for each flight, where each entry consists of a pair, in which the first entry is the offer (i.e., the selected fare) and the second entry is the number of observed bookings for that offer. This data can be queried with the help of $o(t, f)$ and $b(t, f)$ functions.

The forecasting module can be represented by a parametric function $d(f; \boldsymbol{\psi})$ that returns the expected number of customers willing to pay for fare f, where $\boldsymbol{\psi}$ represents the parameters of the demand model. Considering that customers follow the exponential model described in the previous section, we can write

$$\begin{aligned} d\left(f; \boldsymbol{\psi} = (v, \phi)\right) &= \mathbb{E}[\#\text{ of purchases} \mid f, \boldsymbol{\psi}] \\ &= v \cdot e^{-\phi\left(\frac{f}{f_0} - 1\right)}, \end{aligned} \tag{2.2}$$

As the true demand behavior, which we denote by $\boldsymbol{\psi}_* = (v_*, \phi_*)$, is unknown by RMS, the system needs to estimate these demand model parameters from historical booking data. We represent the historical booking data with the help of two functions. The first one, $o(t, f) \in \mathbb{N}_0$, returns the number of times fare f is offered across all flights at the sale date t. Similarly, the second one, $b(t, f) \in \mathbb{N}_0$, returns the number of bookings made for fare f at the sale date t across all flights. Note that the number of customer arrivals is not directly observable (it is not possible to distinguish customers not willing to purchase at price f from no customer arrival at all)*. Every time step, the system estimates the parameters of the demand model from historical offers and bookings and uses this calibrated model to optimize the pricing policy. Given the simplicity of our demand model, we assume that the historical database contains the booking data collected

* In real–world systems, these two functions also have a dependency over the time to departure τ, i.e., $o(t, \tau, f)$ and $b(t, \tau, f)$, that we ignore because we removed seasonality and other temporal effects on the demand behavior.

for the last T sale dates, i.e., when new data are appended, the oldest flight data are removed (first–in, first–out). The historical database layout is illustrated in Figure 2.1.

To estimate model parameters, first, we write the log–likelihood function as [24]

[24]: Newman, Ferguson, et al. (2014), "Estimation of choice-based models using sales data from a single firm"

$$\mathcal{LL}(\nu, \phi) = \sum_{f} \sum_{j=t-\mathrm{T}}^{t-1} \Big[b(j,f) \, \ln\big(d(f;\nu,\phi)\big) - o(j,f)\, d(f;\nu,\phi) \Big], \tag{2.3}$$

and then search for the model parameters maximizing[2] $\mathcal{LL}(\nu, \phi)$. Assuming that the demand behaves independently of time, the log–likelihood function above can be re-written in a very computationally and memory-efficient vectorized representation (demonstrated in Example 2.2.1) that can be implemented with libraries specialized in vectorial operations such as NumPy [25].

2: We find that limited–memory BFGS is quite effective, which is an algorithm in the family of quasi-Newton methods.

[25]: Oliphant (2006), *A guide to NumPy*

Example 2.2.1

A common assumption in RMSs is that the system can choose price f from a finite set $\{f_0, f_1, \ldots, f_{n-1}\}$ containing n fares (or price points) [17, 21]. This set is known as the *fare structure*. Given that the assumed model $d(f; v, \phi)$ is constant in time (i.e., no seasonality), the historical database only needs to keep track of the number of bookings and offers made for each fare during the last T sale dates without the precision of when these bookings and offers were made. In other words, we can represent the historical data at time t by two vectors, $\mathbf{o_t} = [o_0, o_1, \ldots, o_{n-1}]$ and $\mathbf{b_t} = [b_0, b_1, \ldots, b_{n-1}]$, where o_i and b_i represent the number of offers and bookings made for fare f_i in the last T sale dates, respectively, i.e., $o_i = \sum_{j=t-\mathrm{T}}^{t-1} o(j, f_i)$ and $b_i = \sum_{j=t-\mathrm{T}}^{t-1} b(j, f_i)$. How can we reformulate eq. (2.3) for these two vectors $\mathbf{o_t}$ and $\mathbf{b_t}$?

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"
[21]: Gallego and Van Ryzin (1994), "Optimal dynamic pricing of inventories with stochastic demand over finite horizons"

Answer. The first step to answering this question is to realize that the log–likelihood function in eq. (2.3) can be written as

$$\begin{aligned}
\mathcal{LL}(v, \phi) &= \sum_{i=0}^{n-1} \sum_{j=t-\mathrm{T}}^{t-1} \left[b(j, f_i) \ln\left(d(f_i)\right) - o(j, f_i)\, d(f_i) \right] \\
&= \sum_{i=0}^{n-1} \left[\ln\left(d(f_i)\right) \sum_{j=t-\mathrm{T}}^{t-1} b(j, f_i) - d(f_i) \sum_{j=t-\mathrm{T}}^{t-1} o(j, f_i) \right] \\
&= \sum_{i=0}^{n-1} \left[\ln\left(d(f_i)\right) b_i - d(f_i)\, o_i \right],
\end{aligned}$$

where we dropped, for the sake of readability, the explicit parametrization of the demand model $d(f; v, \phi)$.

We can also write the evaluation demand model as a vector $\mathbf{d_\psi} = [d_0, d_1, \ldots, d_{n-1}]$, where each component d_i represents the evaluation of the demand model at fare f_i, i.e., $d_i = d(f_i; \boldsymbol{\psi})$. Now we can write the log–likelihood function in its vectorized form as

$$\mathcal{LL}(v, \phi) = \ln(\mathbf{d_\psi}) \cdot \mathbf{b_t} - \mathbf{d_\psi} \cdot \mathbf{o_t}. \tag{2.4}$$

Such an implementation only requires keeping a count of the number of offers and bookings by fare, using just $O(2n)$ memory. It is also computationally efficient because the number of multiplication/subtract operations is kept at a minimum.

Example 2.2.2

Suppose a leg has one flight a day, and each flight is open for bookings one year before its departure (T = 365). If the system uses one year of booking data for parameter estimation, in total, there are 365 flights x 365 offers per flight = 133225 offers in the historical database. Let's assume that RMS selected 20% of the time the lowest fare f_0 = \$50 (26645 offers), 60% of the time fare f_1 = \$150, and, for the remaining time, fare f_2 = \$200. At time t, the system's historical database contains $\mathbf{o}_t = [26645, 79935, 26645]$ and $\mathbf{b}_t = [3650, 3298, 603]$. Using the log–likelihood estimation defined in eq. (2.4), we find that the parameters that better explain the observed data are $\nu = 0.137$ and $\phi = 0.6$. How can we quickly verify that such an estimation is correct?

Answer. At the lowest fare f_0, we observe that the historical database has a total of $o_0 = 26645$ offers and $b_0 = 3650$ bookings. Given that, at f_0, the purchase probability is one, the ratio between the number of bookings by the number of offers gives an estimation of the arrival rate $\nu \approx b_0/o_0 \approx 0.137$. Analogously, we can verify that the model predictions $d(f;\boldsymbol{\psi})$ for the other fares match the observed data.

$$d(\$150;\nu,\phi) = 0.041264 \approx \frac{b_1}{o_1} = 0.041258$$
$$d(\$200;\nu,\phi) = 0.022646 \approx \frac{b_2}{o_2} = 0.022631$$

Figure 2.2 illustrates the historical database points and the fitted demand model for this example.

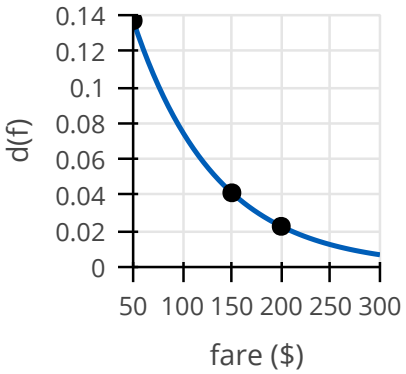


Figure 2.2: The dots represent the historical booking data, here obtained by computing b_i/o_i. The line illustrates a typical fit of the demand behavior function $d(f;\boldsymbol{\psi})$ after maximizing the log–likelihood function eq. (2.3).

2.3 Optimization

The optimization module is responsible for computing the pricing strategy that maximizes revenue. The greatest challenge behind such a task lies in the stochastic nature of the demand behavior: The number of customer arrivals at each time step is uncertain, and each arriving customer may decide to book or deny an offer. Furthermore, because the number of seats available in a flight is limited and adding new seats to a cabin is not practical, the pricing strategy must consider the flight's capacity constraints. Also, the remaining time to departure is a constraint that needs to be accounted for because the time to sell any remaining capacity is limited (flights eventually depart). Therefore, the pricing optimization methods must be able to compute prices according to demand forecasts while addressing both constraints. For example, the RMS may decrease prices to capture more customers when the flight has capacity in excess, or, conversely, it may increase prices to capture only customers willing to book at higher prices when the flight has a deficit of capacity.

To develop some intuition on price optimization, let's first consider the case where the capacity is *unconstrained*, i.e., the flight's total capacity is far greater than the expected number of arriving customers ($C \gg T \cdot v_*$). Considering that the demand behaves according to the exponential demand model described in Section 2.1 and that the fare structure is a finite set, the expected revenue $R(f)$ for fare f can be computed by the product of the fare and the expected demand $d(f; \boldsymbol{\psi})$, thus

$$\begin{aligned} R(f; \boldsymbol{\psi}) &= \mathbb{E}[\text{revenue} \mid f, \boldsymbol{\psi}] \\ &= f \cdot d(f; \boldsymbol{\psi}). \end{aligned}$$

The fare maximizing the expected revenue $R(f)$, named the optimal fare, can be trivially obtained with

$$f_* = \arg\max_f R(f; \boldsymbol{\psi}),$$

where $\arg\max_x f(x)$ denotes the value of x such that $f(x)$ assumes its maximal value. In other words, the system can test the value of $R(f_i)$ for each fare f_i and then select the fare f_i such that $R(f_i)$ assumes its maximal value (see the following Example 2.3.1 for the case of a continuous fare structure). Even though

appealing by its simplicity, such a method is inappropriate when the flight's capacity is constrained because it only optimizes for the expected demand behavior without adapting the prices to a potential deficit or excess of demand.

This section describes how airlines typically compute their pricing strategy according to time and capacity constraints. But before that, we use this concept of the optimal fare under unconstrained capacity to derive a more intuitive way of specifying and interpreting the customer price sensitivity ϕ that we use throughout this work.

Example 2.3.1

When the fare structure is a finite set, we can compute the fare that maximizes the revenue by testing every price point and selecting only the one that yields the most revenue. However, when the system can choose the price f from a continuous fare structure, i.e., prices can be chosen within an interval $f \in [f_0, f_{\max}]$, this "brute force" strategy is no longer possible because there is an infinite number of values in between two real numbers. Instead, we can use the property that the expected revenue function $R(f)$ presents a unique maximum value in the interval $[f_0, f_{\max}]$ and compute the optimal fare directly. How can the optimal fare be obtained when the fare structure is continuous?

Answer. As $R(f)$ assumes its maximum value when its derivative is zero, the optimal price can be obtained with

$$\begin{aligned} \frac{d}{df_*} R(f_*) &= 0 \\ \frac{d}{df_*} [f_* \cdot d(f_*; v, \phi)] &= 0 \\ d(f_*; v, \phi) + f_* \cdot d(f_*; v, \phi) \cdot \frac{-\phi}{f_0} &= 0 \\ f_* &= \frac{f_0}{\phi} \end{aligned} \tag{2.5}$$

Figure 2.3 illustrates the revenue curve when $v_* = 0.137$, $\phi_* = 0.6$, and $f_0 = \$50$. With such settings, the optimal fare is given by $f_* = \$50/0.6 \approx \$83$.

Note that, when $\phi \geq 1$, the optimal fare is simply the lowest ($f_* = f_0$). When the optimal fare is $f_* = f_0$ (or, $f_* = f_{max}$), we can reconfigure the fare structure (i.e., change the pricing range) to give the system more flexibility on prices. In practice, airlines manually calibrate the fare structure to match the real market conditions, and, in a well–calibrated fare structure, the revenue–maximizing price lies somewhere in the interval $]f_0, f_{\max}[$.

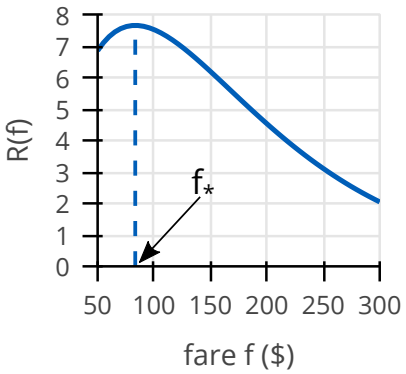


Figure 2.3: Following the example from Figure 2.2, one can compute the revenue function $R(f)$ and the fare maximizes revenue $f_* = \$83$, represented by the vertical dashed line.

2.3.1 Fare ratio at 50% of demand

The price sensitivity can also be referred to in terms of the fare ratio to the lowest fare f'/f_0 at which the purchase probability is 50% [26], denoted by F_5, which is defined as

[26]: Belobaba and Hopperstad (2004), "Algorithms for revenue management in unrestricted fare markets"

$$\begin{aligned} \Pr\{\text{purchase} \mid f'\} &\doteq \frac{1}{2} \\ e^{-\phi\left(\frac{f'}{f_0}-1\right)} &= \frac{1}{2} \\ F_5 \doteq \frac{f'}{f_0} &= \frac{\ln(2)}{\phi} + 1. \end{aligned} \tag{2.6}$$

To understand why specifying the F_5 parameter rather than the price sensitivity is sometimes more convenient, consider computing the revenue–maximizing fare under unconstrained capacity in terms of the F_5. Under such a condition, we can write

$$\begin{aligned} f_* &= \frac{f_0}{\phi} && \text{from eq. (2.5)} \\ &= f_0 \frac{F_5 - 1}{\ln(2)} && \text{use eq. (2.6)} \\ &= \frac{f_0}{\ln(2)} F_5 - \frac{1}{\ln(2)}. \end{aligned}$$

Therefore, the revenue–maximizing fare has an affine relationship (i.e., $y(x) = ax+b$) to the F_5 parameter under the assumption of the exponential demand model and unconstrained capacity, greatly simplifying interpretation. In this work, we use the price sensitivity in equations and for introducing definitions, while we save the F_5 parameter for everything related to experimental settings.

2.3.2 Markov decision process

To address the stochastic behavior of the demand while considering time and capacity constraints, the optimization problem is often formulated as a Markov decision process (MDP) [1, 21], defined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, r, p \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is the reward function, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the state–transition probability function.

[1]: Talluri and Van Ryzin (2004), *The theory and practice of revenue management*
[21]: Gallego and Van Ryzin (1994), "Optimal dynamic pricing of inventories with stochastic demand over finite horizons"

For each active flight, the system faces a sequence of pricing decisions over T time steps. At each time step t (e.g., day), the RMS

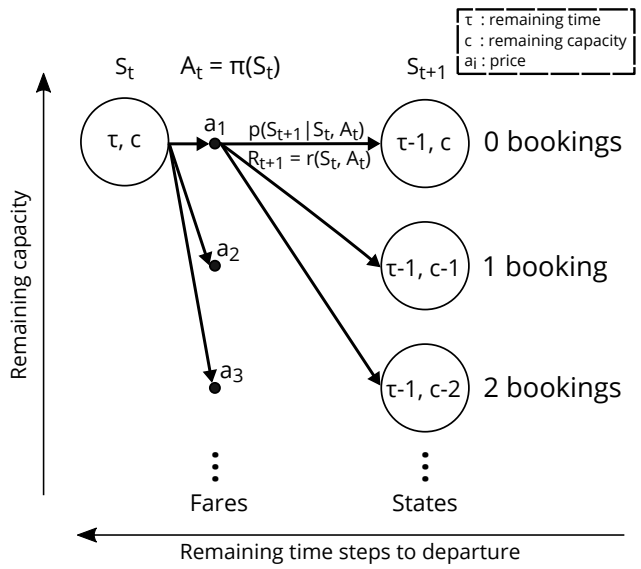


Figure 2.4: The backup diagram for our MDP formulation. The system observes state S_t and it takes an action A_t from a finite set of options $\{a_1, a_2, a_3, \dots\}$. Then, a transition happens according to the state–transition probabilities defined by $p(s' \mid s, a)$. Finally, the system observes a new state S_{t+1} and receives the corresponding reward R_{t+1} defined by the expected reward function $r(s, a)$. Image adapted from [19] under license CC BY 4.0.

observes the inventory state of an active flight $S_t \in \mathcal{S}$, denoted by the tuple $S_t = (\tau, c)$, where $\tau = \mathrm{T} - t$ is the number of time steps to departure, and c is the flight's remaining capacity. Then, it takes an action (i.e., it selects a fare) $A_t \in \mathcal{A} = \{f_0, f_1, \dots, f_{n-1}\}$ according to its pricing policy $\pi : \mathcal{S} \to \mathcal{A}$. Multiple customers can arrive and interact with the RMS by choosing whether to book or not at the selected action $A_t = f$. At time step $t + 1$, with probability $p(S_{t+1}|S_t, A_t; \boldsymbol{\psi})$, the remaining capacity c decreases by the number of bookings made at the previous time step. Finally, the RMS receives a reward equal to the immediately expected revenue[3] $R_{t+1} = r(S_t, A_t)$.

3: We follow the convention of [27], i.e., we use R_{t+1} rather than R_t.

Interactions with customers continue until the flight departs and no further actions are possible, or when the flight's remaining capacity is exhausted and no more bookings are possible. When at least one of these two termination conditions is met, we say that the system reached a *terminal state*, denoted by S_T, marking the end of an *episode*. On the contrary, the first state, for which the flight has all the capacity and time to departure is called the *initial state*, denoted by $S_0 = (\mathrm{T}, C)$. The MDP backup diagram for a single transition is illustrated in Figure 2.4.

The interaction between the system and customers gives rise to a sequence or a *trajectory* in the state–action space

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T, S_T.$$

As RMS seeks to optimize long–term revenue, we can write the objective as a function of the *return* (or total revenue) defined as

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k.$$

The system seeks a policy function π which actions $A_t = \pi(S_t)$ maximize, for every time step t, the expected return $\mathbb{E}_\pi[G_t \mid S_t]$, where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable (in this case, the discounted sum of rewards or the total revenue for a sample episode) while *following policy* π. By following the policy π, we mean that the system selects the actions $A_t, A_{t+1}, \ldots, A_{T-1}$ according to the decisions of the policy function π. The parameter $\gamma \in [0, 1]$ is named the *discount rate*, and it defines how farsighted the system is concerning revenue maximization: a reward (or immediate revenue) received k time steps in the future is worth the fraction γ^{k-1} less as if received immediately. As the discount rate approaches one, i.e. $\gamma \to 1$, the policy function cares about future rewards more strongly. In the case of airline RM, the discount rate is often chosen to be $\gamma = 1$ [17, 21] because the MDP has no loops and the episode's horizon T is finite, guaranteeing that the return is a finite quantity (the choice of $\gamma = 1$ is usually implicit in RM literature).

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"
[21]: Gallego and Van Ryzin (1994), "Optimal dynamic pricing of inventories with stochastic demand over finite horizons"

The final step consists in computing the policy function $\pi(s)$ that maximizes the expected return (i.e., the expected long–term revenue). But to do so, we need first to compute the "mechanics" of the MDP for the exponential demand model, described by the state–transition probability function $p(s' \mid s, a)$ and the expected immediate reward function $r(s, a)$. This is the central subject of the following sections.

The state–transition probability

The state–transition probability function $p(s' \mid s, a)$ returns the probability of observing the next state s' from state s taking action a,

$$p(s' \mid s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}. \tag{2.7}$$

As this function defines a discrete probability distribution, by definition the sum of all possible outcomes must sum to one, i.e.,

$$\sum_{s'} p(s' \mid s, a) = 1. \tag{2.8}$$

To compute this function according to the exponential demand model defined in eq. (2.2), the typical approach consists in dividing the time step into smaller units of time, often referred to as *micro–times*, at which we assume that at most one customer can arrive [21]. However, such an assumption is only a product of mathematical convenience and has no support from a real–world use case. As demonstrated below, this probability function can be computed without introducing the micro–time assumption.

[21]: Gallego and Van Ryzin (1994), "Optimal dynamic pricing of inventories with stochastic demand over finite horizons"

The probability of observing x arrivals at any time step follows a Poisson distribution, which is defined as

$$u(x) \doteq \Pr\{X = x \mid \nu\} = \frac{\nu^x e^{-\nu}}{x!}, \tag{2.9}$$

and each arriving customer purchases according to the exponential probability $z(f;\phi)$ defined in eq. (2.1).

As the selected fare is not allowed to change due to the discrete nature of time in the MDP, for notation simplicity, we denote the purchase probability simply by a constant value of $z = z(f;\phi)$. We can write the probability of observing k bookings as

$$\Pr\{0 \text{ bookings} \mid f\} = \underbrace{u(0)}_{\text{no arrivals}} + \underbrace{u(0)\,(1-z)}_{\text{1 arrival, no purchases}} + \underbrace{u(2)\,(1-z)^2}_{\text{2 arrivals, no purchases}} + \ldots,$$

$$\Pr\{1 \text{ booking} \mid f\} = \underbrace{u(1)\,z}_{\text{1 arrival, 1 purchase}} + \underbrace{\binom{2}{1} u(2)\,z(1-z)}_{\text{2 arrivals, 1 purchase}} + \underbrace{\binom{3}{1} u(3)\,z\,(1-z)^2}_{\text{3 arrivals, 1 purchase}} + \ldots,$$

$$\Pr\{2 \text{ bookings} \mid f\} = u(2)\,z^2 + \binom{3}{2} u(3)\,z^2\,(1-z) + \binom{4}{2} u(4)\,z^2\,(1-z)^2 + \ldots,$$

which can be written compactly as,

$$
\begin{aligned}
\Pr\{k \text{ bookings} \mid f\} &= \sum_{i=k}^{+\infty} \binom{i}{k} u(i)\, z^k\, (1-z)^{i-k} \\
&= \sum_{i=k}^{+\infty} \frac{i!}{(i-k)!\, k!}\, u(i)\, z^k\, (1-z)^k && \text{use } \binom{i}{k} = \frac{i!}{(i-k)!\, k!} \\
&= \sum_{i=k}^{+\infty} \frac{i!}{(i-k)!\, k!} \frac{\nu^i e^{-\nu}}{i!} z^k\, (1-z)^k && \text{apply 2.9} \\
&= \frac{z^k e^{-\nu}}{k!} \sum_{i=k}^{+\infty} \frac{\nu^i}{(i-k)!} (1-z)^{i-k} && \text{rearrange terms} \\
&= \frac{d^k e^{-\nu}}{k!} \sum_{j=0}^{+\infty} \frac{\nu^{j+k}}{j!} (1-z)^j && \text{replace } j = i-k \\
&= \frac{z^k e^{-\nu} \nu^k}{k!} \sum_{j=0}^{+\infty} \frac{\big(\nu(1-z)\big)^j}{j!} && \text{rearrange terms} \\
&= \frac{z^k e^{-\nu} \nu^k}{k!} e^{\nu(1-z)} && \text{use } e^x = \sum_{j=0}^{+\infty} \frac{x^j}{j!} \\
&= \frac{(\nu z)^k}{k!} e^{-\nu z} && \text{rearrange terms} \\
&= \frac{d(f)^k}{k!} e^{-d(f)}. && \text{use } d(f) = \nu z(f) \qquad (2.10)
\end{aligned}
$$

Then, in the general case, where the capacity of the plane has not been exhausted yet, and the remaining capacity is greater than the number of bookings considered, we can write the state–transition probability as

$$p(s' = (\tau - 1, c - k) \mid s = (\tau, c), a = f) = \Pr\{k \text{ bookings} \mid f\} \text{ if } c > k.$$

Finally, the case for which all units of remaining capacity are sold in a single time step ($c = k$) can be written as[4]

4: The customers are not aware of the airline's capacity constraints, thus any arrivals after the exhaustion of capacity are automatically rejected, no matter if customers were willing to pay or not.

$$
\begin{aligned}
p\big(s' = (\tau - 1, 0) \mid s = (\tau, c), a = f\big) &= \sum_{i=c}^{+\infty} \Pr\{i \text{ bookings} \mid f\} \\
&= 1 - \sum_{i=0}^{c-1} \Pr\{i \text{ bookings} \mid f\}. \qquad \text{use eq. (2.8)} \qquad (2.11)
\end{aligned}
$$

Even though tempting, the computation of eq. (2.11) can lead to arbitrarily large errors because each term of the sum carries an estimation error[5] due to floating–point representation [28]. Such errors may add up, making the total error of the sum greater than each term individually. Since the sum can have many terms, the total error may be significant, stacking and propagating when performing optimization. In fact, in our experience, these errors create a measurable inconsistency in results to theoretical estimations.

5: Because computers are limited to binary representation, they need to represent real numbers discretely. Thus, from a computational perspective, every floating–point carries an error, which is often small enough for most applications.

[28]: (2019), *IEEE Standard for floating-point arithmetic*

The workaround is to rewrite eq. (2.11) in terms of the Gamma function $\Gamma(x) \doteq \int_0^\infty t^{x-1}e^{-t}dt$ and the incomplete Gamma function $\Gamma(x,z) \doteq \int_z^\infty t^{x-1}e^{-t}dt$ as

$$
\begin{aligned}
p\big(s' = (\tau-1, 0) \,|\, s = (\tau, c), a = f\big) &= 1 - \sum_{i=0}^{c-1} \Pr\{i \text{ bookings} \,|\, f\} && \text{from eq. (2.11)} \\
&= 1 - \sum_{i=0}^{c-1} \frac{d(f)^i}{i!} e^{-d(f)} && \text{use eq. (2.10)} \\
&= 1 - \frac{\Gamma(c, d(f))}{\Gamma(c)} && \text{use } \frac{\Gamma(x,z)}{\Gamma(x)} = e^{-z} \sum_{i=0}^{x-1} \frac{z^i}{i!}
\end{aligned} \tag{2.12}
$$

Most standard scientific libraries such as SciPy [29] provide efficient and precise implementations for computing the Gamma functions in eq. (2.12).

[29]: Virtanen, Gommers, et al. (2020), "SciPy 1.0: fundamental algorithms for scientific computing in Python"

Example 2.3.2

Consider a flight with a remaining capacity $c = 2$ at an arbitrary remaining time τ, thus, with an inventory state $S_t = (\tau, 2)$. The lowest fare is $f_0 = \$50$, and the system's policy selects fare $A_t = \pi(S_t) = \$100$ for this state. There are three possible outcomes for the next time step. First, there may be no bookings, resulting in state $s' = (\tau - 1, c = 2)$. Or, there may be a single booking $s'' = (\tau - 1, c = 1)$. Finally, RMS may observe two bookings $s''' = (\tau - 1, c = 0)$. If the demand behavior follows the exponential model in eq. (2.2) with $\nu_* = 1.5$ and $F_5^* = 2$, which are the state–transition probabilities for each one of the possible outcomes?

Answer. First, for convenience, we can compute the Gamma function and the demand model with

$$\begin{aligned}
\phi &= \frac{\ln(2)}{F_5 - 1} \approx 0.7 \\
d(f = 100) &= 1.5e^{-0.7(100/50-1)} = 0.745 \\
\Gamma(c = 2) &= 1 \\
\Gamma(2, d(100)) &= 0.828.
\end{aligned}$$

Using eq. (2.10), we can write

$$\begin{aligned}
p(s' \mid S_t, A_t) &= \frac{d(100)^0}{0!} e^{d(100)} = 0.475 \\
p(s'' \mid S_t, A_t) &= \frac{d(100)^1}{1!} e^{d(100)} = 0.353.
\end{aligned}$$

Thus, at fare \$100, we expect to observe 0 bookings in the time step with probability 47.5%, and 1 booking with probability 35.3%.

The probability of observing 2 bookings, which corresponds to the exhausting all remaining units of capacity, can be computed with eq. (2.12),

$$\begin{aligned}
p(s''' \mid S_t, A_t) &= 1 - p(s' \mid S_t, A_t) - p(s'' \mid S_t, A_t) \\
&= 1 - \frac{\Gamma(2, d(100))}{\Gamma(2)} = 0.172.
\end{aligned}$$

We expect to observe that the 2 remaining units of capacity are sold in the time step with probability 17.2%.

The reward function

As the system's goal is to maximize total revenue, the reward signal R_t represents the immediate expected revenue given by

$$r(s = (\tau, c), a = f) = f \cdot \sum_{j=1}^{c} j \cdot \Pr\{j \text{ bookings} \mid f\} \tag{2.13}$$

As in the case of the transition probability, the issue with such a formulation is that the sum may lead to arbitrarily large errors when the number of terms (i.e., the capacity) increases. Instead, we can perform the same trick as in the previous section and rewrite it for the Gamma function as

$$\begin{aligned}
r(s,a) &= f \sum_{j=1}^{c} j \cdot \Pr\{j \text{ bookings} \mid f\} && \text{from eq. (2.13)} \\
&= f \left[\sum_{j=1}^{c-1} j \frac{d(f)^j}{j!} e^{-d(f)} + c \left(1 - \frac{\Gamma(c, d(f))}{\Gamma(c)} \right) \right] && \text{use eq. (2.10) and eq. (2.12)} \\
&= f \left[c \left(1 - \frac{\Gamma(c, d(f))}{\Gamma(c)} \right) + d(f) e^{-d(f)} \sum_{j=1}^{c-1} \frac{d(f)^{j-1}}{(j-1)!} \right] && \text{rearrange terms} \\
&= f \left[c \left(1 - \frac{\Gamma(c, d(f))}{\Gamma(c)} \right) + d(f) e^{-d(f)} \sum_{i=0}^{c-2} \frac{d(f)^i}{i!} \right] && \text{replace } i = j - 1 \\
&= f \left[c \left(1 - \frac{\Gamma(c, d(f))}{\Gamma(c)} \right) + d(f) \frac{\Gamma(c-1, d(f))}{\Gamma(c-1)} \right]. && \text{use } \frac{\Gamma(x,z)}{\Gamma(x)} = e^{-z} \sum_{i=0}^{x-1} \frac{z^i}{i!}
\end{aligned} \tag{2.14}$$

In our experience, this formulation yields better precision and CPU performance.

Example 2.3.3

As in the previous example, consider the case where the flight has a remaining capacity $c = 2$ at an arbitrary remaining time τ. The base fare is $f_0 = \$50$, and the system's policy selects fare $A_t = \pi(S_t) = \$100$. The demand behavior follows the exponential model in eq. (2.2) with $v_* = 1.5$ and $F_5^* = 2$. What is the expected immediate revenue $r(S_t, A_t)$?

Answer. First, we can compute the state–transition probabilities and gamma functions with

$$\begin{aligned}
&\phi = \frac{\ln(2)}{F_5 - 1} \approx 0.7 \\
&d(f = 100) = 1.5e^{-0.7(100/50-1)} = 0.745 \\
&\Gamma(1) = 1.0 \\
&\Gamma(2) = 1.0 \\
&\Gamma\left(2, d\left(100\right)\right) = 0.828 \\
&\Pr\{1 \text{ booking} \,|\, \$100\} = 0.353 \\
&\Pr\{2 \text{ bookings} \,|\, \$100\} = 0.172,
\end{aligned}$$

then we can compute the expected immediate revenue with eq. (2.14)

$$\begin{aligned}
r\left(S_t, A_t\right) &= \$100 \cdot \left(\Pr\{1 \text{ bookings} \,|\, \$100\} + 2\Pr\{2 \text{ bookings} \,|\, \$100\}\right) \\
&= \$100 \left[2\left(1 - \frac{\Gamma(2, d(100))}{\Gamma(2)}\right) + d(100)\frac{\Gamma(1, d(100))}{\Gamma(1)}\right] \\
&= \$69.7.
\end{aligned}$$

The expected immediate revenue for this time step is \$69.7.

Example 2.3.4

The function $d(f;\boldsymbol{\psi})$, as defined in eq. (2.2), represents the expected demand for fare f. Therefore, it may be tempting to set $R_t = f \cdot d(f;\boldsymbol{\psi})$. Why is this incorrect?

Answer. The issue with such an approach is that a part of this demand (or all the demand) may be rejected according to the capacity constraints, which such formulation does not take into account. Indeed, the result of eq. (2.14) converges to $f \cdot d(f;\boldsymbol{\psi})$ when the capacity approaches infinity, as demonstrated below.

$$
\begin{aligned}
\lim_{c\to\infty} r(s,a) &= \lim_{c\to\infty} f\left[c\left(1-\frac{\Gamma\left(c,d(f)\right)}{\Gamma(c)}\right)+d(f)\frac{\Gamma\left(c-1,d(f)\right)}{\Gamma(c-1)}\right] && \text{from eq. (2.14)}\\
&= f\left[\lim_{c\to\infty} c\left(1-\frac{\Gamma\left(c,d(f)\right)}{\Gamma(c)}\right)+d(f)\lim_{c\to\infty}\frac{\Gamma\left(c-1,d(f)\right)}{\Gamma(c-1)}\right] && \text{rearrange terms}\\
&= f\left[\lim_{c\to\infty} c\left(1-\frac{\Gamma\left(c,d(f)\right)}{\Gamma(c)}\right)+d(f)\right] && \text{use } \lim_{x\to\infty}\frac{\Gamma(x,z)}{\Gamma(x)}=1\\
&= f\cdot d(f) && \text{use } \lim_{x\to\infty} x\left(1-\frac{\Gamma(x,z)}{\Gamma(x)}\right)=0
\end{aligned}
$$

2.3.3 Dynamic programming

Once the mechanics of the MDP have been defined, our attention turns toward optimizing the pricing policy. As discussed in Section 2.3.2, the system's goal is to maximize the total amount of reward it receives, meaning that RMS does not seek to maximize the expectation of immediate reward but rather the expectation of cumulative reward in the long run. The system searches for a pricing policy π that tells how to vary prices according to the current inventory state to maximize the expected cumulative reward $\mathbb{E}_\pi[G_t|S_t]$.

To evaluate and compare policies, we define a *value function* $v_\pi(s)$ that outputs the expected return when starting in s and following policy π subsequently,

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s\right]. \tag{2.15}$$

In other words, the value function outputs the expected long–term reward that a policy collects. Note that we can compute the value function for any given policy, and different policies yield different value functions. Furthermore, a practical property is that the value of the terminal state is zero $v_\pi(S_T) \doteq 0$ because it

indicates the end of an episode, where no further rewards can be obtained.

Similarly, we can define the value of starting from state s, selecting action a, and following policy π thereafter as

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \,\middle|\, S_t = s, A_t = a \right]. \tag{2.16}$$

This function is known as the *action–value function* for policy π.

A policy π is said to be better than or equal to another policy π', i.e., $\pi \geqslant \pi'$, if and only if the value function of this policy is equal or greater than the value function of another policy for all states, i.e., $v_\pi(s) \geqslant v_{\pi'}(s) \,\forall s$ [27]. The RMS looks for a policy that is better or equal to all others, which we call an *optimal policy*, denoted by π_*. The value function for the optimal policy is defined as

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

$$v_*(s) \doteq \max_\pi v_\pi(s) \;\; \forall s,$$

or if written in terms of the action–value function,

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \;\; \forall s, a.$$

A useful property is that v_* can be written in terms of q_* as

$$\begin{aligned}
v_*(s) &= \max_a q_*(s,a) \\
&= \max_a \mathbb{E}_{\pi_*} \left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s, A_t = a \right] && \text{use eq. (2.16)} \\
&= \max_a \mathbb{E} \left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a \right] && \text{use eq. (2.15)} \\
&= \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a)\, v_*(s') \right].
\end{aligned}$$

This is known as the *Bellman optimality equation* for v_*, popularized by Richard Bellman [30], who named it the "basic functional equation". We can also write the Bellman optimality equation in terms of the action–value function as

[30]: Bellman (1957), *Dynamic Programming*

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right]$$
$$= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \max_{a'} q_*(s', a'). \qquad (2.17)$$

Dynamic programming (DP) refers to the collection of algorithms that can solve the above equation when assuming that the state–transition probability function is completely known [27]. Indeed, if the mechanics of the MDP are completely known, the MDP presents no loops, and the state–space is small enough, we can keep in memory the values for each state–action pair and compute eq. (2.17) backward from the terminal state S_T, where $q(S_T, a) \doteq 0 \; \forall a$. Once the action–value function has been computed, the optimal policy can be obtained with $\pi_*(s) = \arg\max_a q_*(s,a)$.

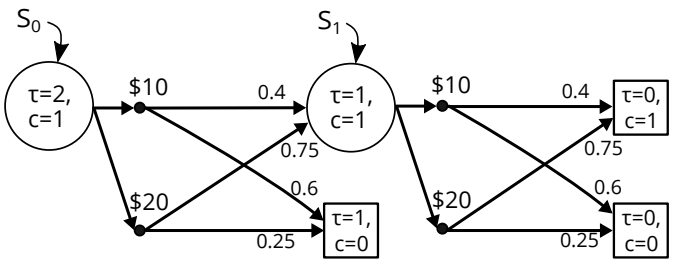


Figure 2.5: The backup diagram for Example 2.3.5. Terminal states are represented with squares.

Example 2.3.5

Consider a flight with only two units of time T = 2 and a single unit of capacity C = 1. At each time step, the system can choose among two possible fares, $f \in \{10, 20\}$. When selecting the base fare $f_0 = 10$, the probability of observing a booking is 0.6, but when selecting the highest fare $f = 20$, the probability of observing a booking decreases to 0.25. The backup diagram for the corresponding MDP is illustrated in Figure 2.5. Assuming the discount rate $\gamma = 1$, which is the optimal policy for each state?

Answer. As the action-value function for the terminal states is zero ($q_*(S_T, a) \doteq 0$), we can compute the Bellman optimality equation in eq. (2.17) "backtracking" from these terminal states. For state $S_1 = (\tau = 1, c = 1)$, we can write

$$q_*(S_1, 10) = 0.4 \cdot 0 + 0.6 \cdot 10 = 6,$$
$$q_*(S_1, 20) = 0.75 \cdot 0 + 0.25 \cdot 20 = 5.$$

Thus, as fare of \$10 corresponds to the action that yields the most revenue in expectation, the optimal policy selects this fare, i.e. $\pi_*(S_1) = \arg\max_{f \in \{10,20\}} q(S_1, f) = \$10$. However, as demonstrated below, the decision changes for the initial state $S_0 = (\tau = 2, c = 1)$.

$$q_*(S_0, 10) = 0.4\,(0 + 1 \cdot 6) + 0.6\,(10 + 1 \cdot 0) = 8.4,$$
$$q_*(S_0, 20) = 0.75\,(0 + 1 \cdot 6) + 0.25\,(20 + 1 \cdot 0) = 9.5.$$

Therefore, the optimal policy selects the highest fare for the initial state $\pi_*(S_0) = \arg\max_{f \in \{10,20\}} q(S_0, f) = \$20$. In summary, to maximize the expected revenue, the system must select the fare of \$20 for the first time step, and if the unit of capacity is not sold, the system must select the fare of \$10 for the second time step.

Example 2.3.6

What is the optimal policy for a flight with horizon T = 365 and capacity C = 50, with a fare structure of 10 price points $f \in \{\$50, \$70, \ldots, \$230\}$, where the lowest fare is $f_0 = \$50$, and the demand follows the exponential model from eq. (2.2) with $\nu_* = 80/365$ arrivals and $F_5^* = 2.75$?

Answer. In Figure 2.6 (left), we represent the policy matrix computed with DP, as described earlier in this section. For each inventory state (remaining capacity and days to departure), the matrix indicates the price point maximizing the long–term revenue. The initial state $S_0 = (365, 50)$ is located in the top–right corner, and, at this state, the optimal price point is $\pi(S_0) = \$130$. If capacity decreases faster than time, the optimal policy increases the prices. In practice, the trajectories in the state–action space follow the inverse diagonal of the policy matrix, i.e., it starts in the top–right corner and moves towards the bottom-left corner.

In practice, many states from state–space rarely are experienced. Thus, when representing the policy, it is natural to weight every action by its probability of being selected. In Figure 2.6 (right), we plot the "rollout" policy that displays the distribution of selected fares obtained by sampling episodes. For example, we can see that price point \$230 is often chosen by the optimal policy matrix represented on the left. However, the states where such a price point occurs are rarely observed. Consequently, the price point \$230 is seldom selected.

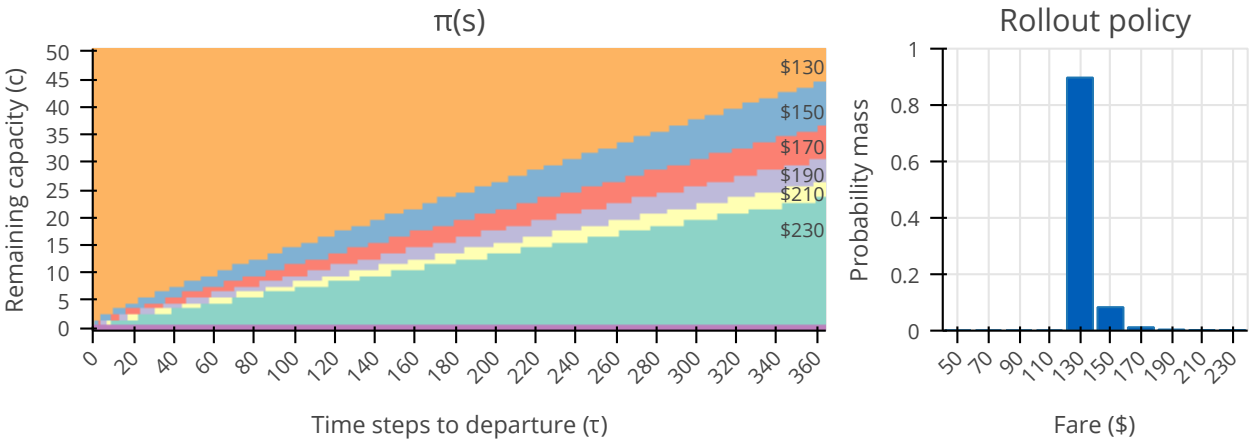


Figure 2.6: The optimal policy is computed with dynamic programming for the exponential demand model with $\nu_* = 80/365$ and $F_5^* = 2.75$.

2.4 Summary

The fundamental design of a revenue management system is composed of a historical database, a forecaster, and an optimizer. The forecaster uses the historical booking data to fit a parametric demand model, which is used for optimizing the pricing policy.

One of the simplest models is the exponential demand model, which assumes that customers arrive according to a Poisson process and decide to purchase or not according to the exponential decay probability as a function of the fare. An approximation of the demand model parameters can be obtained through the maximum log–likelihood estimation.

We can describe the pricing problem as a finite–state Markov decision process (MDP), where states are the number of remaining time steps to departure and the remaining units of capacity, the actions are the fares that the system is allowed select, and the rewards are the immediate expected revenue for each time step. The system's goal is to find the mapping between states and actions (policy) to maximize the reward signal (i.e., long–term revenue).

The collection of algorithms that can solve this optimization problem, called dynamic programming (DP), relies on the concept of value functions to organize the search for optimal policies. DP methods require the perfect knowledge of the MDP mechanics, i.e., the complete distribution of state–transition probabilities.

To implement the DP algorithm, the system needs to translate the exponential demand model to the transition probabilities required by the Bellman optimality equation. Because the MDP presents no loops, the action–value function can be computed by backtracking from terminal states. Finally, the optimized pricing policy can be trivially obtained by selecting the fare maximizing the action–value function for each state.

Pricing Optimization with Reinforcement Learning

3

Applying the techniques of reinforcement learning to RM is not a new concept. This chapter reviews the basic concepts behind reinforcement learning, how researchers applied it to airline RM, and the limits of current approaches.

3.1 The nature of reinforcement learning

Learning from our own experience is probably one of the first ideas that occur when we think about the nature of learning. An infant learning how to walk needs to learn how to translate the sensory information, such as sight and sense of balance, into muscle contraction and relaxation that corresponds to the act of walking. The child learns to walk from experience obtained through trial–and–error with the surrounding environment*. This idea of learning from experience is central to reinforcement learning (RL), which concentrates on the computational approaches for learning through interaction with an environment [27].

The modern field of RL comes from the intertwining of the branch of psychology that focuses on the investigation of animal learning theory and the field of mathematics that focuses on optimal control and its solutions based on value functions and DP [27]. Even though the parallels between the mathematical models and experimental studies from psychology and neuroscience are striking [32–34], we only focus on applying the mathematical methods to address decision problems in the field of airline RM.

The term *reinforcement learning* can sometimes be confusing because it often refers to a set of methods that addresses a particular class of problems and the field that studies this class of problems. By RL, we refer to the field of study that addresses optimal control of *incompletely–known* Markov decision processes, in contrast to classical solution methods that require the *complete* knowledge of the MDP dynamics (i.e., the state–transition probability and expected reward functions), such as DP.

In the center of RL, we find the agent representing the decision–maker. The agent's task is to map *observations* (i.e., states) from

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

[32]: Ludvig, Sutton, et al. (2008), "Stimulus representation and the timing of reward-prediction errors in models of the dopamine system"
[33]: Takahashi, Schoenbaum, et al. (2008), "Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model"
[34]: Tomov, Schulz, et al. (2021), "Multi-task reinforcement learning in humans"

* Children may also learn how to walk by observing adults and other children. Topics of such imitation learning [31] are beyond our scope.

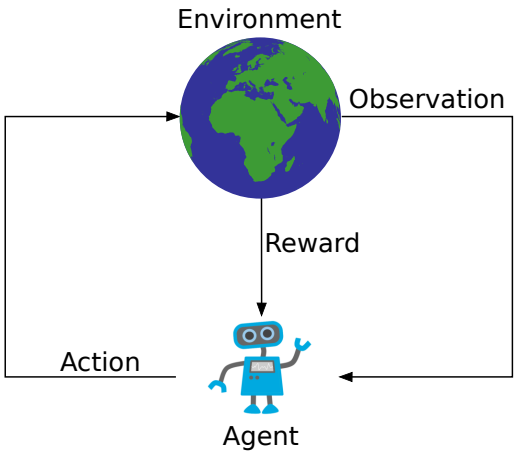


Figure 3.1: The RL agent interacts with its environment. The environment emits observations and rewards. The agent maps the observations into actions, which in turn may influence the environment's future observations. The RL agent seeks to select actions in order to maximize the collected rewards.

the environment into *actions* while maximizing a numerical *reward signal* defining the agent's goal. In principle, the agent is uninformed of which actions to take and needs to discover, through trial–and–error, the actions yielding the most rewards. Usually, the agent starts performing poorly, often as good as choosing actions at random, and, by accumulating experience, the agent's performance gradually increases[1]. In the most interesting cases, selecting an action impacts the future observed states and thus the future rewards. The trial–and–error nature and delayed rewards are the two most distinctive features of RL [27]. In Figure 3.1, we illustrate how the several elements of an RL system interact.

1: The performance metric is most often chosen to be, but not limited to, the sum of the observed discounted rewards (i.e., the return).

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

The definition of observations, actions, and rewards depends on the task. For example, when researchers developed an RL agent capable of defeating Lee Sedol, the 18-time world champion in the game of Go [35] (see Figure 3.2), the observations were a $19 \times 19 \times 48$ image stack in which each point on the 19×19 Go board was represented by 48 binary or integer features. These features described whether the position was occupied or unoccupied by a stone and if the stone was an opponent stone or not, conveying the raw state of the board. Other features were motivated by the game rules, such as the number of adjacent unoccupied points or any features the design team considered important. The action was where to place the next stone, and the reward signal was +1 for victory, −1 for defeat, and zero otherwise.

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

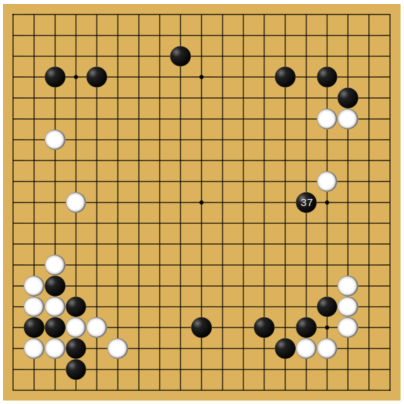


Figure 3.2: Screenshot showing AlphaGo's move B37 described by Michael Redmond, a 9-dan professional player, as "creative" and "the reason people become pros" during the second game of the DeepMind vs. Lee Sedol challenge. Image adapted from en.wikipedia.org under license CC BY-SA 3.0.

Beyond the environment, the agent, and the reward signal, there are three main subelements in the RL system. The first is the *policy*, which is the function that maps observations from the environment into actions. The goal of the RL agent is to find the policy maximizing the reward signal. The policy represents a lookup table, a complex computation process, a set of rules, or a parameterized function, to cite a few. It can be deterministic,

producing a single action to take, or stochastic, returning a probability distribution for all available actions [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

The second subelement is the *value function*, which is responsible for identifying what is good or bad from the perspective of the reward signal in the long run. Whereas rewards define good and bad decisions instantaneously, the value function seeks to predict if a particular state is good or bad regarding future rewards when following a given policy [27]. In the problems we address, the solution often requires short–term sacrifice to achieve long–term goals.

The third and last subelement is the *model* of the environment[2]. The model represents anything the RL agent can use to predict how the environment responds to actions. There are essentially two types of models: The *distribution models*, which describe every possible outcome weighted by their probabilities when transitioning from one state to another, and the *sample models*, which produce just one example from all possible outcomes. RL systems using models and planning are known as *model–based* methods, contrary to pure trial–and–error learners, called *model–free* methods [27].

2: The model is optional and often not present.

3.2 A brief introduction to the mathematical theory of reinforcement learning

This section reviews some mathematical methods for addressing the optimal control of incompletely-known MDPs. We seek to explain and illustrate some of the most fundamental concepts behind the RL methods used throughout this work.

3.2.1 Temporal–difference learning

There are many ways to use the experience to optimize policies, but the most influential methods use the concept of temporal–difference learning (or TD-learning) [36]. To explain this concept, we first concentrate on the *prediction* problem, where the system's policy π is given and fixed, and the goal is to evaluate how good this policy is concerning reward maximization. In other words, we seek to compute an approximation of the value function defined in eq. (2.15), denoted as $V(s)$, from experience generated by interactions between the policy π with the environment, i.e., $V(S_t) \approx v_\pi(S_t)$. The collected experience are sample trajectories in the state–action space $S_0, A_0, R_1, \cdots, S_T$, as described in Section 2.3.2.

[36]: Sutton (1988), "Learning to predict by the methods of temporal differences"

Algorithm 1: Tabular TD(0) for estimating v_π, adapted from [27]

Input: The policy π to be evaluated and the learning rate $\alpha \in [0, 1]$.

initialize $V(s)$, for all $s \in \mathcal{S}$ arbitrarily, except that $V(S_T) \doteq 0$;
Loop forever
    initialize $S \leftarrow S_0$;
    for *each step of episode* **do**
        $A \leftarrow \pi(S)$;
        take action A, observe R, S';
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$;
        $S \leftarrow S'$;

Perhaps the simplest TD-method, known as TD(0), updates the estimate of the value function $V(S_t)$ at time $t + 1$ using the observed reward R_{t+1} and the next state S_{t+1} according to

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - \underbrace{V(S_t)}_{\text{estimate}}}_{\text{TD-error}}\Big], \tag{3.1}$$

where '$\leftarrow$' denotes that the current value is updated with the evaluation of the right–hand side of the equation, and the parameter $\alpha \in [0, 1]$ is the *learning rate*, often referred to as the step–size. The value function $V(\cdot)$ is represented as a table holding separated estimations for every encountered state. The updates are performed as soon as the experience is collected, as described in Algorithm 1.

To capture the intuition behind the TD(0) algorithm, note that the quantity in between the brackets of eq. (3.1) plays the role of an error, also known as the TD-error, which returns the difference between the current estimate $V(S_t)$ and a new better estimate given by $R_{t+1}+\gamma V(S_{t+1})$. This form arises in many RL algorithms, and we often denote the TD-error with

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{3.2}$$

A remarkable aspect of the TD(0) algorithm relates to its similarity to the iterative average $\bar{X}$ of random observations X_i, given by

$$\bar{X}_{i+1} = \bar{X}_i + \frac{1}{i}\Big[\underbrace{\underbrace{X_i}_{\text{target}} - \underbrace{\bar{X}_i}_{\text{estimate}}}_{\text{error}}\Big],$$

where $\bar{X}_0 \doteq 0$. In other words, TD(0) is essentially performing averages, where the target is the next reward added to the current estimation of the value of the observed next state $X_i = R_t + \gamma V(S_{t+1})$, and the old estimate is given by the current estimation of the value function for the observed state $\bar{X}_i = V(S_t)$. The learning rate $\alpha = 1/i$ decreases as the agent experiences new sample observations.

Contrary to the equation of the iterative average, we presented the learning rate α in eq. (3.1) as a fixed value. However, it is a common practice to vary the learning rate as training progresses. Let's denote $\alpha_n(s)$ as the learning rate parameter used to process the reward received after the n-th observation of state s. To guarantee the convergence of the TD(0) algorithm, the learning rate needs to decrease while following the stochastic approximation conditions [27]

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(s) < \infty. \tag{3.3}$$

The simplest way to ensure such a condition is to set $\alpha_n(s) = 1/n$, joining the iterative average equation. If the learning rate is constant $\alpha_n(s) = \alpha$, then the system never converges completely, adapting its responses with more emphasis to the most recently received rewards.

Example 3.2.1

To provide some intuition on the TD(0) algorithm, we present an adaptation of Example 6.4 from [27]. In many cases of interest, the learning system has available only a finite amount of experience for training, e.g., 10 episodes. In such cases, a typical approach is to repeatedly input the available *batch* of data into the learning method until convergence. The TD(0) algorithm converges deterministically to a single answer independently of the learning rate α parameter as long as α is sufficiently small. Consider yourself the predictor of returns for an unknown Markov reward process. Markov reward processes are like Markov decision processes but without actions. Suppose you observe the following eight episodes

(1) B, 1
(2) B, 1
(3) B, 1
(4) B, 1
(5) B, 1
(6) B, 1
(7) B, 0
(8) A, 0, B, 0

The first episode starts in B and immediately terminates with a reward of 1. The six following episodes are similar, except for the last two. In the seventh episode, we observe an episode that starts in B and terminates with a reward of 0. The last episode starts in state A, then it transitions to state B with a reward of 0 and terminates with a reward of 0 as well. Given such a *batch* of observations, what could be reasonable estimates for $V(A)$ and $V(B)$?

Answer. We have observed state B once in all eight episodes, where in six of these episodes, the experienced reward transitioning for the terminal state is 1. Thus,

$$V(B) = \frac{6}{8} = 0.75$$

For state A, we observe it only once, in the eighth episode, where it transitioned to state B with a reward of 0. Because the value of state B is $V(B) = 3/4$, and we did not observe any rewards from state A to state B, it is reasonable to bootstrap the value of state A from the value of state B as $V(A) = V(B) = 3/4$.

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

3.2.2 Q-Learning

In most use cases, we are interested in finding the optimal control policy, which is not possible with model–free methods based only on the value function estimation, such as TD(0). Instead, the system needs to estimate the value of each action explicitly for inferring a policy [27]. Therefore, our goal is to approximate the action–value function $Q(s, a) \approx q_*(s, a)$, which outputs the expected return when starting at state s, taking action a, and following the optimal policy afterward. One of the most influential RL methods for learning this function is named Q-Learning [37], which was one of the first breakthroughs in RL because it enabled early convergence proofs. The Q-Learning update is defined as

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

[37]: Watkins (1989), "Learning from delayed rewards"

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)}_{\text{TD-Error}}. \tag{3.4}$$

Similar to TD(0), Q-Learning computes estimations based on the TD-errors, but differently from TD(0), Q-Learning uses action–value estimates $Q(s, a)$ rather than value estimates $V(s)$. For Q-Learning, correct convergence only requires that all state–action pairs are updated continuously and that the learning rate decreases according to the stochastic approximation conditions presented in eq. (3.3) [27].

To guarantee that all state–action pairs are continuously updated, the RL agent needs to try different actions when experiencing the same state (within different episodes or the same episode if the MDP has loops). However, the RL agent must select the action that yields the most rewards to maximize the expected return.

Actions maximizing the expected return according to the RL agent's current knowledge of its environment are named *greedy* actions, and the act of selecting such actions is known as *exploitation*. On the contrary, selecting nongreedy actions is called *exploration*, and such actions allow the agent to improve its knowledge of how the environment responds to nongreedy actions. Exploitation is the correct thing to do when maximizing the expected return. However, the agent cannot hope to know with certainty the right thing to do because all the agent has available to make decisions are estimations of action values (and the agent needs to try these actions to learn the value of nongreedy actions). As it

Algorithm 2: Q-Learning for estimating $\pi \approx \pi_*$, adapted from [27]

Input: Learning rate α, discount rate $\gamma \in [0, 1]$ and $\epsilon > 0$

initialize $Q(s, a)$ arbitrarily, except that $Q(\text{terminal}, \cdot) = 0$;
Loop forever
    initialize $S \leftarrow S_0$;
    for *each step of the episode* **do**
        choose A from S using the policy derived from Q (e.g., ϵ-greedy);
        take action A, observe R, S';
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$;
        $S \leftarrow S'$;

is impossible to select greedy and nongreedy actions simultaneously, we face a "conflict" between exploration and exploitation, often referred to as the exploration–exploitation trade–off [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

Perhaps the simplest way to enforce that the RL agent explores its environment is by setting some small probability ϵ, such that the agent randomly selects among all possible actions with equal probability, and the rest of the time, the RL agent selects the greedy action. The policies that follow such an exploration method are called ϵ-greedy policies, and they are defined as

$$\pi(s) = \begin{cases} \arg\max_a Q(s, a) & \text{if } \{\mathrm{X} > \epsilon \,|\, \mathrm{X} \sim \mathrm{U}(0, 1)\} \\ \text{random action} & \text{otherwise} \end{cases} \tag{3.5}$$

where $\mathrm{X} \sim \mathrm{U}(0, 1)$ is a random variable sampled according to the uniform distribution in the range $[0, 1]$.

The Q-Learning algorithm presented in Algorithm 2 is similar to TD(0) from Algorithm 1. The agent observes states, interacts with its environment through actions, and receives rewards. The agent updates the value function at each time step according to the error between its expectations and observed rewards. The Q-Learning algorithm differs mainly with respect to the value function update rule and the presence of the ϵ-greedy exploration.

Example 3.2.2

If the environment has $|\mathcal{A}| = 5$ possible actions, and the designer chooses $\epsilon = 0.1$, which is the probability of taking the greedy action?

Answer. The greedy action can be selected as an exploitation action with probability $1 - \epsilon = 0.9$, or as an exploration action with probability $1/|\mathcal{A}| = 0.2$. Thus, the probability of taking the greedy action is given by

$$\begin{aligned}\Pr\left\{A = \arg\max_a Q(S, a)\right\} &= (1 - \epsilon) + \epsilon \frac{1}{|\mathcal{A}|} \\ &= 0.9 + 0.1 \cdot 0.2 \\ &= 0.92\end{aligned}$$

3.2.3 Connections to dynamic programming

Dynamic programming and Q-Learning, represented by eqs. (2.17) and (3.4), seek to compute optimal action–value function through two different means. One of the most important advantages of Q-Learning is that it approximates q_* from sample data. In other words, the system does not need the state–transition probabilities $p(s' \mid s, a)$ and the expected reward $r(s, a)$ functions, as in the previous chapter. This is appealing because it enables learning from direct experience with the world without the need to build any models (model–free). Furthermore, even if learning is performed from experience generated with interactions of a simulated model, "in surprisingly many cases it is easy to generate experience sampled according to the desired probability distributions, but infeasible to obtain the distributions in explicit form" [27], as required by DP. For example, it is much easier to develop a sample model environment for Poker [38] than to compute all possible outcomes weighted by their probabilities.

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

[38]: Heinrich and Silver (2016), "Deep reinforcement learning from self-play in imperfect-information games"

Another issue related to DP is the *curse of dimensionality*. To compute the state–action value function in eq. (2.17), the system must perform at least one single sweep across the state–action space. In many problems of interest, the state–action space can be arbitrarily large, making it impossible to perform even a single sweep in a reasonable amount of time. For example, in StarCraft II, the action space alone is estimated to be the size of 10^{26} [39], making it computationally impractical for exact methods. On the other hand, RL generates trajectories and performs updates at the state–action pairs encountered along the way, directing the learning towards state–action pairs that

[39]: Vinyals, Babuschkin, et al. (2019), "Grandmaster level in StarCraft II using multi-agent reinforcement learning"

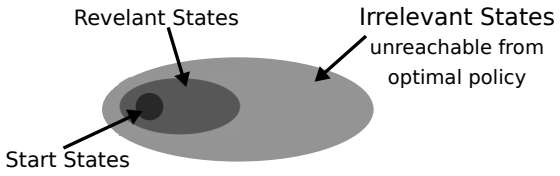


Figure 3.3: From a given set of start states, some states may be visited very rarely or they may not even be reachable while following an optimal policy. For such states, there is no need to specify optimal actions. Figure was adapted from [27] under license CC BY-NC-ND 2.0.

occur more frequently, thus focusing computational resources where they are most needed (see Figure 3.3). We call such a way of generating experience and performing updates *trajectory sampling*[3]. Even though trajectory sampling is simple and efficient, it is not enough when addressing problems with arbitrarily large state–action spaces because tabular RL methods, such as Q-Learning, must keep the memory of the value for all possible state–action pairs, which is impractical for large optimization problems. In the following sections, we discuss ways to address this issue.

3: Adaptations of DP were proposed to incorporate trajectory sampling, such as real–time dynamic programming, which is an on–policy trajectory sampling version of the value–iteration algorithm of DP [27]. This class of algorithms is not considered in our work, because we concentrate on the optimization of unknown MDPs where the state–transition probability function is not available to the system.

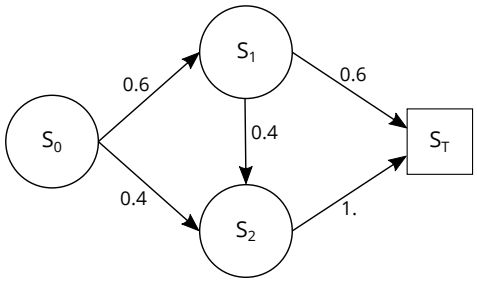


Figure 3.4: The backup diagram for Example 3.2.3.

Example 3.2.3

Consider the three–state Markov chain illustrated in Figure 3.4. Markov chains are like the Markov decision processes, but without actions or rewards, where the system observes only a sequence of states. As usual, the system starts every episode in state S_0, and, in the following time step, it may transition to state S_1 with a probability of 0.6 or transition to state S_2 with a probability of 0.4. Similarly, when in state S_1, the system can transition to the terminal state S_T with a probability of 0.6, or reach state S_2 with a probability of 0.4. From state S_2, the system always transitions to the terminal state. Which is the probability of observing state S_2 during an episode?

Answer. The state S_2 can be reached from two conditions, either from state S_1 with a probability of 0.4 or from state S_0 with a probability of 0.4. In other words, we can write

$$\Pr\{S_2\} = 0.4 \cdot \Pr\{S_1\} + 0.4 \cdot \Pr\{S_0\}$$

The probability of observing the initial state is one because every episode starts in this state, i.e., $\Pr\{S_0\} \doteq 1$. The only way to reach state S_1 is from state S_0, which happens with a probability of 0.6, thus $\Pr\{S_1\} = 0.6$. Finally, the probability of experiencing state S_2 for any arbitrary trajectory is

$$\Pr\{S_2\} = 0.4 \cdot 1 + 0.4 \cdot 0.6 = 0.64$$

Example 3.2.4

When we defined the MDP for single–leg optimization in Section 2.3.2, we said that every selling episode for each flight starts at the initial state $S_0 = (\mathrm{T}, C)$ and terminates when the flight departs $S_T = (0, \cdot)$ or when the remaining capacity is exhausted $S_T = (\cdot, 0)$. As the MDP has no loops, the probability of observing a state while following a deterministic policy $a = \pi(s)$ can be computed by recursively evaluating the probability of experiencing previous states, as presented in the previous example. Which is the probability of observing any arbitrary state s in the MDP while following policy π?

Answer. Let $\Pr\{s|\pi, \boldsymbol{\psi}\}$ denote the probability of observing any state while following the policy π when demand behaves according to $d(f; \boldsymbol{\psi})$. Furthermore, for any state–action pair s, a, we can write the probability of observing the next state s' according to eq. (2.10) described in Section 2.3.2. The probability of experiencing any state s' depends on the transition probabilities from other states preceding this one and actions taken for each previous states according to the policy π. Mathematically,

$$\Pr\{s' \mid \pi, \boldsymbol{\psi}\} = \sum_{s \in \mathcal{S}} \Pr\{s \mid \pi, \boldsymbol{\psi}\} \cdot p(s'|s, \pi(s); \boldsymbol{\psi}).$$

Because any trajectory deterministically starts in state S_0, we can write $\Pr\{S_0 \mid \pi, \boldsymbol{\psi}\} \doteq 1$, allowing us to efficiently compute the recursive formula above for all states in a DP-like algorithm (starting from state S_0, we can forward propagate the probability of experiencing every state until we reach the terminal states).

Figure 3.5 illustrates the state–space distribution (i.e., the probability of observing each state) while following the optimal policy for a flight with $\mathrm{C} = 50$ units of capacity and $\mathrm{T} = 365$ time steps to departure when the demand follows $\nu_* = 80/365$ and $F_5^* = 2.75$. States that are more likely to be experienced during an episode are represented by a darker color. The first thing we bring to attention is that not all states have the same importance because some are more likely to be experienced than others. Furthermore, we also see that a significant amount of the states (43.8%) have less than 0.1% probability of being experienced during a sample episode (states represented in white). While DP dedicates equal computational resources (e.g., CPU time) for all states, RL cuts off the zones of the state–space that play little role in the average

performance, dedicating computational resources where its impact is more important.

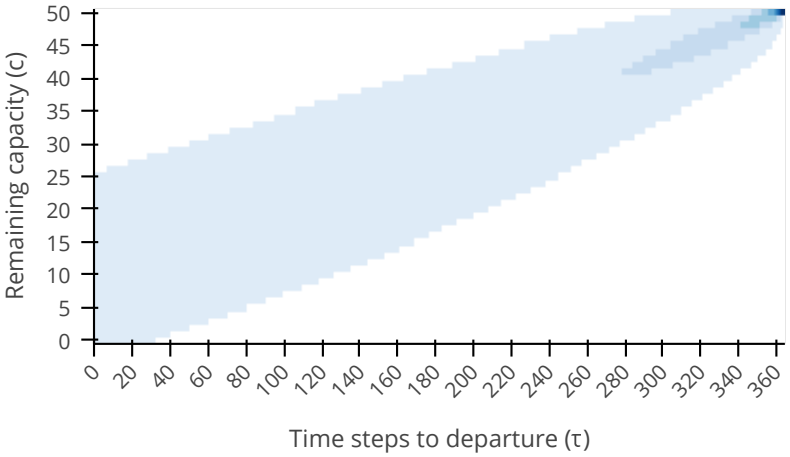


Figure 3.5: The state–space distribution for Example 3.2.4. States more likely to be experienced are represented with a darker color. States represented in white are have less than 0.1% change of being experienced.

3.2.4 Reinforcement learning with function approximation

The methods presented so far (DP and Q-Learning) hold individual estimations for every state–action pair, and, for this reason, they are known as tabular methods. In many tasks, the state space is combinatorial and large, and these methods cannot be used. For example, the number of board combinations for the game of Go is larger than the number of atoms in the visible universe [35]. In such cases, we cannot expect to find the optimal policy or optimal value function even within the limit of infinite data. Instead, we seek an approximate solution obtained with limited computational resources.

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

The problem with large state spaces is not only the memory needed for storing large tables but also the time required to fill them accurately: Many of the states encountered by the learning system may never be observed twice. Thus, making reasonable decisions requires the RL agent to generalize from past observations that share similarities, in some sense, to the current observation.

Fortunately, generalization from sample data is a subject extensively studied in many fields of science, such as machine learning, artificial neural networks (ANNs), and statistical curve fitting, and many ideas from these fields are exploitable in RL. This form of generalization is also known as *function approximation*, and it attempts to construct an approximation of the entire function based on examples. In principle, the agent can represent the value function by a parameterized function $q(s, a; \mathbf{w}) \approx q_\pi(s, a)$ where $\mathbf{w}$ is the weight vector. For example, $q(s, a; \mathbf{w})$ could be a

simple linear function in features of the state–action pair, with $\mathbf{w}$ as the vector of feature weights. Or, $q(s, a; \mathbf{w})$ could be computed by a deep ANN where $\mathbf{w}$ represents the connection weights in all layers. In theory, RL can use any of the methods studied in these fields. However, according to the application, some fit more easily than others. In Chapter 4, we pay particular attention to policy and value function approximation with ANNs because of their ability to generalize from sample data and because ANNs are derivable functions, thus fitting well in the RL framework, as we discuss in the next section.

Example 3.2.5

Consider a designer has the task of training an RL agent to win a car race. At each time step, the system collects the x-y coordinates of the car as well as the current x-y velocities, and the actions consist of accelerating or slowing down at the x-y axis. The rewards are straightforward, +1 if the car wins the race, zero otherwise. The corresponding state–action feature vector is $\mathbf{x}(s, a) = [x, y, v_x, v_y, a_x, a_y]$, where x, y are the coordinates, v_x, v_y are the velocities, and a_x, a_y represent the acceleration. The designer chooses to approximate the action–value function with linear approximation, i.e., $q(s, a; \mathbf{w}) = \mathbf{x}(s, a) \cdot \mathbf{w} \approx q_*(s, a)$, where the $\mathbf{w} \in \mathbb{R}^6$ is the weight vector. Suppose that for time step t, the evaluated feature vector corresponds to $\mathbf{x}(S_t, A_t) = [2, 1, -1, 1, 0, 2]$, and the trained weights are given by $\mathbf{w} = [1, -1.5, 0.5, 0.5, 3, 0.2]$. What is the estimated action–value $q(S_t, A_t; \mathbf{w})$?

Answer. As the designer defined the action value to be the inner product between the feature vector and the weight parameters, the estimated action value can be obtained with

$$\begin{aligned} q(S_t, A_t; \mathbf{w}) &= \sum_{i=0}^{5} x_i \, w_i \\ &= (2 \cdot 1) + (1 \cdot -1.5) + (-1 \cdot 0.5) \\ &\quad + (1 \cdot 0.5) + (0 \cdot 3) + (2 \cdot 0.2) \\ &= 0.9 \end{aligned}$$

Deep Q-Networks

One of the most difficult challenges in applying RL to real–world problems is deciding how to represent and store value functions and policies. Unless the state–action space is small enough to allow exhaustive representation through a table, some degree of function approximation is necessary. Function approximation relies on features that are often carefully handcrafted based on expert knowledge and intuition on the task being solved. These features must be readily accessible to the learning system and carry the necessary information for reaching the desired performance.

In one of the most remarkable studies in the field of RL, the agent reaches super–human performance in a large fraction of the 49 classic Atari 2600 games [40, 41] (see Figure 3.6), using the so-called *deep Q-network* (DQN) that combined Q-Learning with a *deep convolutional* ANN. Convolutional ANNs are networks specialized in processing spatial arrays of data such as images. DQN demonstrates that a generic ANN can automate the process of feature engineering.

Figure 3.6: Atari console. Image from px-here.com.

[40]: Mnih, Kavukcuoglu, et al. (2013), "Playing atari with deep reinforcement learning"
[41]: Mnih, Kavukcuoglu, et al. (2015), "Human-level control through deep reinforcement learning"

Humans playing any 49 Atari games see 210×160 pixel image frames with 128 colors at 60Hz, which, could be, in principle, used for training the agent (see Figure 3.7). However, to reduce memory and processing requirements, researchers preprocessed each frame, reducing them to an 84×84 array of luminance values. Because the full states of many of the Atari games cannot be observed only by the most recent frame, the four most recent frames were stacked, creating a total input dimension of 84×84×4. The actions naively corresponded to the joystick's buttons, such as up, down, right, or left. The rewards were +1 if the game's score increased from one time step to the next, -1 if lowered, and 0 otherwise. The DQN training used an ϵ-greedy policy with ϵ decreasing linearly over the first million frames and remaining low subsequently.

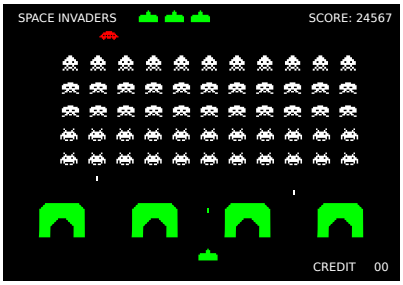


Figure 3.7: Screenshot of Space Invaders, one of the 49 Atari 2600 games which the RL agent achieved super–human performance. Image adapted from openclipart.org.

The ANN approximates the action–value function $q(s, a; \mathbf{w}) \approx q_*(s, a)$ through Q-Learning, i.e., for an experienced tuple S_t, A_t, R_{t+1}, S_{t+1}, we can write

$$q(S_t, A_t; \mathbf{w}) \approx \mathbb{E}_*\left[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a) \,\middle|\, S_t = s, A_t = a\right].$$

The ANN connection weights $\mathbf{w}$ are obtained by minimizing a sequence of the loss functions $\mathcal{L}_i(\mathbf{w}_i)$ that changes at each iteration i,

$$\mathcal{L}_i(\mathbf{w}_i) = \hat{\mathbb{E}}_\pi\Bigg[\Big(\underbrace{R_t + \gamma \max_a q(S_{t+1}, a; \mathbf{w}_{i-1}) - q(S_t, A_t; \mathbf{w}_i)}_{\text{TD-error}}\Big)^2\Bigg].$$

where $\hat{\mathbb{E}}_\pi[\cdot]$ indicates the empirical average over a finite batch of samples collected while following policy π. When optimizing the loss function, the weight parameters of target $R_t + \gamma \max_a q(S_{t+1}, a; \mathbf{w}_{i-1})$ are held fixed to the values of the previous iteration, i.e., $\mathbf{w}_{i-1}$. These targets depend on the ANN weights, differently than most supervised learning tasks where they are fixed before learning begins.

Differentiating the loss function with respect to the weights gives us,

$$\nabla\mathcal{L}_i(\mathbf{w}_i) = \hat{\mathbb{E}}_\pi\Big[\big(R_t + \gamma \max_a q(S_{t+1}, a; \mathbf{w}_{i-1}) - q(S_t, A_t; \mathbf{w}_i)\big)\nabla q(S_t, A_t; \mathbf{w}_i)\Big],$$

which can be optimized through the stochastic gradient descent according to the semi–gradient update rule

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \cdot \hat{\mathbb{E}}_\pi\Big[\big(R_t + \gamma \max_a q(S_{t+1}, a; \mathbf{w}_{i-1}) - q(S_t, A_t; \mathbf{w}_i)\big)\nabla q(S_t, A_t; \mathbf{w}_i)\Big]. \tag{3.6}$$

As presented in the training pseudocode in Algorithm 3, the training experience is collected through interactions with an Atari emulator, and the sampled transitions are stored in an *experience replay* buffer of fixed size. At each training iteration, a *mini–batch* of samples is extracted from the experience replay buffer, and the ANN weights are updated according to eq. (3.6). The experience replay buffer plays a central role in the training stability because it reduces the variance of updates by removing the correlation between successive updates in the weight vector (as it would happen in tabular Q-Learning presented in Section 3.2.2). Another important aspect of DQN is that the targets $R_t + \gamma \max_a q(S_{t+1}, a; \mathbf{w}_{i-1})$ are fixed to a previous estimation when optimizing the loss function, making these targets independent of the parameters being optimized. Such an approach simplifies the procedure while avoiding oscillations or divergence, bringing it closer to the simpler supervised–learning case while allowing the agent to bootstrap.

Algorithm 3: Deep Q-Learning for Atari, adapted from [40]

Input: The learning rate $\alpha \in [0, 1]$, the buffer capacity N and the exploration rate ϵ.

initialize the replay memory $\mathcal{M}$ with capacity N;
initialize the action–value function weights $\mathbf{w}$ arbitrarily;
Loop forever
  initialize $S_0 = X_0$ with the first image X_0 and preprocess sequence $x_0 = X(S_0)$;
  for $t = 1, \dots T$ **do**
    choose $A_t = \max_a q(X(S_t), a; \mathbf{w})$ with probability $1 - \epsilon$, otherwise choose a random action;
    take action A_t, observe reward R_{t+1} and next image X_{t+1};
    set $S_{t+1} = \{X_{t-2}, \cdots, X_{t+1}\}$ and preprocess $x_{t+1} = X(S_{t+1})$;
    store transition $(x_t, A_t, R_{t+1}, x_{t+1})$ in $\mathcal{M}$;
    sample random minibatch of transitions $(x_j, a_j, r_{j+1}, x_{j+1})$ from $\mathcal{M}$;
    set $y_j = \begin{cases} r_{j+1} & \text{for terminal } x_{j+1} \\ r_{j+1} + \gamma \max_a q(x_{j+1}, a; \mathbf{w}) & \text{for non-terminal } x_{j+1} \end{cases}$;
    perform a gradient descent step on $\left(y_j - q(x_j, a_j; \mathbf{w})\right)^2$ according to eq. (3.6);

3.3 Model–based and model–free reinforcement learning

One of the most important aspects of RL methods is their ability to learn directly from experience. As illustrated in Figure 3.9 (left), the experience used for training can be obtained through *direct* interactions with an environment (i.e., model–free or direct learning), or, conversely, as illustrated in Figure 3.9 (right), the experience may be *simulated* from interactions of a model of the environment (i.e., model–based or indirect learning). For example, the agent can learn from direct interactions, or, instead, a model of the environment can be built from some initially collected experience, and then used to generate the training data that feeds the RL agent [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

The best choice between model–free and model–based depends on the properties of the problem that is being addressed. How cheap is it to obtain fresh experience? Are the rules of the game known in advance? Are data from expert play available? How long can we accept the system to follow a suboptimal policy? Many of the recent advances of RL applied to real–world applications involve some degree of training through interactions with a model of the environment. For example, when training the RL agent to play Go [35] and StarCraft II [39], researchers combined model–based self–play and supervised learning from a dataset of human expert games. Or, when controlling magnetic

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

[39]: Vinyals, Babuschkin, et al. (2019), "Grandmaster level in StarCraft II using multi-agent reinforcement learning"

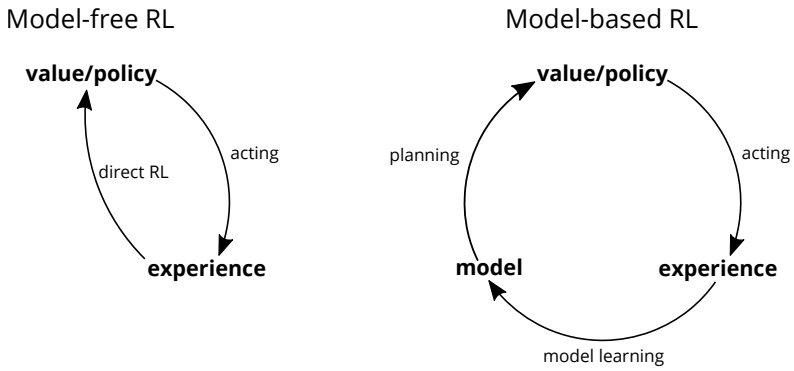


Figure 3.9: Model–based and model–free methods. Figure adapted from [27] under license CC BY-NC-ND 2.0.

fields of a tokamak[4] [42] (see Figure 3.8), where the laws of physics are known, researchers implemented a trustworthy simulated model of a real–world tokamak, which was then used for training the agent, and only after training, the agent was deployed in a real–world tokamak.

4: Tokamaks are devices that use magnetic fields to confine the plasma, currently being developed to produce controlled thermonuclear fusion power.

[42]: Degrave, Felici, et al. (2022), "Magnetic control of tokamak plasmas through deep reinforcement learning"

The RMS general layout displayed in Figure 1.4 is remarkably similar to model–based RL in Figure 3.9 (right). The RMS interacts with real–world customers, and then, from the collected experience, it builds a demand model used for planning the pricing policy through DP. Analyzing the RMS from this perspective opens the question of what other ways we could design it. Could RMS learn directly from experience without first building a demand model explicitly? Could we complement real data with simulated data for training? How much of each? The financial risks of training the agent directly through interactions with a real–world demand may be too consequential, suggesting that, like many other real–world applications of RL, at least some level of model learning may be necessary. Naturally, this raises the question of what model should be used. The number of options is substantial, ranging from traditional parametric models, such as the one developed in Chapter 2, or non-parametric models, that use artificial neural networks, each having pros and cons.

Figure 3.8: Experimental tokamak fusion reactor DIII-D. Figure from commons.wikimedia.org under license CC BY-SA 4.0.

3.4 Reinforcement learning applied to revenue management

Applying RL to airline RM has been a relatively new stream of research, where most of the studies can be grouped into three categories. The first seeks to exploit the ability of RL to deal with large–scale optimization tasks. Indeed, optimizing realistic demand models considering multiple fare classes, stochastic demand behavior, overbooking, and class–dependent random cancellations is a challenge in RM because they result in an optimization problem that is difficult to compute with classical

methods such as DP. Instead, RL could be used to optimize such problems [43–45].

Other studies point out the opportunities behind the online model–free nature of RL algorithms. As discussed in Section 1.4, the RMS typically uses historical booking data to fit a demand model, then it uses such a demand model to forecast future demand, and finally, the forecast is passed to an optimization routine that computes the optimal prices. New bookings are fed back into the system, and this process repeats cyclically over time. In contrast, *adaptive methods* update the pricing policy directly from observations, without assistance from the complex cycles of forecasting and optimization [1]. RL can be seen as a type of adaptive method because it can adjusts its pricing strategies without the need for explicit demand models. This has been perhaps one of the most influential motivations for its application in RM. Indeed, it is very tempting to remove the need for expert–designed models while simplifying the system's architecture and enabling the system to accommodate to new situations autonomously [17, 46, 47].

Another application of RL is not directly related to improving RMSs, but rather improving how we evaluate them. When deciding whether to upgrade the RMS, managers generally would like to have more information on how much revenue improvement such a change may bring, so they can better balance the benefits and risks of such a decision. One study suggests that RL can be used to evaluate the quality of different pricing policies directly from historical booking data before deployment [48].

In summary, the RM community has shown interest in RL mainly because of its capabilities of dealing with large–scale stochastic optimization problems, its model–free nature, and the possibility to support managerial decisions. However, we believe there is still space for many creative applications of RL algorithms in the field of RM. Chapter 4 illustrates yet another possible application: Some complex problems of interest in the literature of RM are addressed with expert–designed heuristics, that even though effective, they lack proof of optimality in part due to the natural complexity of the problem. When the problem is so complex that the optimal solution cannot be described by a simple set of rules, to make progress, we can only hope that the next idea performs better in practice than the previous ones. Such a manual search procedure requires a significant amount of expert time and dedication. Instead, we propose to use RL to automate the search for new policies to address these complex issues without explicit human help.

[43]: Gosavi, Bandla, et al. (2002), "A reinforcement learning approach to airline seat allocation for multiple fare classes with overbooking"

[44]: Gosavi (2004), "A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis"

[45]: Lawhead and Gosavi (2019), "A bounded actor-critic reinforcement learning algorithm applied to airline revenue management"

[1]: Talluri and Van Ryzin (2004), *The theory and practice of revenue management*

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"

[46]: Kastius and Schlosser (2021), "Dynamic pricing under competition using reinforcement learning"

[47]: Shihab, Logemann, et al. (2019), "Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking"

[48]: Ham (2021), *Know your worth: valuing new pricing policies with reinforcement learning*

3.5 The (possibly) false promise of model–free revenue management

Building RMSs capable of adapting autonomously to new market conditions directly from interactions with customers is a very appealing idea already explored in some studies [17, 47]. However, interactions with real customers are expensive, and airline RMSs need to optimize pricing policies with limited booking data (experience). Even though learning directly from interactions is a powerful idea, this approach has the issue regarding the quantity of experience needed to obtain acceptable performance.

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"
[47]: Shihab, Logemann, et al. (2019), "Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking"

To illustrate the issue, consider an airline optimizing one flight with infinite capacity, representing the scenario in which the flight's remaining capacity is much larger than the expected number of customer arrivals. Let's assume that the demand follows the exponential model defined in eq. (2.2), with the arrival rate $\nu_* = 1$, the price sensitivity $F_5^* = 1.95$, and the base fare $f_0 = \$50$. For simplicity, suppose that RMS and the customers interact only for a single time step before departure and that the system can choose between two fares $f \in \{\$70, \$90\}$. In this scenario, we can straightforwardly compute the action values with

$$q_*(\$70) = \$70 \cdot d(\$70) \approx 52.2,$$
$$q_*(\$90) = \$90 \cdot d(\$90) \approx 50.2.$$

The optimal fare is $f_* = \arg\max_{f \in \{\$70, \$90\}} q_*(f) = \$70$ because it delivers the highest expected revenue. This computation is simple because we know the true demand model, and the expectations can be obtained directly without interactions with the demand. Instead, a learning system would need to discover which of the two fares is optimal through trial and error (i.e., through sample data).

To perform learning, we use the same demand model, but instead, we generate examples through a sample model for each fare independently. As described in Chapter 2, this sample model generates customer arrivals according to the Poisson distribution, and the arriving customers decide or not to purchase at the selected fare according to an exponential probability. The collected revenue R_i of each experience is obtained trivially by the product of the number of bookings B_i observed during the time step and the selected fare, i.e., $R_i = B_i \cdot f$. Throughout the experiment, we gradually increase the number of samples n of interactions with a simulated demand, and the action–value

estimate for each fare can be obtained by simply averaging the revenue observed after each interaction, given by

$$Q(a) = \frac{1}{n} \sum_{i=1}^{n} R_i. \tag{3.7}$$

To correctly decide which is the optimal fare, the learning system does not need to compute the action–value function exactly but only that the estimated action value of the optimal fare $Q(\$70)$ is higher than the estimated value for the suboptimal fare $Q(\$90)$. Thus, we measure the learning efficiency as the probability that the action value for the optimal fare is larger than the suboptimal fare, i.e., $\Pr\{Q(\$70)) > Q(\$90)\}$. Figure 3.10 displays the result of this experiment. To learn the optimal policy with a probability of 0.95, the agent needs about 6000 examples for each fare. This number is so high because the variance of the learned objective (i.e., revenue) is much larger ($\mathrm{Var}[R_i \mid f = \$70] = 3637.5$) than the difference between the expectations $q(\$90) - q(\$70) = 2.0$. This example explains why learning through samples in RM is difficult: The demand behavior is very stochastic, requiring large amounts of observations to support any conclusion. Furthermore, adding more states and actions can only make this issue worse.

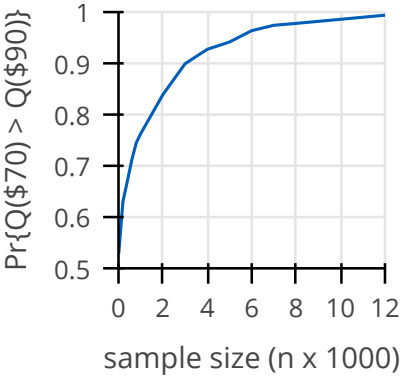


Figure 3.10: We compute the probability that the action–value estimation of the optimal fare is larger the suboptimal fare with respect the sample size.

In contrast, model–based RMSs are much more data–efficient because experts inject domain knowledge represented by the demand model, thus easing learning. However, this does not mean that all hopes are lost and that learning about the demand behavior without expert assistance is impossible. Perhaps, the data efficiency could be improved significantly by combining data from many flights, increasing the number of predictions the agent needs to do (such as the number of expected bookings) in the form of auxiliary tasks [49] or predicting the shape of the distribution with distributional RL [50, 51]. Even though this research direction deserves further investigation, it is not the focus of our work. Instead, in Chapter 4, we discuss another approach where a sample model is employed to train the agent to solve specific tasks prior to deployment.

[49]: Jaderberg, Mnih, et al. (2016), "Reinforcement learning with unsupervised auxiliary tasks"

[50]: Dabney, Rowland, et al. (2018), "Distributional reinforcement learning with quantile regression"

[51]: Gatti Pinheiro, Bondoux, et al. (2021), *Towards a distributional reinforcement learning approach to revenue management*

3.6 Summary

Reinforcement learning (RL) is a field of study that focuses on the optimal control of incompletely–known Markov decision processes and has strong connections to psychology and neuroscience. At the center of RL, we find the agent which learns through interaction with its surrounding environment. At each time step, the agent collects observations from the world, takes an action, and receives a scalar reward that indicates whether the agent is doing well or not in its task. In most tasks of interest, the agent must compromise short–term rewards to increase future rewards. Learning from trial–and–error and delayed rewards are the two most distinctive features of RL.

An RL system has several distinct blocks. The *policy* is the function mapping observations to actions; the *value function* indicates what is good or bad from the long–term perspective; the *model* represents anything the agent can use to predict how the environment will respond to its actions. The model is an optional block and often not present. RL methods using a model of the environment are known as *model–based* methods, in contrast to pure trial–and–error methods known as *model–free* methods.

The most popular RL methods use some level of value–function bootstrapping, which uses the approximations of future states to update the estimation of current states (updating a guess from guess). The quantity used for the updates is referred to as the time–difference error or TD-error. One of the most influential TD-methods is Q-Learning, which converges to the optimal action–value function if the learning rate decays appropriately and all states are visited continuously. This second condition can be satisfied by selecting nongreedy actions with a minimal probability, such as in the ϵ-greedy policy. This requirement gives rise to a conflict between *exploitation*, i.e., the act of maximizing reward, and *exploration*, i.e., improving the knowledge of how the environment behaves for nongreedy actions.

Many concepts behind RL, such as bootstrapping, come from the Bellman optimality equation and dynamic programming. The two most important differences between these two classes are that DP requires complete knowledge of the state–transition probability function and that the state–action space is small enough to allow the computation of the optimal action–value function. Instead, RL learns directly from interaction with the environment and it performs trajectory sampling, focusing computational resources where they are most needed.

If the state–action space is too large, some degree of function approximation is required, which can be done by approximating

the action–value function $q(s, a; \mathbf{w}) \approx q_*(s, a)$ with a parameter vector $\mathbf{w}$. In one of the most popular methods, known as deep Q-network (DQN), this parameter vector $\mathbf{w}$ represents the connection weights of an artificial neural network. The DQN has shown great success in many areas, perhaps most remarkably reaching super–human performance levels while playing Atari games.

In the past, researchers applied RL methods to revenue management (RM) because of their capacity to deal with large stochastic optimization problems, their model–free nature, and their ability to evaluate pricing policies before deployment. Ideally, we wish for an autonomous revenue management system that learns and adapts through its own experience. However, today, it is unclear if such an approach is feasible because modern model–free RL methods need larges amounts of data for learning, which are usually not available in airline RM. Instead, in Chapter 4, we show that RL can be used to address problems that until now rely exclusively on expert intuition and heuristic optimization.

Earning while Learning 4

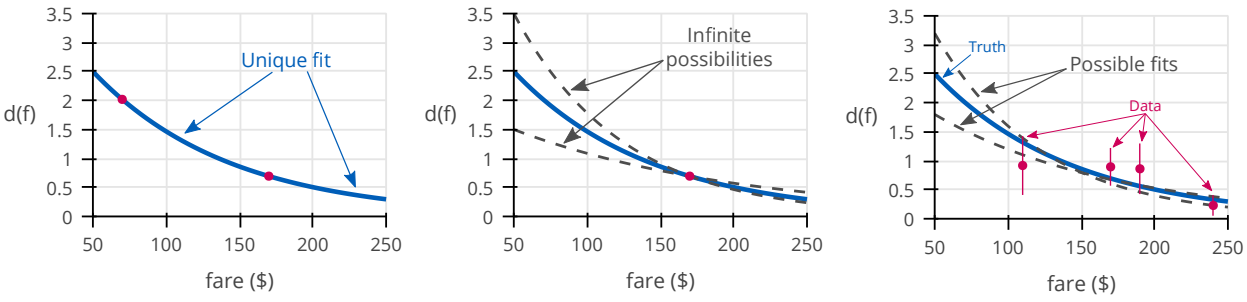
As discussed in Chapter 2, the pricing policy obtained with dynamic programming is optimal only if the demand behavior is known by the revenue management system. However, this assumption is unrealistic primarily because the presumed shape of the demand model may not reflect the real–world demand behavior. Even if assuming it does, RMS has only an estimate of the demand behavior obtained from historical booking data. Optimizing prices while interacting with an unknown demand behavior is called the *earning–while–learning* problem, and it is the central subject of this chapter.

4.1 Balancing earning and learning

When optimizing pricing policies, RMSs assume that the estimation of the demand behavior is perfect. Unfortunately, this is never true because the parameters of the demand model are obtained from historical bookings, thus subject to noise and estimation errors. For this reason, much of the research in RM has been dedicated to improving forecast quality [23].

[23]: Weatherford (2016), "The history of forecasting models in revenue management"

Perhaps one of the most surprising facts is that the RMS policy impacts the quality of the estimated demand model parameters. To understand why first consider Figure 4.1 (a). The demand model defined by the exponential curve could be, in principle, computed with any two points of this curve. If, for whatever reason, the system prices exclusively a single price point, as illustrated in Figure 4.1 (b), then it would be impossible to discover the true demand model because there is an infinite number of exponential curves that fit this unique point. However, RMS



(a) Unique fit. **(b)** Many fits. **(c)** RMS scenario.

Figure 4.1: Intuition to earning while learning in the context of an exponential demand model.

usually presents price variability, and the scenario from Figure 4.1 (b) is unrealistic. The scenario from Figure 4.1 (a) is also unrealistic since perfectly estimating two points of the demand curve would require an infinite amount of data. In practice, what happens in RMS is better represented in Figure 4.1 (c). The demand curve approximations of each price have inherited noise and uncertainty around them that is proportional to how many times RMS selected that price. Therefore, many possible curves could explain the data, and the forecasting module returns the best (that does not necessarily correspond to the true demand behavior). As the amount of data available for demand model estimation is fixed to the size of the historical database, RMS can only "trade" experimented prices (which price points are estimated) and how many times each price is selected (the amount of uncertainty around the estimation).

To further illustrate this fact, consider the experiment in Figure 4.2. We set the RMS policy to a fixed strategy that selects prices according to a predefined probability distribution. This policy interacts with a demand that behaves according to the exponential model. Then, the system stores the resulting bookings of these interactions in the historical database. This stored data is used to estimate the demand model parameters as usual. Finally, by choosing different pricing strategies, we can analyze how the system's policy can influence the quality of the estimated demand model parameters.

As a measure of the quality of the learned demand model, we choose the average accuracy of the estimated parameters given by

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \frac{\|\boldsymbol{\Psi}_i - \boldsymbol{\psi}_*\|}{\|\boldsymbol{\psi}_*\|}\right), \tag{4.1}$$

where $\boldsymbol{\Psi}_i = (\nu_i, \phi_i)$ is a sample estimation of the arrival rate and price sensitivity parameters, $\|\cdot\|$ denotes the Euclidean norm, and n is the number of collected samples. Ideally, the system should estimate the demand behavior parameters precisely, i.e., $\boldsymbol{\Psi}_i \approx \boldsymbol{\psi}_*$, which is equivalent to obtaining the parameter error

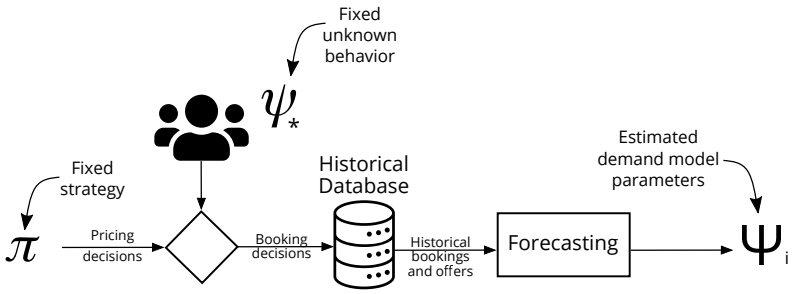


Figure 4.2: Experimental settings demonstrating how policy influences demand model learning.

$\|\Psi_i - \psi_*\|$ close to zero and the parameter accuracy close to one.

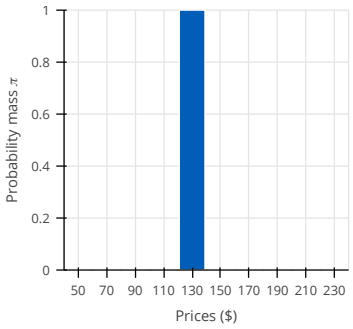
We analyze the revenue performance and the accuracy of the estimated demand behavior parameters of each of the five policies illustrated in Figure 4.4 (on page 62; a copy of the policies is available in Figure 4.3 on the margins of this page). The true demand behavior parameter is $\psi_* = (\nu_*, F_5^*) = (60/365, 2.75)$, and the system can choose among any of the ten price points $f \in \{\$50, \$70, \$90, \ldots, \$230\}$, where the lowest fare is $f_0 = \$50$. The flight has a capacity of C = 50, and the booking horizon has T = 365 time steps.

The first aspect we seek to demonstrate is that concentrating pricing decisions can negatively impact the accuracy of the estimated demand model parameters. For example, the policy in Figure 4.4 (a) chooses the price $\pi(s) = \$130$ for every encountered state, while the policy in Figure 4.4 (b) selects any fare with equal probability (e.g., $\epsilon = 1$). Even though policy (a) is better than policy (b) in terms of revenue performance, policy (a) is much worse in terms of parameter accuracy (0.676) than policy (b) (0.983).

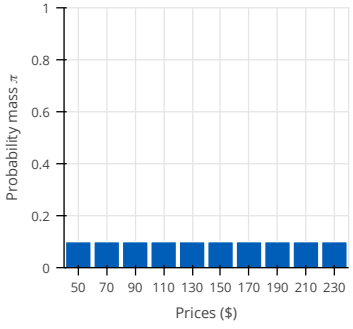
Furthermore, the prices selected by the system can also influence the quality of the estimated demand model. For example, consider the case in which two different policies choose between two prices, each with the same probability (50% each). Policy (c) selects the most extreme prices of the fare structure, while policy (d) selects the two center prices. We observe that the parameter accuracy for policy (c) is higher (0.998) than policy (d) (0.905), demonstrating that the prices the system selects play a central role in the quality of the demand model.

Finally, even the slightest levels of price experimentation can have strong positive effects on the average quality of the estimated demand model. For example, policy (e) selects the fare \$130 with a probability of 0.9 and fares \$110 and \$150 with an equal probability (0.05 each). Even though the accuracy of policy (e) is far better (0.864) than the accuracy of policy (a), which concentrates all choices into a single price point, both policies present a similar revenue performance.

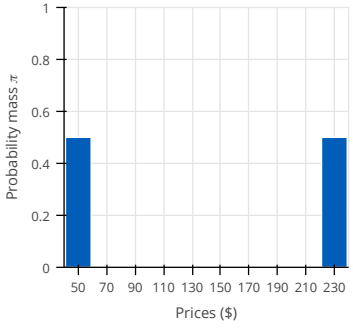
In principle, we seek the policy that generates the most revenue while keeping the accuracy of the estimated parameters under control. In Figure 4.4 (g), we plot how each policy behaves in terms of earning (revenue performance) and learning (accuracy of the learned parameters). Even though policy (a) presents the best revenue output, it is obviously not an achievable goal because the resulting accuracy of estimated parameters is very low and thus not a stable compromise. Policies (b) and (c) are far



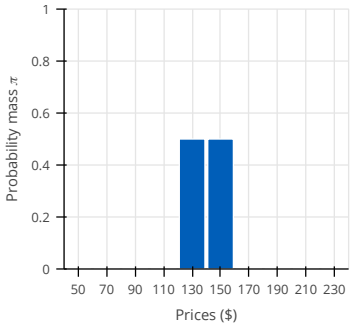
(a) accuracy = 0.676 ± 0.001.



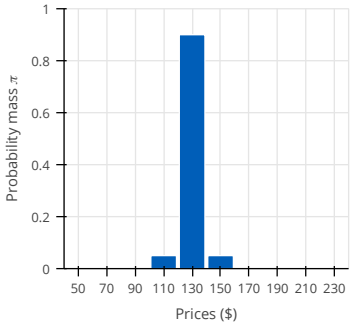
(b) accuracy = 0.983 ± 0.001.



(c) accuracy = 0.988 ± 0.001.



(d) accuracy = 0.905 ± 0.001.



(e) accuracy = 0.864 ± 0.001.

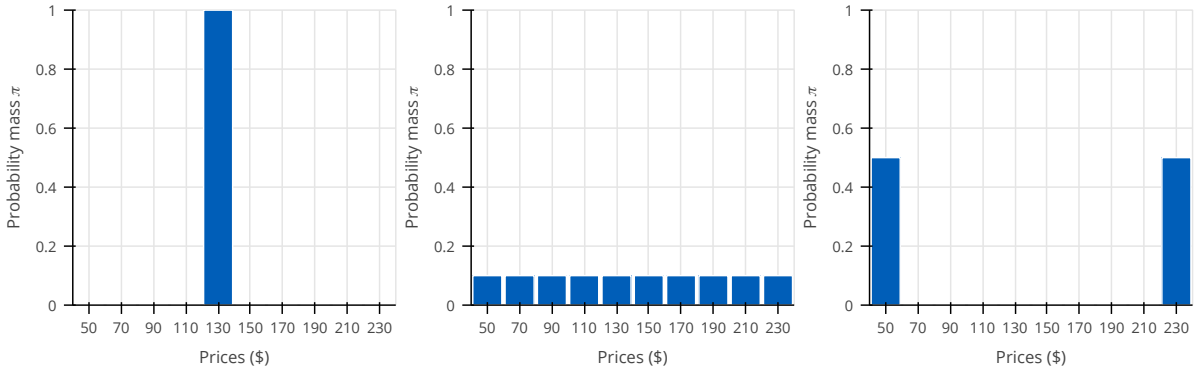
Figure 4.3: Policies used for earning wile learning experiment.

from optimality considering the revenue perspective, therefore undesirable solutions. Lastly, policies (d) and (e) show a high revenue performance while displaying a far better parameter accuracy. However, it is unclear whether the compromise found by these two policies is a stable one. We also illustrate where RMS currently fits in this balance (Figure 4.4 (f) displays the RMS rollout policy). Indeed, RMS finds an equilibrium point on its own due to its natural price variability, which results from misestimations of the true demand behavior: At time step t, the price corresponding to the estimated price sensitivity ϕ_t is \$150, but, in a later time step, say $t + k$, the system chooses price \$110 corresponding to a new estimated price sensitivity ϕ_{t+k}*. In other words, the average distribution of offers in the historical database, which can be roughly visualized by the rollout policy in Figure 4.4 (f), is a result of T distinct policies, each computed according to an instantaneous (mis)estimation of the demand model.

Although RMS was not designed to control the accuracy of the estimated parameters of the demand, its ability to find a balance between earning and learning is impressive and one of the reasons for being so difficult to outperform it in the *earning–while–learning* (EWL) problem (the next section presents an overview on historical methods). However, these experimental settings are particularly favorable to RMS because the number of offers in the historical database is large ($T^2 = 365^2$) for the number of optimized demand model parameters (i.e., two parameters $\psi = (\nu, \psi)$). In practice, real–world RMSs must estimate dozens of parameters from roughly the same quantity of data.

The central question is: How RMS should select prices while facing an unknown demand behavior, where the best estimates of this demand behavior can only be accessed from historical bookings? We have seen that current pricing decisions impact future knowledge of demand behavior. Thus, RMS may have to compromise immediate revenue by efficiently performing price experiments expecting that the information gained about the demand behavior will lead to better future pricing decisions. In the following sections, we review the various heuristic methods proposed by researchers for addressing this question and demonstrate how to adapt one of the most promising methods to the single–leg problem. We close by showing how RL methods can be used to tackle the EWL problem and by comparing the solution discovered with RL to the adapted method designed by experts.

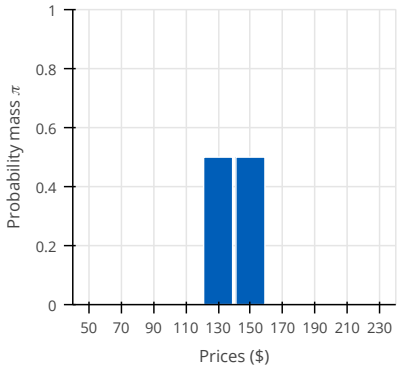
* The arrival rate for this experimental settings is relatively low, and there is little capacity constraining. Thus, the pricing optimization mostly leads to a pricing policy that selects a single price as discussed in Section 2.3.



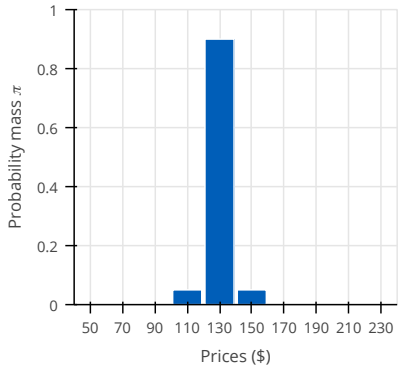
(a) accuracy = 0.676 ± 0.001.

(b) accuracy = 0.983 ± 0.001.

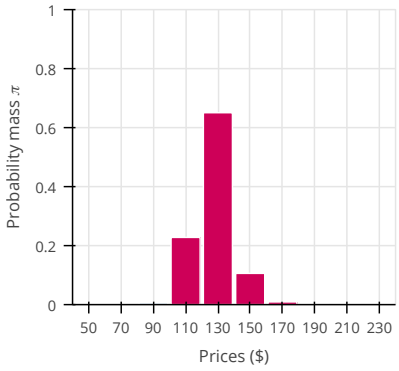
(c) accuracy = 0.988 ± 0.001.



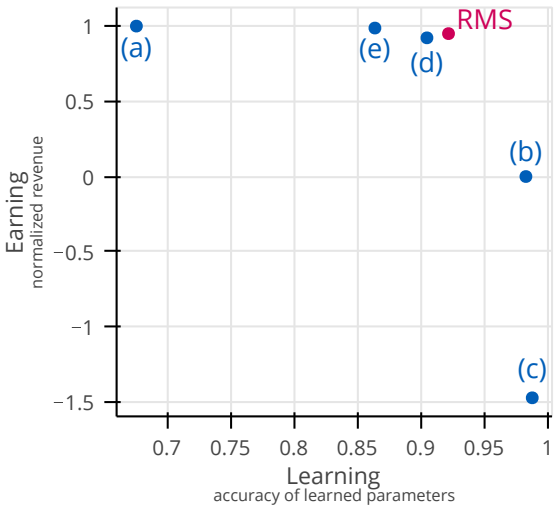
(d) accuracy = 0.905 ± 0.001.



(e) accuracy = 0.864 ± 0.001.



(f) RMS accuracy = 0.922 ± 0.001.



(g) Earning and learning. The revenue metric (vertical axis) is normalized so one represents the maximum possible expected revenue, and zero represents the revenue output of the random policy.

Figure 4.4: An illustration of how the system's pricing policy impacts demand learning.

4.2 Methods for earning and learning

The EWL problem relates to price optimization and control (earning), and to statistical learning and economics (learning about the demand behavior). Even though these fields are more than a century old each, for a long time, researchers studied them independently. In the past, researchers often assumed that some reliable previous knowledge about the demand behavior was available to the seller when performing price optimization. Perhaps the earliest works to combine these two research fields date back to 1954 when the first analytical analysis was developed considering dynamic pricing with unknown model parameters of the demand behavior [52]. This early result did not receive the deserved attention, maybe because of the technical difficulties of implementing pricing variation on the time (e.g., updating catalogs). Some attempts made by commercial firms to estimate the demand curve for their products were unsuccessful, typically due to difficulties in obtaining reliable estimates and changes in competitor pricing [53]. The rise of modern computers and the Internet made it much easier to track sales and change prices accurately, allowing the industry to address the EWL problem in practice.

[52]: Hansen (1954), *Report of the Uppsala meeting*

[53]: Hawkins (1957), "Methods of estimating demand"

The literature on the EWL problem can be organized according to the method used for learning about the demand behavior. Many studies consider that the demand behavior is learned through the Bayesian framework [54–56], others concentrate on the parametric framework (based on ordinary least–squares or maximum likelihood estimation) [57, 58], and yet some other studies are oriented towards non–parametric demand models [16, 59]. Even though we have a strong interest in parametric models, many discoveries generalize across frameworks, and thus they are presented below.

[54]: Lobo and Boyd (2003), "Pricing and learning with uncertain demand"
[55]: Chhabra and Das (2011), "Learning the demand curve in posted-price digital goods auctions"
[56]: Kwon, Lippman, et al. (2012), "Optimal markdown pricing strategy with demand learning"

[57]: Besbes and Zeevi (2011), "On the minimax complexity of pricing in a changing environment"
[58]: Keskin and Zeevi (2014), "Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies"

[16]: Chen and Gallego (2022), "A primal-dual learning algorithm for personalized dynamic pricing with an inventory constraint"
[59]: Chen and Gallego (2021), "Nonparametric pricing analytics with customer covariates"

One of the most influential early works addressing the EWL problem proposes to learn the demand model parameters in a Bayesian fashion [60]. In theory, dynamic programming could compute the optimal Bayesian policy, but no closed-form analytical expression exists in many situations of interest. Thus, the proposed workaround, known as certainty equivalent pricing (CEP), consists of selecting, at each time step, the price that would be optimal if the current demand model parameter estimates were correct. CEP is optimal under certain conditions, such as when only the intersect of a linear demand function is being estimated but is sub–optimal when learning both the slope and intersect of a linear demand model [61]. However, further analytical studies [15, 62] pointed to a more fundamental problem: The sequence of prices may converge to a sub–optimal price even

[60]: Aoki (1973), "On a dual control approach to the pricing policies of a trading specialist"

[61]: Chong and Cheng (1975), "Multistage pricing under uncertain demand"

[15]: McLennan (1984), "Price dispersion and incomplete learning in the long run"
[62]: Rothschild (1974), "A two-armed bandit theory of market pricing"

with unlimited data. This phenomenon was named *incomplete learning*, and it was theoretically demonstrated that the system must follow a policy that accumulates information about the demand behavior at an adequate rate without deviating too much from the greedy policy to avoid it [58]. Several heuristic methods combining this principle with a classical parametric demand model [18, 54, 63–65] have been studied, in which controlled variance pricing (CVP) [66] has arguably been the most influential method. The CVP algorithm imposes a constraint on the greedy pricing policy that requires the selected prices not to be too close to the average of previously selected prices. This constraint seeks to guarantee sufficient price dispersion by imposing a "taboo interval" over prices that the system is not allowed to choose from.

Another remarkable heuristic method found to outperform CVP in simulated studies and real–world benchmarks combines the revenue maximization and the uncertainty of the parameters of the learned model into a single objective function [67] in the form of

$$U(f) = R(f) - \eta \sum_i \frac{\sigma_i(f)}{\psi_i} \tag{4.2}$$

where η is the trade–off parameter, $R(f)$ represents the expected revenue for fare f, ψ_i is the i-th parameter of the estimated demand model $\boldsymbol{\psi} = (\psi_0, \psi_1, \ldots, \psi_k)$, and $\sigma_i(f)$ represents the corresponding estimated uncertainty of the i-th demand model parameter after selecting fare f. At each time step, the system selects the fare maximizing the objective function $U(f)$. The main idea of this heuristic is to include the uncertainty of the demand model parameters in the optimization objective as a penalty term (this linear combination of several goals, i.e., earning and learning, into a single objective function is similar to linear scalarization in multi–objective theory [68]). If the uncertainty over the parameters becomes too large, the system shifts from a revenue–maximizing to an information–maximizing fare. The trade–off parameter, η, can be used to tune the importance that the system should pay to the uncertainty of the demand model parameters.

Another stream of research brings attention to the similarities between the EWL problem and other problems investigated in machine learning, such as active learning [69] and the exploration–exploitation trade–off [62, 70, 71]. The literature on these topics is large, and some of the most popular techniques are Thompson sampling [72], upper confidence bound [73], and intrinsic curiosity [74]. However, in the EWL problem, a model of the demand

[58]: Keskin and Zeevi (2014), "Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies"

[18]: Keskin and Zeevi (2017), "Chasing demand: Learning and earning in a changing environment"

[54]: Lobo and Boyd (2003), "Pricing and learning with uncertain demand"

[63]: Besbes and Zeevi (2009), "Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms"

[64]: Boer and Zwart (2015), "Dynamic pricing and learning with finite inventories"

[65]: Ferreira, Simchi-Levi, et al. (2018), "Online network revenue management using thompson sampling"

[66]: Boer and Zwart (2014), "Simultaneously learning and optimizing using controlled variance pricing"

[67]: Elreedy, Atiya, et al. (2021), "Novel pricing strategies for revenue maximization and demand learning using an exploration-exploitation framework"

[68]: Hwang and Masud (2012), "Multiple objective decision making—methods and applications: A state-of-the-art survey"

[69]: Aviv and Pazgal (2005), "Dynamic pricing of short life-cycle products through active learning"

[62]: Rothschild (1974), "A two-armed bandit theory of market pricing"

[70]: Cope (2007), "Bayesian strategies for dynamic pricing in e-commerce"

[71]: Xia and Dube (2007), "Dynamic pricing in e-services under demand uncertainty"

[72]: Thompson (1933), "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples"

[73]: Auer, Cesa-Bianchi, et al. (2002), "Finite-time analysis of the multiarmed bandit problem"

[74]: Pathak, Agrawal, et al. (2017), "Curiosity-driven exploration by self-supervised prediction"

Table 4.1: Earning-while-learning literature review.

Study	Demand Model	Capacity Constraining	Multi–Flight
CEP [60]	Bayesian, non-parametric, parametric	✓	✓
CVP [66]	parametric		
(Chen and Gallego) [16]	non-parametric	✓	
(Elreedy et al.), see eq. (4.2) [67]	parametric		

behavior is present, and the model describes how each possible price relates to the others. This is very different from solving the exploration–exploitation trade–off in the general case, which often assumes that each possible choice delivers a response independent of the others. The ability to exploit the existence of a demand model is key to solving the EWL problem.

The key studies in Table 4.1 are organized according to the important requirements for successful integration with airline RM. For the first requirement, we seek methods that are directly compatible with parametric demand models. Furthermore, these methods must either comply with pricing under limited inventory capacity or be adaptable to current practices for pricing optimization. For our last requirement, these methods must consider that the historical booking data are generated by many flights simultaneously (multi–flight) or must be integrated easily into this scenario. These three requirements sum up the core conditions to apply the EWL methods to airline RM.

In the next section, we adapt the algorithm proposed in eq. (4.2) to multi–flight optimization, in which many flights of a single–leg must be optimized simultaneously. Unfortunately, to our knowledge, there are no extensions to this algorithm for addressing capacity constraints, making this new algorithm unrealistic for most scenarios of interest. However, this new algorithm demonstrates how RL solutions compare to expert–designed methods, providing a benchmark for the most simplistic scenarios. In short, if RL is better than heuristic methods when assuming unconstrained capacity, then, in our understanding, there is little reason to search for ways to “fix” these heuristic methods.

4.3 Optimizing for earning and learning

Most of the research developed around the EWL problem assumes that only the pricing decision of an individual product is being optimized at a time. However, in the airline industry, many flights depart every day, and the system must optimize all flights at each time step. In practice, when optimizing prices, modern RMSs assume that each active flight is independent of the others, splitting the problem of optimizing T active flights into T smaller optimization problems. This "divide to conquer" strategy may no longer be correct when tackling the EWL problem because each flight is no longer *truly* independent. The data collected for each flight are aggregated and used for model calibration. Then, the calibrated model is used to optimize the prices of future time steps. Thus, the pricing decision of each flight has longstanding consequences for the quality of future model calibration. Furthermore, each price decision contributes to the quality of the estimation of future demand models, making each decision a small part of a whole, suggesting that some collaboration across flights may be needed.

In this section, we show how to adapt the heuristic method in eq. (4.2) [67] to the single–leg problem assuming that capacity is *unconstrained*, i.e., the number of average customer arrivals is far smaller than the flights' total capacity, which implies that optimizing prices can be done simply by selecting the fare that satisfies $f_* = \arg\max_{f \in \mathcal{A}}[f\ d(f;\boldsymbol{\psi})]$ (no need to compute DP, as presented in Chapter 2) [75]. To start, the revenue and model uncertainty terms of eq. (4.2) need to be adapted to multi–flight optimization.

[67]: Elreedy, Atiya, et al. (2021), "Novel pricing strategies for revenue maximization and demand learning using an exploration-exploitation framework"

[75]: Gatti Pinheiro, Defoin-Platel, et al. (2022), "Optimizing revenue maximization and demand learning in airline revenue management"

First, let's address the revenue function. We define the multi–flight pricing policy as a multinomial distribution specified according to the parameter vector $\boldsymbol{\pi} = [\pi_0, \pi_1, \ldots, \pi_{n-1}]$. Each π_i component of this distribution represents the probability of selecting the price point f_i for every active flight for sale date t. Then, the expected revenue $R(f)$ can be written in terms of $\boldsymbol{\pi}$ as

$$R(\boldsymbol{\pi}) = \mathrm{T} \sum_{i=0}^{n-1} f_i\ \pi_i\ d(f_i;\boldsymbol{\psi}). \tag{4.3}$$

As explained above, without capacity constraints, the optimal pricing policy proposes the revenue–maximizing fare with probability 1 and 0 to all other fares, i.e., $\arg\max_{\pi} R(\boldsymbol{\pi})$ sets probability one to the fare maximizing $\arg\max_{f \in \mathcal{A}} f\ d(f;\psi)$ and zero to all others. For example, if the optimal fare corresponds to $f_* = f_2$,

then the vector π maximizing the revenue function is given by $\pi = [0, 0, 1, 0, \ldots, 0]$.

For the next step, the model uncertainty term $\sigma_\phi(f)$ of eq. (4.2) needs to be rewritten in terms of the policy $\boldsymbol{\pi}$. Without loss of generality, we consider that only the customer price sensitivity is being estimated and the customer arrival rate is perfectly known by the system. In principle, such an assumption is not necessary but greatly simplifies the mathematical derivations.

The precision of the estimated error $\sigma_\phi(\pi)$ is not derivable directly but it is lower bounded by the inverse of the Fisher information, i.e., $\sigma^2_\phi(\pi) \geq 1/\mathcal{I}(\boldsymbol{\pi})$ (in the literature of statistics this is known as the Cramér–Rao bound). One limitation of choosing to set the uncertainty term to its lower bound is that the system may not give the correct importance to the penalty term of eq. (4.2) when optimizing prices for the case where the true error is much larger than the lower bound. In such a scenario, the system will behave closer to the traditional revenue–maximizing objective rather than aggressively trying to improve the demand model uncertainty.

The Fisher information is given by

$$
\begin{aligned}
\mathcal{I}(\boldsymbol{\pi}) &\doteq -\mathbb{E}\left[\frac{\partial^2}{\partial^2\phi}\mathcal{LL}(\phi)\,\middle|\,\phi, \boldsymbol{\pi}\right] \\
&= -\mathbb{E}\left[\frac{\partial^2}{\partial^2\phi}\sum_{i=0}^{n-1}\sum_{j=t-\mathrm{T}}^{t-1}\left[b(j,f_i)\,\ln\left(d(f_i;v_*,\phi)\right) - o(j,f_i)\,d(f_i;v_*,\phi)\right]\,\middle|\,\phi, f_i \sim \boldsymbol{\pi}\right] && \text{use eq. (2.3)} \\
&= \mathbb{E}\left[\frac{\partial}{\partial\phi}\sum_{i=0}^{n-1}\left(\frac{f_i}{f_0}-1\right)\sum_{j=t-\mathrm{T}}^{t-1}\left[b(j,f_i) - o(j,f_i)\,d(f_i;v_*,\phi)\right]\,\middle|\,\phi, f_i \sim \boldsymbol{\pi}\right] && \text{first derivative} \\
&= \mathbb{E}\left[\sum_{i=0}^{n-1}\left(\frac{f_i}{f_0}-1\right)^2\sum_{j=t-\mathrm{T}}^{t-1} o(j,f_i)\,d(f_i;v_*,\phi)\,\middle|\,\phi, f_i \sim \boldsymbol{\pi}\right] && \text{second derivative} \\
&= \sum_{i=0}^{n-1} \underbrace{\mathbb{E}\left[\sum_{j=t+\mathrm{T}-1}^{t} o(j,f_i)\,\middle|\,f_i \sim \pi_i\right]}_{\text{expected historical offers at next time step } t+1} d(f_i;v_*,\phi)\left(\frac{f_i}{f_0}-1\right)^2 && \text{rearranging terms} \\
&= \sum_{i=0}^{n-1}\left(\underbrace{\sum_{j=t+\mathrm{T}-1}^{t-1} o(j,f_i)}_{\text{already observed}} + \underbrace{\mathbb{E}\left[o(t,f_i)\,\middle|\,f_i \sim \pi_i\right]}_{\text{expected future offers}}\right) d(f_i;v_*,\phi)\left(\frac{f_i}{f_0}-1\right)^2 \\
&= \sum_{i=0}^{n-1}\left(\sum_{j=t-\mathrm{T}+1}^{t-1} o(j,f_i) + \mathrm{T}\pi_i\right) d(f_i;v_*,\phi)\left(\frac{f_i}{f_0}-1\right)^2. && (4.4)
\end{aligned}
$$

Finally, the objective function $U(f)$ in eq. (4.2) can be rewritten as a function of $U(\boldsymbol{\pi})$ instead.

$$\arg\max_{\pi} \; U(\boldsymbol{\pi}) = R(\boldsymbol{\pi}) - \eta \frac{1}{\sqrt{\mathcal{I}(\boldsymbol{\pi})}}$$

subject to

$$\pi_0, \pi_1, \ldots, \pi_{n-1} \geq 0$$
$$\sum_{i=0}^{n-1} \pi_i = 1.$$

The trade–off parameter η can be manually calibrated to optimize revenue. When $\eta = 0$, the system maximizes revenue only, potentially decreasing the quality of the learned demand model. As the trade–off parameter η increases, the system focus will gradually shift from revenue to information maximization.

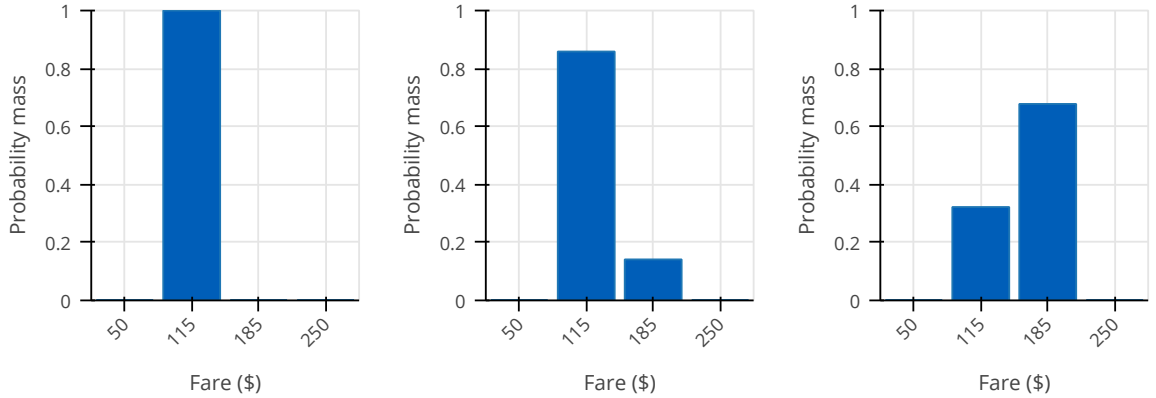
Once the probability distribution $\boldsymbol{\pi}$ is obtained, the system selects the fare for each active flight independently according to $\boldsymbol{\pi}$. The greatest weakness of this method is its reliance on the assumption of unconstrained capacity.

Example 4.3.1

Consider that, at a certain time step, the historical data consists of $\boldsymbol{o}_t = [0, 10, 15, 0]$ and $\boldsymbol{b}_t = [0, 3, 2, 0]$, the arrival rate is perfectly known $\nu_\star = 0.5$, and the fare structure is given by $\mathcal{F} = \{\$50, \$115, \$185, \$250\}$. What is the corresponding policy $\boldsymbol{\pi}$ when $\eta = 1500$? And, what if $\eta = 2000$?

Answer. First, we need to compute the price sensitivity parameter from historical bookings. This can be achieved by maximizing the log–likelihood in eq. (2.3), which gives us $\phi = 0.466$. Next, we can compute the policy $\boldsymbol{\pi}$ maximizing the objective function $U(\boldsymbol{\pi})$ using the algorithm described in this section. The resulting policies are displayed in Figure 4.5. When $\eta = 0$, the policy selects fare \$115 with probability one, greedily maximizing revenue according to the latest estimation of the price sensitivity parameter. As η increases, the system gradually changes its decisions to favor actions that bring more information about the demand behavior.

One could wonder why this example needs so "large" values of η. In principle, η needs to be adjusted for the range of values assumed by the revenue and information maximization objectives in eq. (4.2) (e.g., the revenue objective could assume values of magnitude 10^2 while the information objective could be in the range of 10^{-2}). In other words, if the designer wants η to vary in a specific interval, let's say $\eta \in [0, 1]$, then the objectives in eq. (4.2) would need to be normalized accordingly. We prefer to keep the original implementation because of its simplicity and adjust η to the problem instead.



(a) $\eta = 0$. **(b)** $\eta = 1500$. **(c)** $\eta = 2000$.

Figure 4.5: Policy distribution according to various the trade–off parameters.

Example 4.3.2

The equation eq. (4.4) gives the expected amount of observed information at the next time step after following the policy defined by π while considering the data already observed during the previous time steps. Note that the amount of information each offer adds is equal to $I(f) = d(f; v_*, \phi)(\frac{f}{f_0} - 1)^2$ and that the function $I(f)$ has a unique maximal value, meaning that there is a price that provides more information than any other. Assuming that the demand behaves according to $F_5^* = 1.9$ and the base fare is $f_0 = 50$, which are the fares that provide the lowest and the highest amount of information?

Answer. The lowest amount of information is obtained by the base fare $I(f_0) = 0$, which is not surprising because we assumed that the customer arrival rate is known. Intuitively, the base fare gives the most information about the customer arrival rate but does not bring any value to the estimation of the customer price sensitivity because the purchase probability is one at this fare.

The fare maximizing the amount of information can be computed by equating its derivative to zero.

$$\frac{d}{df}\left[\left(\frac{f}{f_0} - 1\right)^2 d(f; v_*, \phi)\right] = 0$$

$$2\left(\frac{f}{f_0} - 1\right)\frac{1}{f_0} d(f; v_*, \phi) - \left(\frac{f}{f_0} - 1\right)^2 d(f; v_*, \phi)\frac{\phi}{f_0} = 0$$

$$\phi\left(\frac{f}{f_0} - 1\right) = 2$$

$$f = \left(\frac{2}{\phi} + 1\right) f_0.$$

According to the settings, the information–maximizing fare is $f = (\frac{2}{0.770} + 1) \cdot 50 \approx \$180$. In Figure 4.6, we plot the function $I(f)$ for this example.

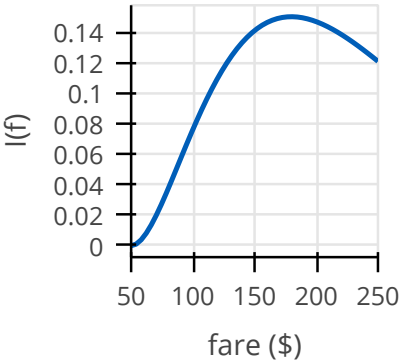


Figure 4.6: We plot the function $I(f)$ which returns the amount of information obtained by each fare.

4.4 A new perspective

We discuss in this section an alternative way to formulate the EWL in the single–leg problem [19]. The first main idea explored in this section relies on bringing in the estimated parameters of the demand model into the state representation, which allows us to train a single RL agent for a wide range of possible demand behaviors, as defined by ψ_*. We propose that instead of re-training the agent for each demand behavior it may encounter, the system can be trained just once in an off–line manner and then deployed to interact with real–world demand without further training. Thus, the agent must be ready for each possible demand behavior it may find (i.e., multitasking). By doing so, we also realize that, as the system does not have access to the true demand behavior, it must rely on an approximation of the demand behavior derived from the estimated demand model parameters (i.e., partial observability). The second main idea relates to the multi–flight optimization aspect of the single–leg problem, which allows us to define the MDP as a task that never ends, where the revenue maximization goal extends beyond the departure of individual flights (continuing task). Below, we review these three aspects of the EWL problem: multitasking, partial observability, and continuing task.

[19]: Gatti Pinheiro, Defoin-Platel, et al. (2022), "Outsmarting human design in airline revenue management"

Most importantly, the parameters of true demand behavior can assume any value, and each value defines a different *task*, i.e., different demand behavior parameters define different sets of transition probabilities and reward signals. Training a learning system to perform many tasks is related to the field of *multitasking* in reinforcement learning, which concentrates on building value functions [76] that can generalize across tasks so anything learned in one task can be exploited by another similar one. In our case, given two tasks, each defined by different demand behavior parameters, the transition probabilities and the expected revenue obtained by any policy should be close as long as the two demand behavior parameters are close to each other, suggesting the existence of a certain smoothness in the space of value functions. Because of this presumed smoothness, we can use the concept of *universal value* function approximators [77] that can exploit it.

[76]: Barreto, Dabney, et al. (2017), "Successor features for transfer in reinforcement learning"

[77]: Schaul, Horgan, et al. (2015), "Universal value function approximators"

In Chapter 2, the demand model parameters are estimated from historical booking data and, once calibrated, used to compute the transition probabilities essential to dynamic programming. However, these same parameters can be seen as an identification of the current state of the demand behavior, i.e., the true state of the environment could be represented by the true demand behavior parameters and the current inventory capacities. Indeed,

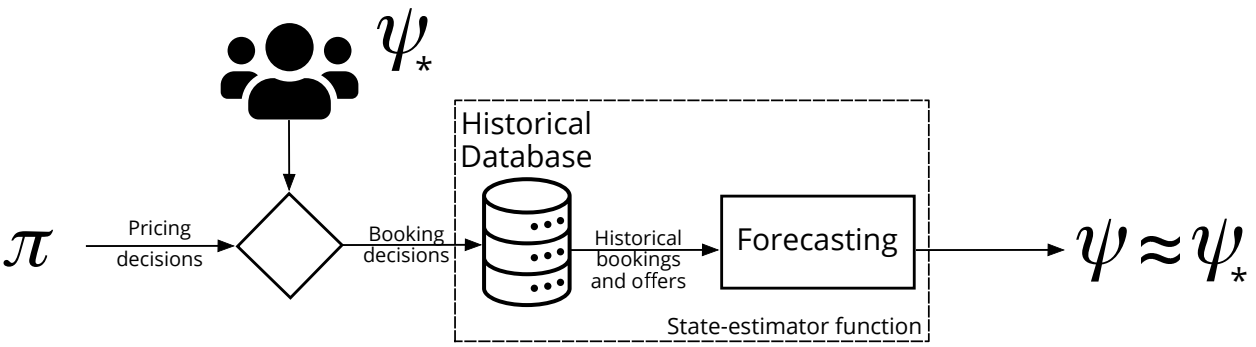


Figure 4.7: The state–estimator function. The system does not have access to the true state of the demand behavior ψ_*, but it does have an access to a noisy estimate of the true demand behavior $\psi \approx \psi_*$ that can be obtained through maximum likelihood estimation as described in Chapter 2. The combination of the historical database and parameter estimation plays the role of a state–estimator function.

if these two quantities were perfectly known, the system could compute the policy that is optimal for revenue maximization, referred to from now on as the *revenue–maximizing* policy. In reality, the system does not have access to the true state of the environment (i.e., not fully observable) because the true demand behavior parameters are unknown and must be estimated from historical booking data. This situation recalls *partial observability* [78] in Markov decision processes, where the system does not have access to the true state of the environment and must optimize actions based only on the noisy observations. The standard approach for addressing partially–observable problems is to build a state–estimator function that uses past observations to approximate a *belief state* of the true state of the environment. In our case, this is the role of the forecaster module, which uses past booking data to build a belief of the true demand behavior (see Figure 4.7). The particularity of the EWL problem is that the quality of this belief state depends on the policy the system follows.

[78]: Kaelbling, Littman, et al. (1998), "Planning and acting in partially observable stochastic domains"

The third and last aspect to consider is related to the *continuing* nature of the EWL problem. Traditional RMS optimizes the pricing policy by splitting the active flights into several independent optimizations, as illustrated in Figure 4.8 (left). We denote the i-th active flight in the selling horizon with the superscript $[i]$, such that $S_t^{[i]}$ implies the inventory state at the time step t for the i-th active flight in the selling horizon. For each flight, the pricing policy is optimized for the flight's departure and current capacity. At departure, the expected revenue for flight i is defined to be zero $v_*^{[i]}(S_T^{[i]}) \doteq 0$, reflecting that no new revenue is generated by a departed flight. However, this is not *really* true because the collected flight data still presents some value (which could be measured in terms of future revenue) for the period in which the system uses it for demand model learning. But measuring the intrinsic value of a flight's data in terms of revenue is no simple

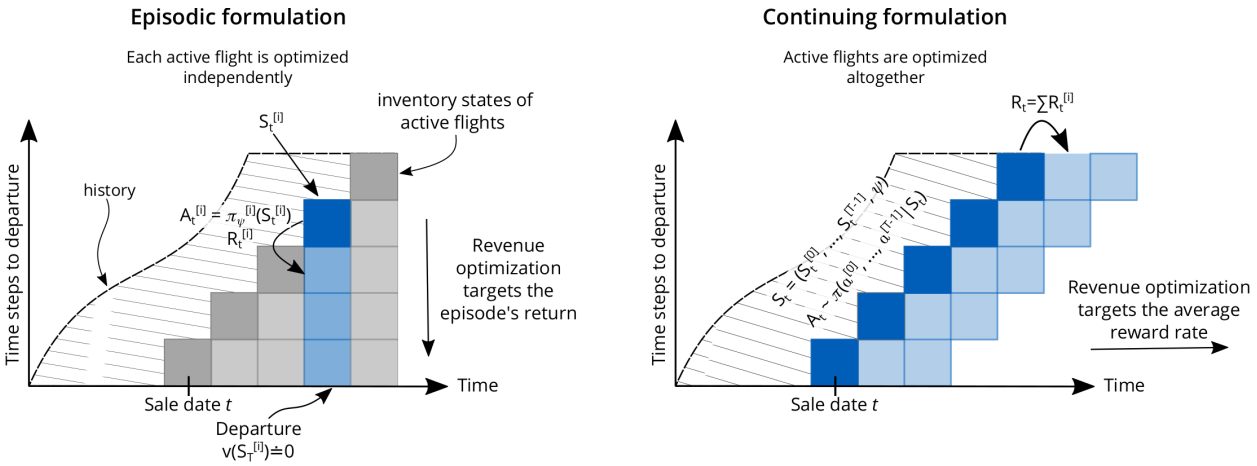


Figure 4.8: An illustration of the episodic and continuing formulations for the single–leg problem. Image from [19] under license CC BY 4.0.

task because it depends on the contents of the flight's data, how long the data are kept in the historical database and how these data complete the data obtained by other flights. In contrast, when optimizing the active flights altogether, as illustrated in Figure 4.8 (right), the state is now represented as a tuple of the inventory states of all active flights, i.e., $S_t = (S_t^{[0]}, \ldots, S_t^{[\mathrm{T}-1]}, \boldsymbol{\psi})$, and the reward signal is the sum of all immediate rewards obtained by each active flights, i.e, $R_t = \sum_{i=0}^{\mathrm{T}-1} r(S_t^{[i]}, A_t^{[i]})$. The value function for the latest observed state $v(S_t)$ represents the sum of revenues obtained for the next days until the end of time, i.e., $v(S_t) = \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1}]$. Note that this new formulation of the Markov decision process does not present start or end states, and we call it the *continuing formulation*. Furthermore, this formulation enables the intrinsic value of information to be captured by the value function because this function estimates the value of current flights for an infinite horizon. Consequently, the system only needs to search for a policy that improves the value function (long–term revenue when including exploration) rather than a policy that balances exploration and exploitation.

In short, the properties of the EWL problem applied to the single–leg can be summarized as

- **Multitasking**: The agent is trained to react to many possible situations (or many possible values of ψ_*), in contrast to RMS that plans a pricing policy according to a specific set of transition probabilities;
- **Partial–observability**: The system does not have access to the true state of the demand behavior and must rely on a noisy estimate;
- **Continuing task**: Optimization is performed over infinite horizons rather than the classic episodic definition presented in Chapter 2.

Our attention now turns to optimize a pricing policy under the above conditions. The problem's dimensionality makes it impossible to solve with exact methods such as DP. The cardinality of the action space alone is $|\mathcal{A}|^{\mathrm{T}}$, making it impractical even for toy problems[1]. Furthermore, the observation space has a finite component with size $(\mathrm{T} \cdot C)^{\mathrm{T}}$ and a continuous component representing the estimated demand model parameters. The second observation component, $\boldsymbol{\psi}$, makes it far too complicated to compute the state–transition probability function because it depends on the distribution of the future estimated demand model parameters. These estimated demand model parameters are a result of interactions between complex systems (the historical data and the demand model optimization method), thus hard to describe analytically. Instead, it is trivial to build a sample model for training.

1: "Toy problems" are problems that have no immediate practical interest, often used as an expository problem that illustrates a trait shared with another more complicated problem.

The above problem description fits RL, leaving the question of which RL method is suitable to address it. Among the many RL methods in the literature, algorithms that use policy parameterization, such as actor–critic methods, are well adapted to large action spaces [79], thus in the center of interest. In the next section, we present the theory behind the actor–critic framework and how it can be adapted to solve the EWL in the single–leg problem.

[79]: Degris, White, et al. (2012), "Off-policy actor-critic"

4.5 A brief review of actor–critic methods

Actor–critic methods refer to a class of RL methods that learns the policy and value functions, contrary to Deep Q-Networks, presented in Chapter 3, that learn only the action–value function. This section aims at developing the intuition of how these methods work by reviewing the basic principles behind actor-critic algorithms.

4.5.1 Stochastic policies

Throughout Chapter 2, the policy is defined as a function that maps *deterministically* states to actions $\pi : \mathcal{S} \to \mathcal{A}$. In such a case, the policy function cannot exist without the action–value function (i.e., $\pi(s) = \arg\max_a q(s, a)$), and computing the action–value function $q(s, a)$ exactly or as an estimate has been the focus of dynamic programming and Q-Learning. Alternatively, the policy function can represent the probability of selecting an action given a state $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ [27], known as a *stochastic* policy.

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

Perhaps one of the most common ways to model a stochastic policy is by defining it in terms of the soft–max distribution

$$\pi(a|s;\boldsymbol{\theta}) \doteq \frac{e^{h(s,a;\boldsymbol{\theta})}}{\sum_{a' \in \mathcal{A}} e^{h(s,a';\boldsymbol{\theta})}} \tag{4.5}$$

where $h(s, a; \boldsymbol{\theta}) \in \mathbb{R}$ is the action *preference*, in which actions with the highest values have more probability of being selected. These preferences themselves may be parameterized arbitrarily with a simple linear model, such as

$$h(s, a; \boldsymbol{\theta}) \doteq \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}(s, a) \tag{4.6}$$

where $\mathbf{x}(s, a) \in \mathbb{R}^d$ is a *feature vector* for state–action pair (s, a). The action preferences can be computed by a deep artificial neural network, where $\boldsymbol{\theta}$ represents the weights of all network connections [35, 80].

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"
[80]: Berner, Brockman, et al. (2019), "Dota 2 with large scale deep reinforcement learning"

There are many reasons to prefer modeling a stochastic rather than a deterministic policy. The first one is that stochastic policies can be differentiable functions, allowing us to use the policy gradient theorem (which is reviewed in the following sections). With the policy gradient theorem, the RL agent can learn a policy directly without learning the action–value function first. Furthermore, stochastic policies can represent deterministic ones, while the opposite is generally not possible. For example, if the optimal policy is deterministic then the system can set the action preferences to be significantly higher for the optimal actions. Another advantage is that, in problems with significant function approximation, the best approximate value of the optimal policy may be a stochastic policy, e.g., bluffing in Poker is better done with a certain probability rather than deterministically. RL methods using stochastic policies can explicitly model (and learn!) these probabilities [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

But perhaps the most straightforward reason to model a stochastic policy is that it might be a simpler function to approximate. Problems vary in complexity regarding the functional form of the policy and action–value functions, and approximating the policy rather than action values may be easier for some problems. For example, in Tetris, policy gradient methods often yield a superior performance while learning much faster than action–valued methods [81].

[81]: Şimşek, Algorta, et al. (2016), "Why most decisions are easy in tetris – and perhaps in other sequential decision problems, as well"

4.5.2 Policy gradient methods

Policy gradient methods seek to approximate the policy function directly without requiring the computation of the action–value function. The value functions can still be used for learning the policy, but they are not necessary for action selection. These methods approximate the parameters of the stochastic policy function $\pi(a \mid s; \boldsymbol{\theta})$ that maximizes a performance measurement[2] $J(\boldsymbol{\theta})$ while updating the parameter $\boldsymbol{\theta}$ according to stochastic gradient ascent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}), \tag{4.7}$$

According to the policy gradient theorem (see [27]), we can write

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a \mid s; \boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a \mid S_t; \boldsymbol{\theta}) \right], \end{aligned} \tag{4.8}$$

where $\mu(s)$ represents the on–policy distribution under π, such that $\mu(s) \geq 1$, $\sum_s \mu(s) = 1$, where this function often represents the fraction of the time step in s when following policy π. We could stop here and write the all–actions policy gradient algorithm [82] as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \sum_a q(S_t, a; \mathbf{w}) \nabla \pi(a \mid S_t; \boldsymbol{\theta}), \tag{4.9}$$

where $q(s, a; \mathbf{w})$ is an approximation of $q_\pi(s, a)$.

2: There are two main choices for the performance metric $J(\boldsymbol{\theta})$, one for the episodic and another for the continuing task formulation (in continuing tasks, there are no start nor terminal states). For now, we focus on the episodic formulation, where the performance metric is defined as $J(\boldsymbol{\theta}) \doteq v_{\pi\theta}(s_0)$, where $v_{\pi\theta}$ is the true value function for π_θ, and the policy defined by $\boldsymbol{\theta}$. We save the discussion about the continuing formulation for later in this work.

[82]: Sutton, McAllester, et al. (1999), "Policy gradient methods for reinforcement learning with function approximation"

Example 4.5.1

Suppose that, in a problem, the RL agent needs to decide between two actions (let's say, *right* or *left*). The designer decides to use an artificial neural network (ANN) with a single unit (or neuron) for this task, where the output of the ANN represents the policy function as the probability of taking the *right* action (R). There are many possible choices of activation functions that this single unit could use (e.g., hyperbolic tangent, rectified linear activation), and, because the designer is modeling a probability, he believes that the sigmoid function is the most appropriate choice. In other words, the designer chooses the output of the network to be given by

$$\pi(a = \mathrm{R} \,|\, s; \theta) = \frac{1}{1 + e^{-\theta \mathrm{x}(s)}}, \tag{4.10}$$

where $\mathrm{x}(s) \in \mathbb{R}$ is the feature vector for state s that the designer may choose casually (e.g., it could be a linear combination of the states' attributes). Naturally, the probability of selecting the *left* action is given by $\pi(\mathrm{L} \,|\, s; \theta) = 1 - \pi(\mathrm{R} \,|\, s; \theta)$. Assuming such a policy definition, which is the all–actions policy gradient update function from eq. (4.9)?

Answer. The first step for writing the all–actions policy update equation consists in computing the gradient of the policy function for the learned parameter θ, which can be written as

$$\begin{aligned}
\nabla_\theta \pi(a = \mathrm{R} \,|\, s; \theta) &= \frac{\partial}{\partial \theta} \left[\frac{1}{1 + e^{-\theta \mathrm{x}(s)}} \right] \\
&= (-1) \cdot \frac{-\mathrm{x}(s) e^{-\theta \mathrm{x}(s)}}{\left(1 + e^{-\theta \mathrm{x}(s)}\right)^2} \\
&= \mathrm{x}(s) \cdot \frac{1}{1 + e^{\theta \mathrm{x}(s)}} \cdot \frac{e^{-\theta \mathrm{x}(s)}}{1 + e^{-\theta \mathrm{x}(s)}} \\
&= \mathrm{x}(s)\, \pi(\mathrm{R} \,|\, s; \theta)\, (1 - \pi(\mathrm{R} \,|\, s; \theta)) \\
&= \mathrm{x}(s)\, \pi(\mathrm{R} \,|\, s; \theta)\, \pi(\mathrm{L} \,|\, s; \theta).
\end{aligned}$$

Analogously, we can write

$$\begin{aligned}
\nabla_\theta \pi(a = \mathrm{L} \,|\, s; \theta) &= \frac{\partial}{\partial \theta} \left[1 - \pi(a = \mathrm{R} \,|\, s; \theta) \right] \\
&= -\mathrm{x}(s)\, \pi(\mathrm{R} \,|\, s; \theta)\, \pi(\mathrm{L} \,|\, s; \theta),
\end{aligned}$$

which gives us the final update equation

$$\theta_{t+1} = \theta_t + \alpha\, \pi(\mathrm{R} \mid S_t; \theta)\, \pi(\mathrm{L} \mid S_t; \theta)\, x(S_t)\, \big[q(S_t, \mathrm{R}; w) - q(S_t, \mathrm{L}; w)\big].$$

The above update equation makes very clear how the all–actions method work. At every time step, the weight parameter θ increases when the estimation of the action value for *right* is greater than *left* ($q(s, \mathrm{R}; w) > q(s, \mathrm{L}; w)$), consequently strengthening the probability of selecting *right*. On the contrary, the weight parameter θ decreases when the action estimation of *left* is larger than *right* ($q(s, \mathrm{L}; w) > q(s, \mathrm{R}; w)$), thus reducing the probability of selecting *right*.

REINFORCE

The REINFORCE method is an influential policy gradient algorithm [83], also known as a Monte–Carlo algorithm because it uses only the episode return for learning, i.e., it does not perform bootstrapping like in DP or Q-Learning. Continuing from eq. (4.8), we can write

[83]: Williams (1992), "Simple statistical gradient-following algorithms for connectionist reinforcement learning"

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[\sum_a \pi(a \mid S_t; \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a \mid S_t; \boldsymbol{\theta})}{\pi(a \mid S_t; \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t \mid S_t; \boldsymbol{\theta})}{\pi(A_t \mid S_t; \boldsymbol{\theta})} \right] && \text{replace } a \text{ by the sample } A_t \sim \pi \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}) \right] && \text{use } \frac{\nabla x}{x} = \nabla \ln x \\
&= \mathbb{E}_\pi \left[G_t \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}) \right] && \text{use } q_\pi(S_t, A_t) = \mathbb{E}_\pi[G_t \mid S_t, A_t],
\end{aligned} \tag{4.11}$$

which yields the REINFORCE update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\, G_t\, \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}_t).$$

This formulation is appealing because of its intuitive form. Each increment is proportional to the return G_t in the direction of the probability of taking the chosen action divided by the probability of taking that action. When the return is positive, the update increases the probability of repeating the action A_t, otherwise decreasing it when the return is negative.

The REINFORCE method suffers mainly from two issues. The first one is that the agent must wait until the end of the episode to compute the return G_t, thus unsuitable for continuing tasks

(or tasks that never end). The second issue relates to the variance of the return G_t. The return is a product of many stochastic interactions with the environment. Such stochasticity may come from the policy or the probabilistic nature of transitions. As a result, the learning process must average across many episodes to converge properly, requiring small learning rates and large amounts of experience.

Actor–critic

The actor–critic methods use the value functions to evaluate the expectation of the policy gradient theorem in eq. (4.8). These methods compute two parameterized functions: The actor approximates the policy function $\pi(a \mid s; \boldsymbol{\theta}) \approx \pi(a \mid s)$, and the critic estimates the value function $v(s; \mathbf{w}) \approx v_\pi(s)$. Like in TD(0) and Q-Learning, these methods also use the temporal–difference error for updating the value estimate for a state by using the value estimates of future states. This bootstrapping significantly reduces the variance of returns, thus accelerating learning on the cost of the introduced bias of the value function misestimations that can decrease the method's asymptotic performance. Generally, this is a beneficial trade–off [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

To derive our first actor–critic algorithm [27], we can continue from eq. (4.11)

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[(R_{t+1} + \gamma v_\pi(S_t) - v_\pi(S_{t+1})) \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}) \right] \\
&\approx \mathbb{E}_\pi \left[(R_{t+1} + \gamma v(S_t; \mathbf{w}) - v(S_{t+1}; \mathbf{w})) \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}) \right] && \text{use } v(s; \mathbf{w}) \approx v_\pi(s) \\
&= \mathbb{E}_\pi \left[\delta_t \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}) \right] && \text{use eq. (3.2)}
\end{aligned} \tag{4.12}$$

where $\delta_t = R_{t+1} + \gamma v(S_t; \mathbf{w}) - v(S_{t+1}; \mathbf{w})$ is the TD-error[3] presented in Chapter 3. The policy gradient update rule is given by

3: For simplicity, only the *one–step* temporal–difference error is presented, but many choices for computing this quantity are available in the literature [84], ranging from n-step bootstrapping to eligibility traces, which are outside of our scope.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\theta \delta_t \nabla \ln \pi(A_t \mid S_t; \boldsymbol{\theta}_t), \tag{4.13}$$

and, similarly, the update for the value function weights is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_w \delta_t \nabla v(s; \mathbf{w}_t), \tag{4.14}$$

where α_θ and α_w are the learning rate for the actor and the critic respectively.

Actor–critic in practice

The actor–critic method described in the previous section updates the learned parameters after every experience collected from the environment. However, such an approach is not convenient for some of the most popular function approximation methods[4] (such as deep learning), being suitable only for didactic purposes. In practice, most popular actor–critic methods collect a batch of data across several interactions with the environment before updating the actor and the critic parameters. The idea is to approximate the expectation in eq. (4.12), by

$$\nabla J(\boldsymbol{\theta}_t) \approx \hat{\mathbb{E}}_\pi \left[\delta_t \nabla \ln \pi(A_t|S_t;\boldsymbol{\theta}_t) \right]. \tag{4.15}$$

where $\hat{\mathbb{E}}_\pi[\cdot]$ indicates the empirical average over a finite batch of samples collected while following policy π. The parameters for the value function $v(s;\mathbf{w})$ can be updated according to

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \hat{\mathbb{E}}_\pi \left[\left(R_{t+1} + \gamma v(S_{t+1};\mathbf{w}_t) - v(S_t;\mathbf{w}) \right)^2 \right], \tag{4.16}$$

which can be computed through stochastic gradient descent. Note that the target $R_{t+1} + \gamma v(S_{t+1};\mathbf{w}_t)$ remains fixed and only the weights of $v(S_t;\mathbf{w})$ are adjusted by the optimization algorithm. This allows turning an unsupervised learning problem, with unknown value targets, into a supervised learning problem, where the current value estimations are adjusted to fixed targets from a previous estimation.

In general, actor–critic algorithms collect a batch of experiences, then compute eqs. (4.15) and (4.16), and finally deploy the new policy function for renewed data collection. Figure 4.9 illustrates the typical training loop. The actor–critic algorithms can differ in how experiences are collected and used for training. For example, many algorithms may reuse experience by introducing a replay buffer [85, 86], while others may distribute the training process by computing the gradients remotely on each rollout worker [87, 88].

4.5.3 Continuing tasks

When defining goals in episodic Markov decision processes (MDPs), the discount rate can be set to one $\gamma = 1$ (see eq. (2.15)) because it is possible to search for a policy π maximizing it as long as the return G_t is bounded (i.e., it does not diverge).

4: Most function approximation methods require identically and independently distributed input data, which is not the case for RL, which collects data through trajectories in the state–action space.

[85]: Casas (2017), "Deep deterministic policy gradient for urban traffic light control"
[86]: Haarnoja, Zhou, et al. (2018), "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor"
[87]: Wijmans, Kadian, et al. (2019), "Ddppo: Learning near-perfect pointgoal navigators from 2.5 billion frames"
[88]: Horgan, Quan, et al. (2018), "Distributed prioritized experience replay"

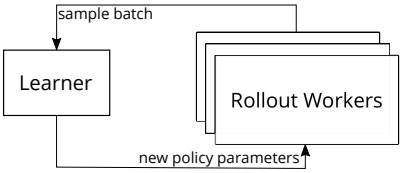


Figure 4.9: The rollout workers perform interactions with the agent's current policy and store the collected trajectories in a training batch. This data collection is often performed in parallel across many instances. The collected training batch is sent to the trainer that will compute eqs. (4.15) and (4.16), updating the agent's policy and value function. This new policy is submitted to the rollout workers for further collection of experience. This loop continues until convergence or any other stop criteria defined by the designer.

However, in the case of continuing problems that go on and on forever without any particular start or terminal states, setting the discount rate to one is not possible anymore, and an appropriate value should be chosen carefully. If too small, the agent can be too shortsighted, preferring short–term rewards to long–term goals. In an alternative classical setting for formulating objectives in MDPs that overcomes this limitation, known as the *average reward*, the agent pays attention to delayed rewards as much as it does to immediate rewards [27]. The average reward defines the quality of a policy $r(\pi)$ as the average reward rate while following that policy, denoted as

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

$$\begin{aligned} r(\pi) &\doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] \\ &= \lim_{t\to\infty} \mathbb{E}\left[R_t \mid S_0, A_{0:t-1} \sim \pi\right], \end{aligned}$$

where the expectation is conditioned on the initial state S_0 and the subsequent action sequence $A_0, A_1, \ldots, A_{t-1}$, which are selected according to policy π. This formulation assumes that the start state S_0 and the policy's early decisions have only a temporary effect: In the long run, the expectation depends only on the policy and the transition probabilities. This property is known as *ergodicity*, and it is sufficient to guarantee the existence of the limits in the above equations [27].

With the average reward setting, the differential value function $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$ is defined in terms of the *differential return*

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \ldots. \tag{4.17}$$

The differential value functions can be also written in terms of the Bellman equation

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \left[r(s, a) - r(\pi) + v_\pi(s')\right] \\ q_\pi(s, a) &= \sum_{s'} p(s' \mid s, a) \left[r(s, a) - (\pi) + \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\right], \end{aligned}$$

and there is also the differential TD-error, which can be written as

$$\delta_t \doteq R_{t+1} - \bar{R}_{t+1} + v(S_{t+1}; \mathbf{w}_t) - v(S_t; \mathbf{w}_t)$$

where $\bar{R}_t$ is an estimate at time t of the average reward $r(\pi)$. There are several methods for computing the differential return [89], and it has already been extended successfully to the actor–critic framework [90]. One of the simplest ways to estimate the average reward $r(\pi)$ is by simply averaging the reward obtained over long horizons of simulated experience, which is possible because, in the typical actor–critic framework, the training data are generated in batches.

[89]: Wan, Naik, et al. (2021), "Learning and planning in average-reward markov decision processes"

[90]: Zhang, Wan, et al. (2021), "Average-reward off-policy policy evaluation with function approximation"

Example 4.5.2

To better understand the average reward formulation, we propose Exercise 10.6 from [27]. Consider the Markov reward process represented in Figure 4.10, which has three states, A, B, and C, and the rewards are +1 upon the arrival in state A and otherwise is zero. What are the differential values for the three states?

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

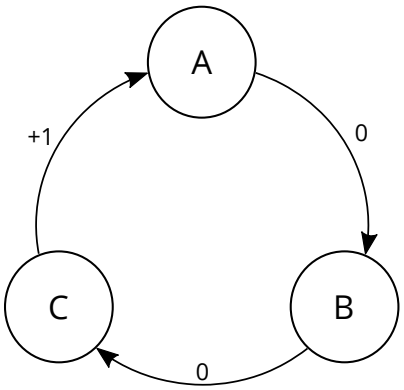


Figure 4.10: Markov reward process.

Answer. The first step consists of computing the average reward $\bar{R}$ that is given by

$$\begin{aligned}\bar{R} &= \lim_{h\to\infty} \frac{1}{h}\mathbb{E}\left[\sum_{t=1}^{h} R_t\right] \\ &= \lim_{h\to\infty} \frac{1+0+0+1+0+0+1+0+0+\ldots}{h} = \frac{1}{3}.\end{aligned}$$

where the instantaneous reward is given by $R_t = \mathbb{1}\{t \mod 3 \equiv 0\}$, and $\mathbb{1}\{\cdot\}$, called the indicator function, assumes the value of one when the condition is met, or zero otherwise. The value function for state A is given by

$$
\begin{aligned}
v(A) &\doteq \mathbb{E}\left[\sum_{t=1}^{\infty}(R_t - \bar{R})\right] \\
&= \lim_{\gamma\to 1}\left[\sum_{t=1}^{\infty}\gamma^{t-1}(R_t - \bar{R})\right] \\
&= \lim_{\gamma\to 1}\left[-\frac{1}{3} - \frac{1}{3}\gamma + \frac{2}{3}\gamma^2 + \sum_{t=4}^{+\infty}\gamma^{t-1}(R_t - \bar{R})\right] \\
&= \lim_{\gamma\to 1}\left[-\frac{1}{3}\left(1+\gamma-2\gamma^2\right) + \gamma^3\underbrace{\sum_{t=1}^{\infty}\gamma^{t-1}(R_t - \bar{R})}_{v(A)}\right] \\
\leftrightarrow v(A) &= \lim_{\gamma\to 1} -\frac{1}{3}\frac{1+\gamma-2\gamma^2}{1-\gamma^3} \\
&= \lim_{\gamma\to 1} -\frac{1}{3}\frac{(\gamma-1)(2\gamma+1)(-1)}{(\gamma-1)(\gamma^2+\gamma+1)(-1)} \\
&= -\frac{1}{3}.
\end{aligned}
$$

Lastly, the value for states B and C can be obtained through bootstrapping from state A with

$$
\begin{aligned}
v(C) &= \left(1 - \frac{1}{3}\right) + v(A) = \frac{1}{3}, \\
v(B) &= \left(0 - \frac{1}{3}\right) + v(C) = 0.
\end{aligned}
$$

4.6 Revisiting the earning–while–learning problem through reinforcement learning

As discussed in Section 4.4, the system seeks to optimize a policy function $\pi(a \mid s)$ that can coordinate pricing decisions across multiple flights. This policy observes a state containing the current inventory states of all flights together with the estimated forecast demand parameters $S_t = (S_t^{[0]}, \ldots, S_t^{[\mathrm{T}-1]}, \boldsymbol{\psi})$, and it seeks to maximize the combined revenue of all flights, i.e, $R_t = \sum_{i=0}^{\mathrm{T}-1} r(S_t^{[i]}, A_t^{[i]})$. We propose to compute such a policy with the actor–critic framework, in which the pseudocode is presented in Algorithm 4. The training environment uses a sample model where each active flight interacts with a simulated demand following the ordinary assumptions (Poisson arrivals,

Algorithm 4: Actor–critic for EWL

Input: The training range $\boldsymbol{\psi}_* \in [\boldsymbol{\psi}_{\min}, \boldsymbol{\psi}_{\max}]$ and the episode horizon $H \gg \mathrm{T}$

Loop forever
- initialize an empty training batch $\mathcal{B}$;
- sample $\boldsymbol{\psi}_*$ uniformly within training range;
- warmup historical booking database according to $d(f; \boldsymbol{\psi}_*)$ and while following the random policy;
- **for** $t = 1, \ldots, H-1$ **do**
  - estimate $\boldsymbol{\psi}$ from historical booking data;
  - set $S_t = (S_t^{[0]}, \ldots, S_t^{[\mathrm{T}-1]}, \boldsymbol{\psi})$;
  - select $A_t = (A_t^{[0]}, \ldots, A_t^{[\mathrm{T}-1]}) \sim \pi(a \mid S_t; \boldsymbol{\theta})$;
  - simulate demand response according to $d(f = A_t; \boldsymbol{\psi}_*)$;
  - observe S_{t+1} and $R_{t+1} = \sum_{i=0}^{\mathrm{T}-1} r(S_t^{[i]}, A_t^{[i]})$;
  - store $S_t, A_t, R_{t+1}, S_{t+1}$ to $\mathcal{B}$;
- recompute the rewards $R'_i = R_i - r(\pi)\ \forall R_i \in \mathcal{B}$, see eq. (4.17);
- update $\boldsymbol{\theta}$ and $\mathbf{w}$ according to eq. (4.15) and eq. (4.16), respectively, using the collected experience $\mathcal{B}$;

negative exponential purchase probability), and the estimation of the demand model parameters is performed as usual (as described in Section 2.2). The only difference to standard methods is how the total return is obtained. At each time step, the RL agent observes the sequence of inventory states and the latest model parameters and selects the fares for each active flight. This continues from any particular starting point without termination, suggesting the use of the average reward.

About the modeling of the policy and the value functions in Algorithm 4, we propose the use of artificial neural networks (ANNs). We recognize that there may be many possible designs for the ANN, and finding the best one is not the focus of our work. The architecture illustrated in Figure 4.11, which uses a long short–term memory (LSTM) [91], the Luong–style attention mechanism [92] following the encoder–decoder architecture [93], is found to deliver stable training. This type of ANN is inspired by research focused on sequence–to–sequence predictions, such as natural language processing. The attention module is also a standard component in the ANNs presented in these fields, and it has been shown to dramatically improve the LSTM performance when input and output sequences grow too large [94]. In essence, the attention module computes "attention" weights that tell for each prediction (or the output of the LSTM) how much emphasis it should give for each input when making decisions. Perhaps the most critical aspect of this design is the LSTM decoder in the actor–network. The number of output units grows linearly to the number of parallel active flights (the agent must compute

[91]: Hochreiter and Schmidhuber (1997), "Long short-term memory"

[92]: Luong, Pham, et al. (2015), "Effective approaches to attention-based neural machine translation"

[93]: Badrinarayanan, Kendall, et al. (2017), "Segnet: A deep convolutional encoder-decoder architecture for image segmentation"

[94]: Bahdanau, Cho, et al. (2014), "Neural machine translation by jointly learning to align and translate"

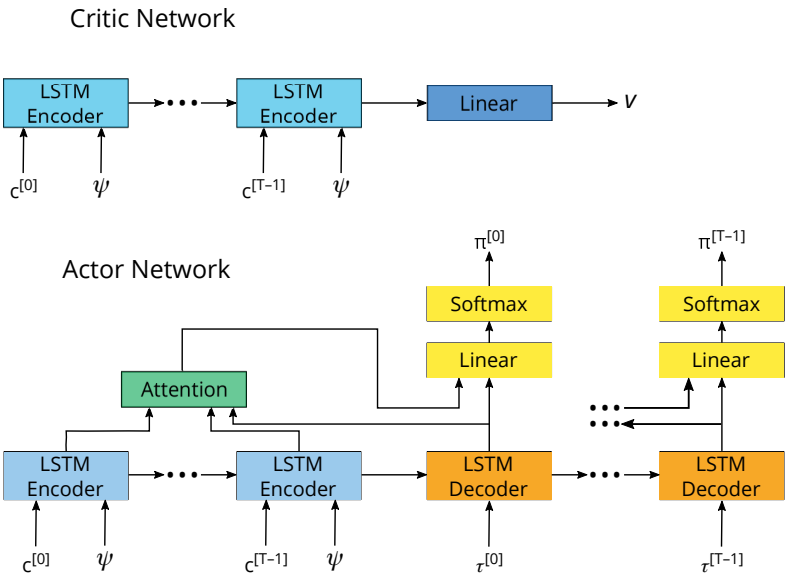


Figure 4.11: The RL agent uses two separate artificial neural networks. The critic network approximates the value function, and the actor network computes the policy function. The long short–term memory (LSTM) encoder takes the remaining capacity inputs $c^{[i]}$. The LSTM decoder takes the remaining time inputs $\tau^{[i]}$ and outputs the probability of selecting each fare in the fare structure $\pi^{[i]}$ for each active flight. Image from [19] under license CC BY 4.0.

for each active flight the probability of selecting each fare in the fare structure) and can quickly cause the ANN to have too many outputs to learn. The LSTM decoder allows the ANN to share the learned parameters between the outputs of each active flight, enabling a more compact and efficient representation.

4.6.1 Evaluating the methods on the single–leg problem

In this section, we evaluate and compare RMS, the heuristic method defined in Section 4.3, and the RL described earlier while performing experiments with a simulated demand. The experiments are separated into two distinct scenarios. In the first one, the RL agent is compared to the heuristic method while estimating a single demand model parameter (price sensitivity ϕ). In this first experiment, we assume the single–leg problem under unconstrained capacity (i.e., infinite inventory capacity). Even though the unconstrained capacity is an unrealistic assumption for most real–world scenarios, this experiment provides a baseline to measure the effectiveness of methods presented earlier when solving the EWL in the single–leg problem. In the second one, the solution obtained by RL is compared with standard RMS (which behaves as certainty equivalent pricing) when capacity is constrained. Furthermore, we assume the system must estimate two model parameters from historical booking data (arrival rate ν and price sensitivity ϕ). This scenario aims at demonstrating the RL agent's ability to optimize prices under capacity constraints (which is not possible with the work developed in Section 4.3) and learn how to solve more complex settings (such as controlling the uncertainty over two model parameters).

Table 4.2: Training hyperparameters. "ep." abbreviates episode. The RLLib implementation of PPO dedicates one CPU to the master thread, which is not used for generating experience.

Experiment	Hyperparameter	Value
unconstrained capacity	train batch size	$102 \frac{\text{eps.}}{\text{CPU}} \times 440 \frac{\text{time steps}}{\text{ep.}} \times 23 \text{ CPUs} = 1{,}032{,}240$
	learning rate (α)	$3 \cdot 10^{-5}$
	entropy coefficient (see [95])	0.005
	value function clip (see [95])	30
	eligibility trace (see [84])	0.15
constrained capacity	train batch size	$175 \frac{\text{eps.}}{\text{CPU}} \times 440 \frac{\text{time steps}}{\text{ep.}} \times 23 \text{ CPUs} = 1{,}771{,}000$
	learning rate (α)	$3 \cdot 10^{-6}$
	entropy coefficient	0.015
	value function clip	30
	eligibility trace	0.1

Throughout the experiments, we consider that each flight has capacity $C = 50$ (representing, for example, the number of seats in the business cabin), the fare structure has ten price points $\mathcal{A} = \{\$50, \$90, \ldots, \$230\}$, the booking horizon has $T = 22$ time steps, the demand follows the exponential model, and the historical database keeps the booking data for the most recent T flights.

For the RL training setup, the proximal policy optimization algorithm (PPO) [95], an actor–critic algorithm, is used because of its simplicity and stability. The training workload is distributed in a cluster with 4 GPUs and 24 CPUs with RLLib [96]. The hyperparameters[5] presented in Table 4.2 were obtained through automated tuning for each experiment independently (some of these parameters are PPO specific, and a formal presentation is beyond our scope). Under such configurations, the agent's performance stabilizes in about 100 training steps, taking 95 hours of computation. The source code is available at [97].

Figure 4.12 illustrates the training and evaluation procedures. For each episode, the parameters of true demand behavior are sampled arbitrarily from a pre-defined range of possibilities (Figure 4.12 (a)). Then, the historical database is warmed up while following the random policy, which is necessary to enable the first estimation of the demand model parameters (Figure 4.12 (b)). Next, the loop of forecasting, optimization, and interactions with the demand is performed for $H = 20 \times T = 440$ time steps (440 time steps are enough for our goals, but this number could be larger; Figure 4.12 (c)). Note that price optimization can be either done with RL, DP, or with the heuristic from Section 4.3. The optimization module can be switched to evaluate a particular method or train the RL agent. The RL agent is trained by iterating

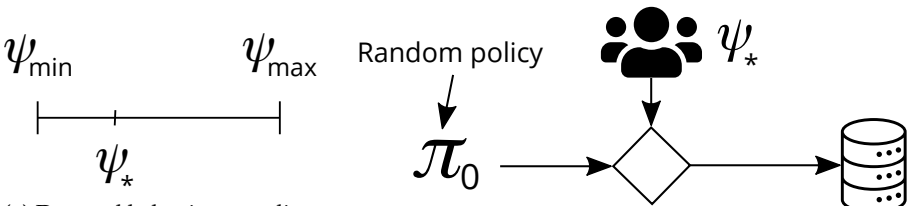
[84]: Schulman, Moritz, et al. (2015), "High-dimensional continuous control using generalized advantage estimation"

[95]: Schulman, Wolski, et al. (2017), "Proximal policy optimization algorithms"

[96]: Liang, Liaw, et al. (2018), "RLlib: Abstractions for distributed reinforcement learning"

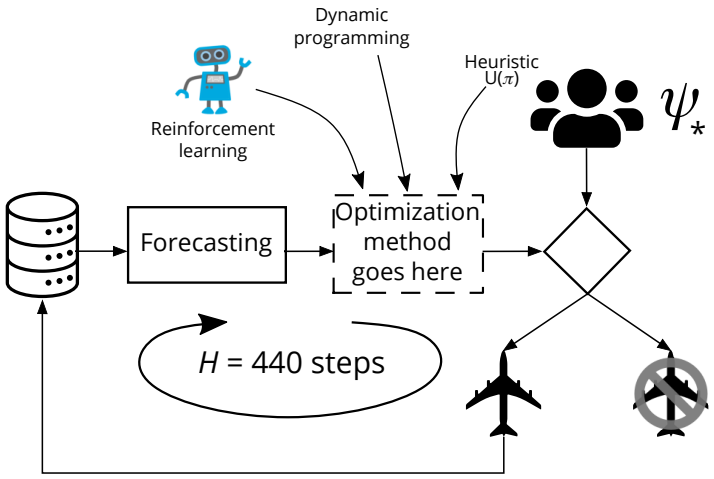
5: A hyperparameter is a parameter that controls the learning process (e.g., the learning rate), in contrast to the values of other parameters that are derived during training (e.g., the connection weights of an ANN).

[97]: Gatti Pinheiro, Defoin-Platel, et al. (2022), *Talos*

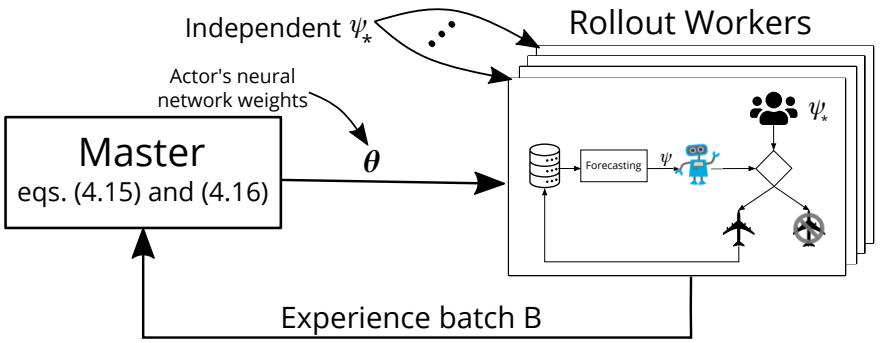


(a) Demand behavior sampling.

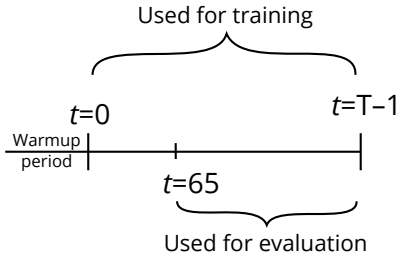
(b) Historical database warmup.



(c) Episode rollout.



(d) Distributed training.



(e) Training vs. evaluation.

Figure 4.12: The anatomy of training and evaluation.

between data collection (i.e., experience batch) obtained by running many independent episodes across several workers and updating the actor and critic's ANN weights with eqs. (4.15) and (4.16). After each update of the ANN weights (learning), the actor's new policy, represented by the weights $\boldsymbol{\theta}$, is updated across all rollout workers simultaneously (to perform policy rollouts, only the actor is required; Figure 4.12 (d)). Concerning the data used for training, only interactions between the system and the demand are kept, including data from the warmup period. However, when performing an evaluation, the first $3\times\mathrm{T} = 66$ time steps are eliminated because we are only interested in the stable regime of each method (optimization methods may be temporarily helped by the high parameter accuracy due to the initialization of the historical database with the random policy; Figure 4.12 (e)). Furthermore, when evaluating the RL agent, the actor and critic's neural network weights are not updated.

Experiment 1: estimating only the price sensitivity under unconstrained capacity

This section considers the case where the system must estimate only the price sensitivity parameter ϕ, and the true arrival rate $\nu_* = 4/22$ is fixed and known at all time steps. The methods are evaluated under a low number of arrivals because estimating the demand price sensitivity is challenging due to the scarcity of booking data, making efficient price experimentation essential for success. Even though the capacity is finite, $C = 50$, given the low arrival rate $\nu_* = 4/22$, it is unlikely that it will ever exhaust even if the base fare (the fare whose purchase probability is one) $f_0 = \$50$ is always selected ($\Pr\{50 \text{ bookings} \mid A_0, \ldots, A_{\mathrm{T}-1} = f_0\} < 10^{-16}$). Therefore, from the optimization perspective, the capacity is practically infinite, allowing us to implement the heuristic method, as presented in Section 4.3.

The system performs model calibration at every time step. Both the heuristic method and RL are evaluated within the interval $F_5^* \in [2.1, 3.8]$. In this interval, the revenue–maximizing fare if the true demand behavior was known covers almost the entirety of the fare structure. In theory, the system does not know anything about the evaluation interval, and any positive value of the price sensitivity could be assumed. However, to avoid extreme evaluations for the price sensitivity parameter, the estimates are limited to the range $F_5 \in [1.5, 4.3]$ (this parameter clipping applies to all methods: RL, RMS, and heuristic). Safety mechanisms like this one are often present in real–world systems. Figure 4.13

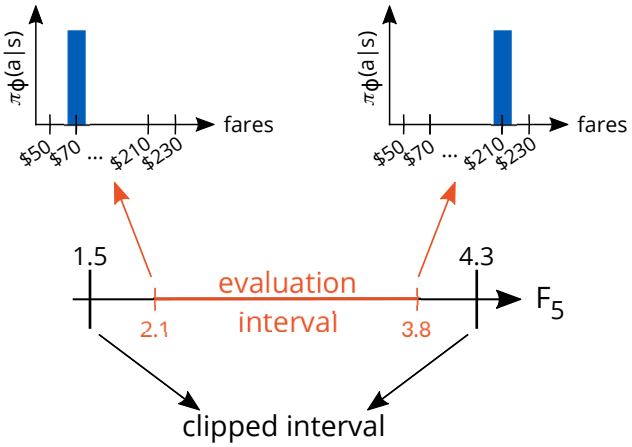


Figure 4.13: An illustration of the clipped and evaluation intervals and its corresponding revenue–maximizing policy. Image from [19] under license CC BY 4.0.

illustrates the relationship of these two intervals to the revenue–maximizing policy when the true demand price sensitivity is known at all times.

We focus the analysis on the revenue performance, represented by the average collected revenue normalized for the revenue–maximizing policy (which knows the true demand price sensitivity at every time step) and the random policy (that selects fares from a uniformly random distribution). Furthermore, we also look at the mean square error (MSE) of the estimated price sensitivity (representing the demand model quality), defined as

$$MSE(\phi) \doteq \frac{1}{n}\sum_{i=1}^{n}(\Phi_i - \phi_*)^2,$$

where Φ_i represents as sample estimation of the price sensitivity parameter.

Calibrating the heuristic method. As we are interested in the system's overall performance within the evaluation interval, the trade–off parameter η is calibrated to maximize the average collected revenue for the system's pricing policy π when sampling the F_5^* parameter uniformly within the evaluation interval. We sample randomly 160 values for the exploration rate in the interval $\eta \in [0, 8000]$, and, for each sampled value, we further sample 2560 values of F_5^* randomly in the evaluation interval. Each episode is simulated as described earlier for 440 time steps. In other words, the performance metric is the average of the revenue obtained by the 2560 episodes of 440 time steps, each having a different F_5^* parameter selected randomly in the evaluation interval or, mathematically, we approximate $\mathbb{E}_{F_5^* \sim U(2.1,3.8)}[r(\pi; F_5^*)]$. Figure 4.14 (left) shows the system's performance as a function of the trade–off parameter. When $\eta = 0$,

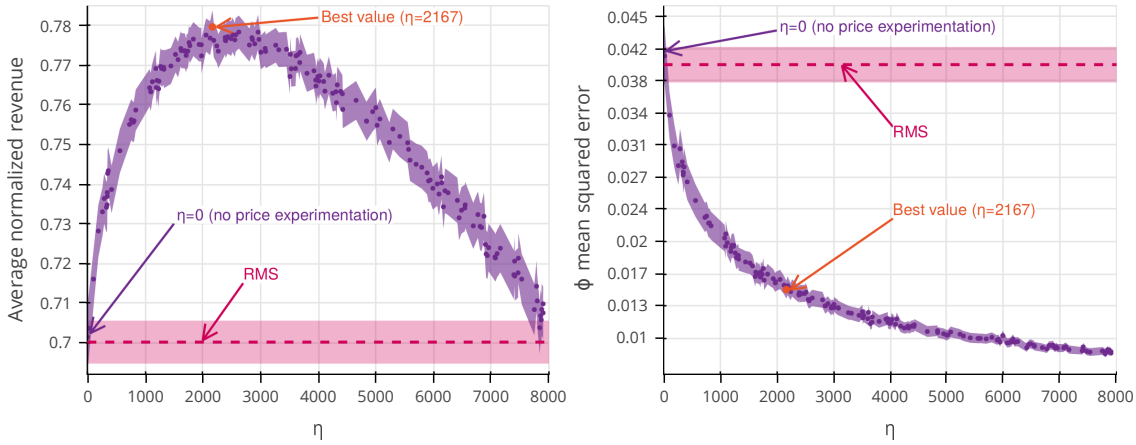


Figure 4.14: Optimizing the trade–off parameter η (99% confidence level). **(Left)** The average revenue is normalized to the revenue–maximizing policy that knows the true demand price sensitivity at every time step and the random policy that selects fares from a uniformly random distribution. We plot a separate estimation of the average behavior for RMS, represented by the dashed horizontal line and its corresponding confidence interval. **(Right)** The mean squared error of the price sensitivity estimation. Image from [75] under license CC BY 4.0.

the system greedily chooses the revenue–maximizing price only, behaving like RMS. As the trade–off parameter increases, the MSE of the price sensitivity estimation decreases Figure 4.14 (right), and the revenue performance increases Figure 4.14 (left), until the point that increasing the trade–off parameter translates into a loss of revenue due to excessive price experimentation. The best revenue performance was obtained for $\eta = 2167$, with a normalized expected revenue of 78.0%, representing an absolute improvement of 7.0% compared to RMS.

Results. The analysis is separated into two distinct parts. In the first one, we compare how each method behaves within the evaluation interval. Then, we perform a more detailed analysis of the final pricing policy obtained for each method.

The average revenue performance and the price sensitivity MSE in Figure 4.15 are computed as a function of the F_5^* for 3565 independent evaluation runs. For the heuristic method, a constant value for the trade–off parameter $\eta = 2197$ is chosen, which is the value that maximizes the expected revenue for the evaluation interval. In Figure 4.15 (left), we observe that RL outperforms RMS and heuristic methods in the entire interval, except for $F_5^* = 3.8$. The reason RL underperforms the heuristic method for $F_5^* = 3.8$ will be explained in the second experiment of this section when the final policies obtained are illustrated. In Figure 4.15 (right), we plot the MSE over the estimation of the price sensitivity. All methods have larger values for price sensitivity MSE for the lower values of F_5^*, which decreases as the F_5^* increases. Such an effect is because the true arrival rate parameter ν_* is known by the system. As the F_5^* increases, so does the optimal fare. The further the selected price is from the base fare f_0, the higher

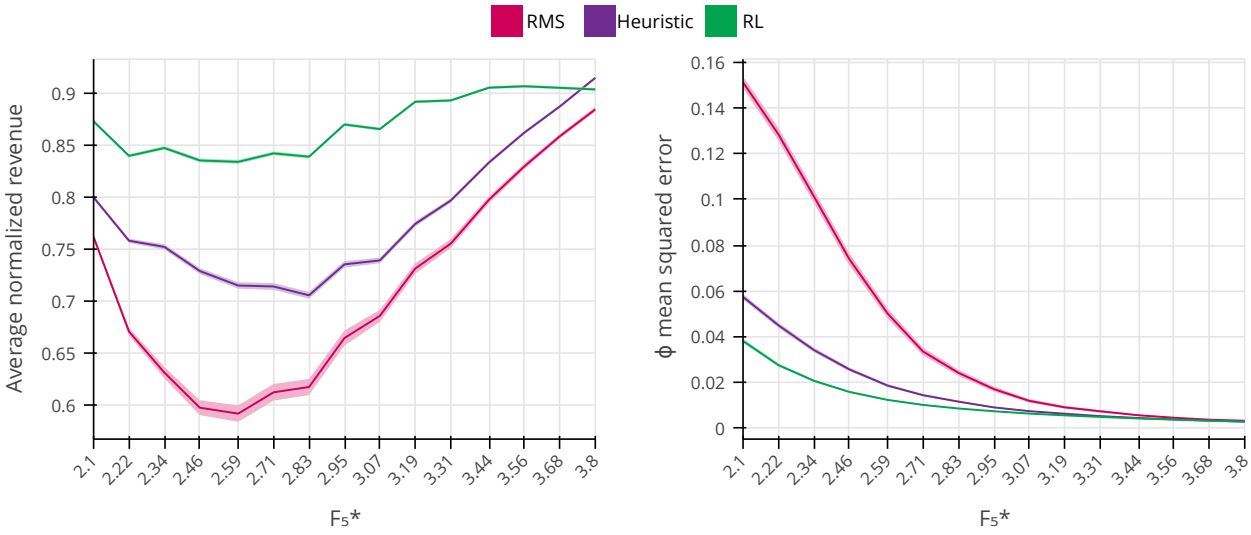
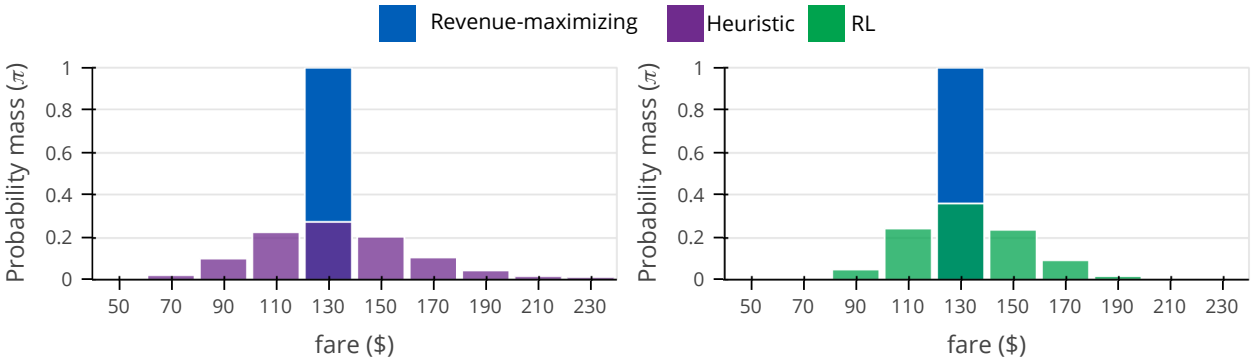


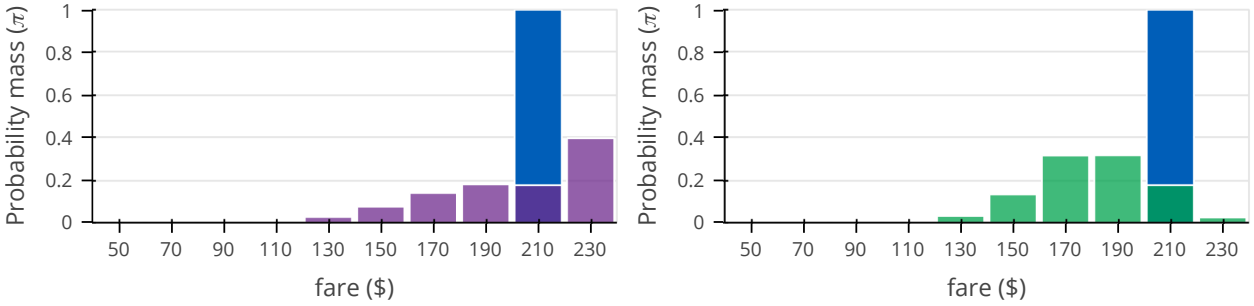
Figure 4.15: The comparison between the average performance of RMS, heuristic method, and RL (99% CL). **(Left)** The system's performance is better the higher the average normalized revenue is. The revenue performance is normalized between the revenue–maximizing (which requires the perfect knowledge of the demand parameters at all time steps) and random policies. **(Right)** The estimation of MSE of the demand model (lower errors are better). Images from [19] under license CC BY 4.0.

the amount of information collected by that price, which can be verified with eq. (4.4). Therefore, the estimation of the price sensitivity is "easier" for higher values of the F_5^* parameter, and, naturally, there is less value in price experimentation. In general, RL and heuristic methods display much less average MSE on the estimation of the customer's price sensitivity than RMS. The effect is more important for lower values of the F_5^* parameter, where the system's policy approaches the base fare f_0, reducing the amount of information obtained by the interaction with the demand. Finally, the RL agent pricing policy displays a better estimation of the demand price sensitivity and generates more revenue on average than the heuristic method and RMS.

Figure 4.16 shows the pricing policy obtained through Monte–Carlo rollouts for both heuristic and RL methods. When $F_5^* = 2.71$ (Figure 4.16 (a)), RL displays a performance advantage over the heuristic method (see Figure 4.15) with lower price sensitivity MSE. The reason is clear when comparing the two policies. RL tends to price closer to the revenue–maximizing fare: It chooses fares \$110, \$130, and \$150 more frequently (83.6%) than the heuristic method (69.9%). RL is slightly more efficient than the heuristic method for the price sensitivity MSE,because RL tends to price the higher fares $f \geq \$130$ more frequently (70.7%) than the heuristic method (65.1%) (in this problem, higher prices contain more information about the demand price sensitivity than the lower prices). When $F_5^* = 3.8$ (Figure 4.16 (b)), RL generates *less* revenue than the heuristic method, and they both display the same price sensitivity MSE. Analyzing



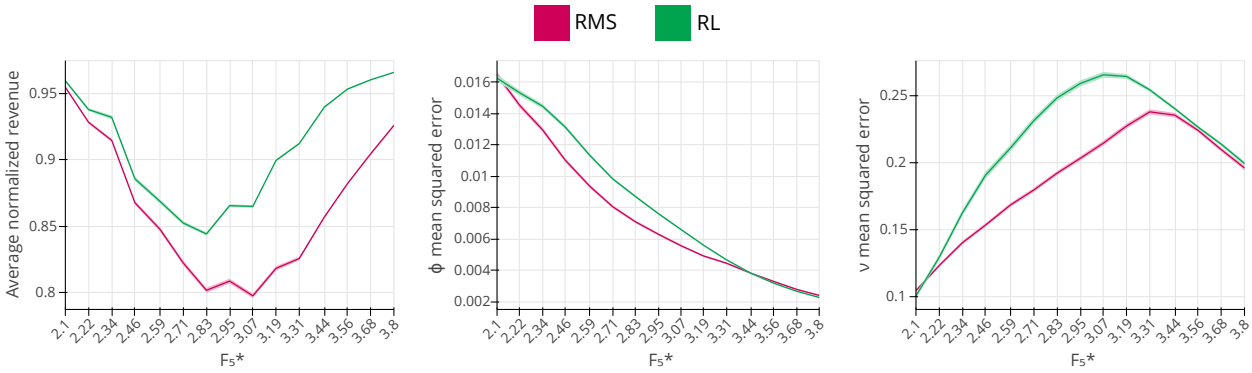
(a) $F_5^* = 2.71$.



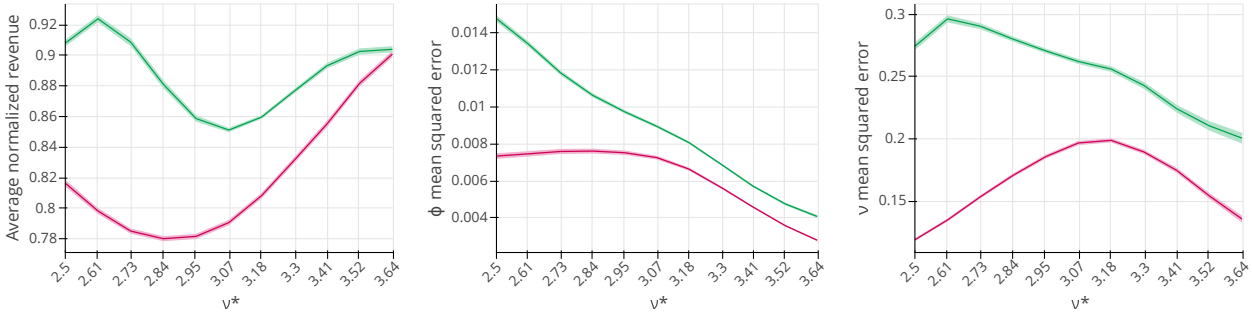
(b) $F_5^* = 3.8$.

Figure 4.16: The comparison between heuristic and RL rollout policies. For convenience, we represent the revenue–maximizing policy. Image from [19] under license CC BY 4.0.

the heuristic and RL policies closely, we see that the largest difference is that the heuristic method selects the highest fare \$230 more often (39.6%) than RL (2.4%). This is surprising because, under the evaluation settings, the highest price is near–optimal for revenue maximization and demand model learning (see eq. (4.4)). The reason fare \$230 is preferred by the heuristic is that, when $F_5^* = 3.8$, the price sensitivity clipping happens at $F_5 = 4.3$ (20.1%; the same is observed with RL), causing the heuristic algorithm to select fare \$230. When the price sensitivity parameter is clipped (and the estimated uncertainty of the price sensitivity is unavailable), the system cannot know whether it was clipped due to large errors or because the true value is close to the limits. Compared to the heuristic (which has access to the estimation of the price sensitivity uncertainty, see eq. (4.2)), RL takes a more conservative approach, distrusting the parameter estimation and pricing fares (\$170, \$190, and \$210) that are often better for demand price sensitivity estimation and that work well for high values of F_5^*.



(a) $\nu_* = 70/22$.



(b) $F_5^* = 2.9$.

Figure 4.17: The comparison between the average performance of RMS, RL and revenue–maximizing policy (99% CL). Image from [19] under license CC BY 4.0.

Experiment 2: estimating the price sensitivity and arrival rate under constrained capacity

For our second experimental setting, we consider that the system must estimate the demand model parameters (arrival rate ν and price sensitivity ϕ) from historical bookings. The true arrival rate can assume any value in the range $\nu_* \in [55/22, 80/22]$, making the flights' capacity $C = 50$ finite in practice. As in the previous experiment, the true price sensitivity can adopt any value in the range $F_5^* \in [2.1, 3.8]$, and the RL agent is trained over the clipped interval wider than the evaluation interval ($\nu \in [50/22, 85/22]$ and $F_5 \in [1.5, 4.3]$).

Results. As in the previous section, we first analyze how the RMS and the RL agent perform in the evaluation interval concerning revenue performance and model quality. Then, we compare how the two policies differ for a specific value of true demand behavior parameters.

In Figure 4.17(a), we show how the average normalized revenue and the MSE for the demand model parameters respond to different values of true price sensitivity within the evaluation interval. In the left chart, we see that RL outperforms RMS in the entire interval, with a larger advantage over higher values

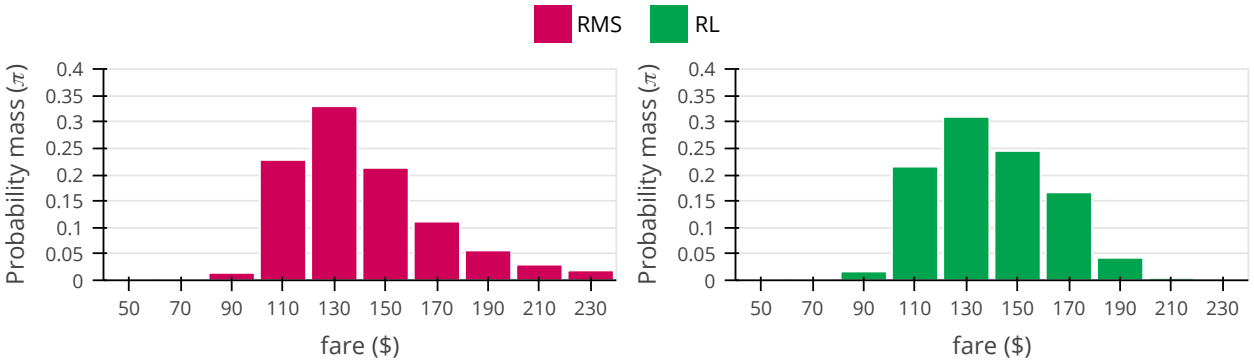


Figure 4.18: The comparison between RMS (left) and RL (right) rollout policies for $\nu_* = 70/22$, $F_5^* = 2.5$. The RL policy performs better (89.9%) than RMS (80.6%) with respect to the revenue performance, but worse MSE for both demand model parameters. Image from [19] under license CC BY 4.0.

of the F_5^* parameter. In the center and right charts, we see that the MSEs of both parameters are close to each other, but RL consistently demonstrates a worse MSE performance. The same behavior is also present when the true customer price sensitivity is fixed, and only the arrival rate varies, as in Figure 4.17(b). This may seem paradoxical: How can RL have a better revenue performance while having a worse estimation of the demand model parameters? Unfortunately, we do not completely understand the RL strategy to precisely answer this question, and we can only speculate about its strategy. As it is observed in the following analysis, instead of reducing model uncertainty, the agent can learn to select a "safe price" unlikely to lose much revenue, especially when the estimated demand model parameters are too extreme (i.e., close to the clipping limits).

In the second analysis, Figure 4.18 displays the RMS and the RL agent pricing policies when the true arrival rate and price sensitivity are fixed at $\nu_* = 70/22$ and $F_5^* = 2.5$. We choose this point because the flights' average load factor is 70%, which brings a natural price variability and a high amount of booking data, being a favorable situation for estimating the demand model parameters (the historical data contains, on average, 1078 bookings). Given these two properties, obtaining a good estimation of the demand model parameters is relatively easy for RMS, which puts the value of price experimentation in question. RMS achieves a normalized revenue performance of 80.6%, with an MSE for the price sensitivity and the arrival rate of 0.007 and 0.20, respectively. The agent is better for revenue maximization, with a normalized revenue performance of 89.9%, but has a worse model quality with an MSE for the price sensitivity and the arrival rate of 0.008 and 0.26, respectively. When comparing the two policies, the major difference is that the agent prefers to price center fares \$130, \$150, and \$170 more frequently (72.1%) than RMS (65.3%). This strategy harms the overall model quality

Table 4.3: Summary of experimental results.

Experiment	Method	Average normalized revenue	Price sensitivity MSE	Arrival rate MSE
Unconstrained capacity	RMS (CEP)	0.702 ± 0.007	0.0401 ± 0.0026	—
	heuristic	0.783 ± 0.003	0.0148 ± 0.0008	—
	RL	**0.868 ± 0.002**	**0.0109 ± 0.0005**	—
Constrained capacity	RMS	0.862 ± 0.003	**0.0072 ± 0.0002**	**0.162 ± 0.003**
	RL	**0.912 ± 0.002**	0.0090 ± 0.0003	0.219 ± 0.004

because it decreases the price variability of the historical data. However, it turns out that the center fares are better for revenue maximization than higher fares such as $210 and $230 because they are less extreme given the true demand behavior. This aligns with our intuition that the RL agent prefers "safer prices" when "price experimentation" is less suitable.

Table 4.3 presents a summary of the performance of each method for the two investigated scenarios. The results in this table correspond to the average behavior of 3565 episodes randomly sampled in the evaluation range for each method.

4.6.2 Ablation studies

To gain more insight into the contribution to the system's final performance of the several ideas presented in the past sections, we remove the various features one by one when performing training. Then, we measure how the final performance is impacted by each change. Three essential changes are analyzed. First, we study how the estimated demand model parameters influence the agent's final policy. Then, we analyze the encoder–decoder architecture by comparing it with another simpler architecture specifically designed to suppress some of its properties. Finally, we investigate the importance of the average reward by comparing it to the discounted formulation.

State representation

We mask the estimated demand model parameters from the RL agent during training to evaluate how much each parameter impacts the system's performance. The masking procedure is straightforward: The parameter value is forced to zero before inputting it into the agent. Figure 4.19 compares earning and learning of the final policy obtained by the agent when only the arrival rate ν parameter is hidden from the agent, or only the

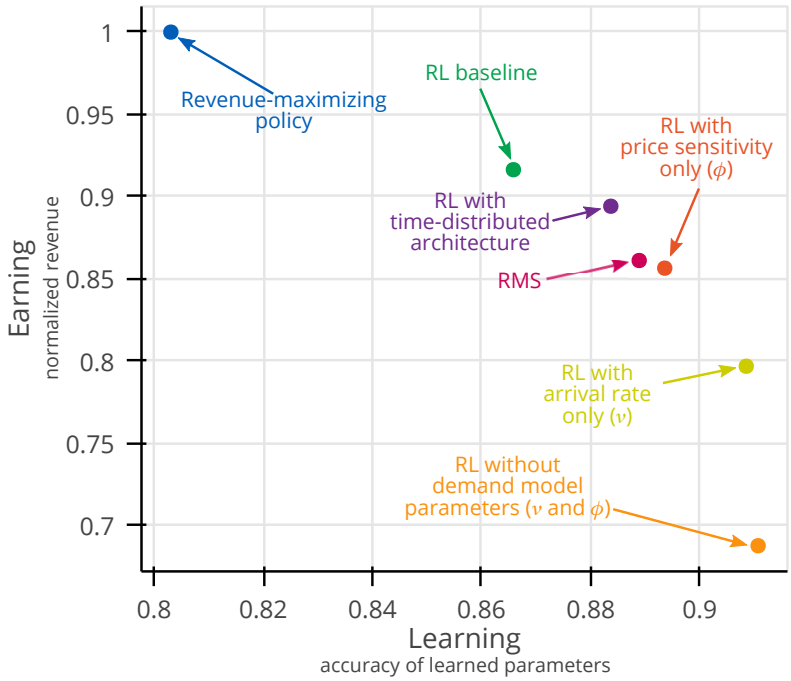


Figure 4.19: Ablation studies.

price sensitivity parameter ϕ is hidden, or when both parameters $\psi = (\nu, \phi)$ are hidden simultaneously. This figure is computed using the forecasting module to estimate both arrival rate and price sensitivity, and the true demand behavior parameters are sampled uniformly in the evaluation interval. For reference, we provide the revenue–maximizing policy (which knows the true demand behavior parameters at all time steps), RMS, and finally, the RL agent trained without any masking (that we refer to as baseline).

As expected, baseline produces more normalized average revenue (0.916 ± 0.002; 99% confidence level) than all three versions where masking is applied. The presence of the estimated demand model parameters is decisive: Without it, the agent's revenue performance drops dramatically (0.687 ± 0.010). The estimation of the price sensitivity (0.856 ± 0.002) is more important to the agent than the estimation of the arrival rate (0.797 ± 0.003), aligning with expert intuition [5, 22] that it is more important, for revenue maximization, to correctly choose the lowest opening fare (which is associated with the price sensitivity parameter, as discussed in recall Section 2.3) than controlling capacity constraints (which is related to the arrival rate parameter).

With respect to learning (horizontal axis), there is a general trend that the estimated demand model parameters have lower accuracy as more information about the true state of the environment is available. This behavior can be linked to the degree of "determinism" of the final learned policy. How much any policy is deterministic can be measured with its entropy that is given

[5]: Fiig, Härdling, et al. (2014), "Demand forecasting and measuring forecast accuracy in general fare structures"
[22]: Fiig, Weatherford, et al. (2019), "Can demand forecast accuracy be linked to airline revenue?"

by

$$\mathrm{H}(\pi) \doteq -\mathbb{E}_{\pi}\left[\sum_{a} \pi(a \mid S_t)\, \ln(a \mid S_t)\right].$$

For example, assuming a fare structure with $n = 10$ price points and a horizon with $\mathrm{T} = 22$ time steps, when training starts and the agent follows the near–random policy, i.e., $\pi(a_j^{[i]} \mid S_t) \approx 0.1$, we can write

$$\begin{aligned}\mathrm{H} &= -\sum_{i=0}^{\mathrm{T}-1}\sum_{j=0}^{n-1} \pi(a_j^{[i]} \mid S_t)\, \ln \pi(a_j^{[i]} \mid S_t) \\ &= -220 \cdot 0.1 \cdot \ln(0.1) \\ &\approx 51.\end{aligned}$$

The learned policies when both estimated parameters are hidden, or only the arrival rate is provided to the agent have higher entropy ($\mathrm{H} \approx 23$) than the learned policy when only the price sensitivity is provided ($\mathrm{H} \approx 19$). Furthermore, the lowest entropy is obtained by the baseline policy ($\mathrm{H} \approx 16$), which is still significantly higher than the fully deterministic revenue–maximizing policy ($\mathrm{H} = 0$). In other words, the agent's policy is more deterministic as more information is available. As the agent's policy becomes more deterministic, this determinism degrades the quality of the estimated demand model accuracy. In a sense, the agent starts following the random policy and tries to reduce the policy entropy as much as possible, which generally increases of the noise in the estimated demand model parameters. Gradually it finds a balance that depends on how trustworthy information is available in the observations.

In summary, without both estimated demand model parameters, the agent is unable to perform better than RMS (0.861 ± 0.003), meaning that these parameters are required for success.

Artificial neural network architecture

The baseline uses the encoder–decoder architecture illustrated in Figure 4.11. In principle, this architecture allows the agent to observe the remaining capacities of the active flights altogether while coordinating the pricing strategy across these flights. To measure how influential this architecture is to revenue maximization, we propose the simpler architecture illustrated in Figure

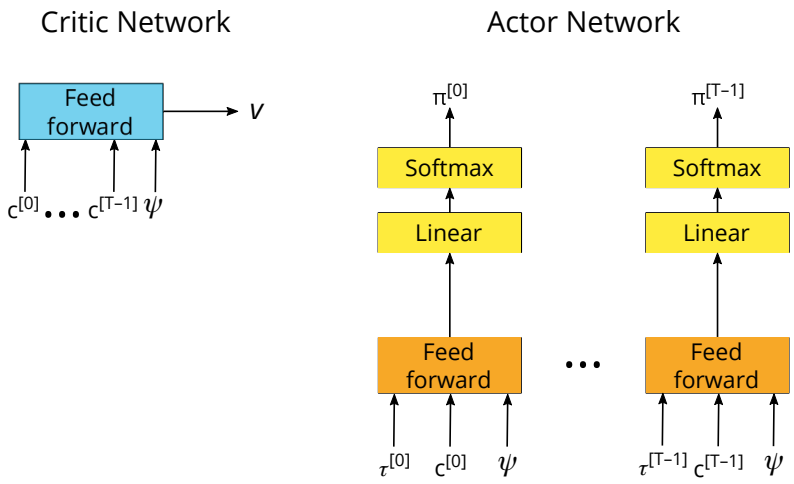


Figure 4.20: Time–distributed architecture.

4.20 that uses a time–distributed feedforward network that optimizes decisions locally. The outputs of this ANN are the probabilities of selecting prices while considering the remaining capacity and time of each active flight independently of each other. The critic architecture is also changed to use a feedforward ANN as well. The critic architecture is also changed to use a feedforward ANN as well. In Figure 4.19, we observe that the time distributed architecture achieves a lower final average revenue performance (0.894 ± 0.002) than baseline (0.916 ± 0.002) but better than RMS (0.861 ± 0.003).

The quality of the critic network predictions can be evaluated with the explained variance of the value function defined as

$$\mathrm{EV}(\pi) \doteq 1 - \frac{\mathrm{Var}_\pi[G_t - v(S_t; \mathbf{w})]}{\mathrm{Var}_\pi[G_t]},$$

where $\mathrm{Var}_\pi[\cdot]$ denotes the variance of a random variable while following policy π. Roughly, the critic predictions are better as the explained variance is closer to one. The critic network performances can be compared using this metric for both time distributed and encoder–decoder architectures. Indeed, the critic from these two architectures display similar final performances ($\mathrm{EV} \approx 0.933$). Thus, the observed loss of revenue performance is more likely due to the actor than the critic network, suggesting that encoding the active flights' capacities and communicating local decisions are valuable features for the actor network.

Computing the return

The continuing nature of the EWL problem suggests using the average reward rather than the more classical discounted return

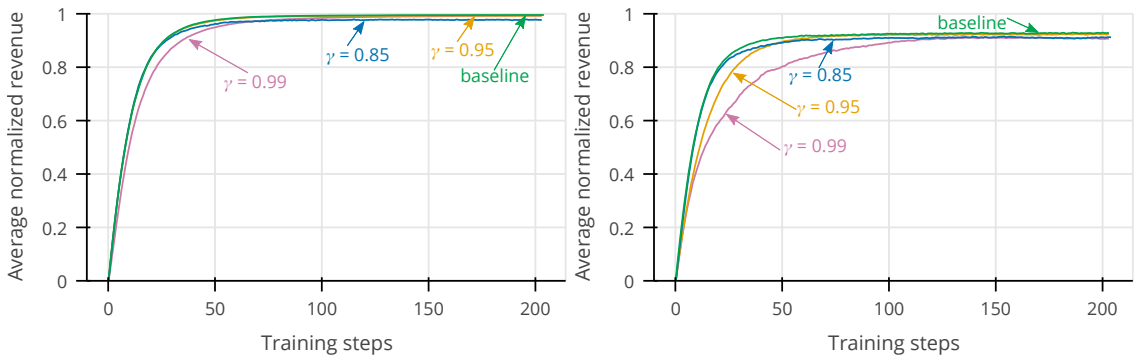


Figure 4.21: Discounted return vs. average reward. **(Left)** Training progress when the true demand model parameters ψ_* are known by the agent at all time steps. **(Right)** Training progress when the true demand model parameters are unknown by the agent and must be estimated from historical booking data.

formulation. To evaluate the importance of such a choice, the average reward computation in Algorithm 4 (line 12) can be replaced by the discounting formulation, i.e., $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. Then, we can compare the properties of these two algorithms. Recall that the discount rate must be less than one ($\gamma < 1$) when working with continuing tasks because the value function may diverge otherwise ($v(s) \rightarrow \infty$). First, we compare the performance of these two algorithms in the case that the true demand behavior parameters ψ_* are known at all time steps. This is different from the EWL problem where the demand model parameters must be estimated from historical bookings, but it allows the verification that all methods converge to the same answer as the discount rate approaches one. Figure 4.21 (left) compares the baseline performance to various values of the discount rate during training. As one would expect, the agent is more farsighted as the discount rate increases, thus converging to a better solution. When $\gamma = 0.99$, the agent reaches a similar average revenue performance (0.993 ± 0.001) as when $\gamma = 0.95$ (0.993 ± 0.001), but the worst performance is when $\gamma = 0.85$ (0.978 ± 0.001). Nonetheless, baseline, which uses the average reward, is the best–obtained solution (0.996 ± 0.001).

When comparing the discounting formulation to the average reward in the EWL problem setting, see Figure 4.21 (right), the best–obtained solution is, again, the average reward (0.928 ± 0.002). However, in contrast to the previous scenario where the true demand behavior parameters are perfectly known, the best–obtained solution for the discounted formulation is when $\gamma = 0.95$ (0.922 ± 0.002), followed by $\gamma = 0.85$ (0.910 ± 0.002) and $\gamma = 0.99$ (0.904 ± 0.002). Furthermore, the RL agent takes more training steps to converge as the value of the discount rate increases. Even though the final performance difference between the average reward and discounted return has little practical meaning, the average reward formulation leads to faster

convergence and better results while eliminating the need to tune the discount rate hyperparameter. This finding aligns with other studies in the RL literature that question the worthiness of the discounted formulation for continuing tasks when using significant function approximation [27] or when the true state of the environment is not fully observable [98].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

[98]: Singh, Jaakkola, et al. (1994), "Learning without state-estimation in partially observable Markovian decision processes"

To develop some intuition on why the average reward formulation provides the best result, consider that the states, represented by feature vectors, do little to distinguish between each other, which is the case when the true underlying state of the environment cannot be directly observed (such as in the EWL problem). There are no start or terminal states, only an infinite sequence of actions and rewards without a beginning or an end, and the performance must be assessed from those. One way to measure performance is by averaging the rewards over a long interval, which is the idea behind the average reward formulation. Alternatively, the discounted returns could be computed, but, as they can vary according to a random variable that is not fully observable, it would be needed to average these returns across long intervals. If done this way, the average of discounted returns is found to be proportional to the average reward $r(\pi)/(1-\gamma)$ (see "The Futility of Discounting in Continuing Problems" [27]), which is essentially the average reward.

4.6.3 Discussion on reinforcement learning

Perhaps one of keys to RL success was the simplification of the multi–objective optimization aspect of the EWL problem into a single optimization dimension: Revenue maximization. Instead of describing the EWL problem as an explicit requirement of balancing revenue maximization and model learning, common to heuristic methods [66, 67], the RL agent is asked to maximize revenue only and has the flexibility to trade–off between earning and learning as it finds best suitable. Figure 4.22 shows how RL trades earning and learning throughout its training. Before training begins, RL starts following a near–random policy at the bottom right. As training progresses, RL policy improves its revenue performance (earning) at the expense of the demand behavior parameter's accuracy (learning). At some point in training, RL displays the same revenue performance as RMS while presenting a better estimation of the demand model parameters. As training continues, RL can further improve revenue by sacrificing the demand model quality, ultimately converging towards the equilibrium point represented by the dot at the end of its trajectory, where it cannot further improve revenue.

[66]: Boer and Zwart (2014), "Simultaneously learning and optimizing using controlled variance pricing"

[67]: Elreedy, Atiya, et al. (2021), "Novel pricing strategies for revenue maximization and demand learning using an exploration-exploitation framework"

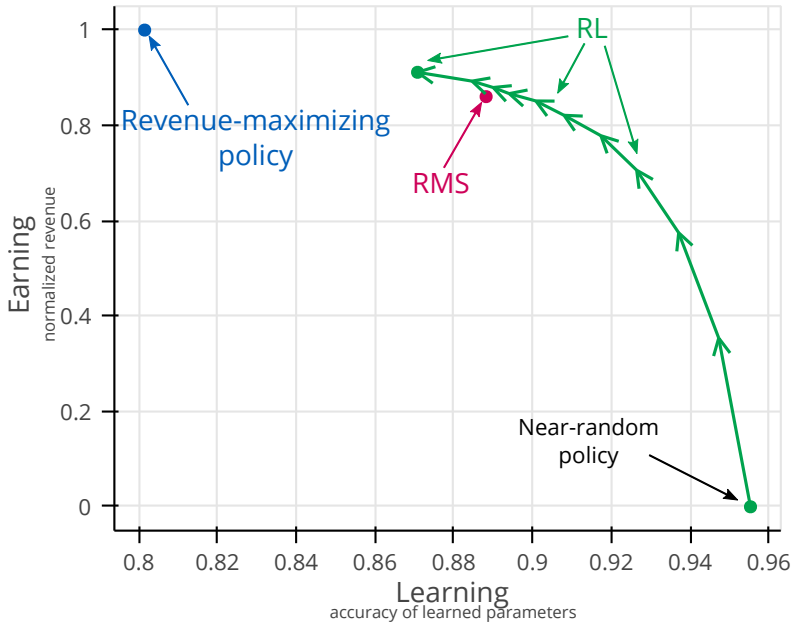


Figure 4.22: RL trajectory in the earning–learning space during the training phase. The earning axis represents the average normalized revenue for the evaluation interval, while the learning axis represents the average estimated demand parameters accuracy. The revenue–maximizing policy is obtained by providing the true demand behavior parameters to the optimizer while keeping the estimation of parameters of the demand model as usual. The balance found by the revenue–maximizing policy is unstable, thus not being an answer to the problem. Image from [19] under license CC BY 4.0.

Since controlling model uncertainty is required to succeed in the task, the agent has the mission of identifying *when* and *how much* price experimentation should be done (if ever done). Such an approach has strong connections to the "reward is enough" hypothesis [99], which suggests that associate abilities of intelligence (e.g., social intelligence, perception, knowledge representation, planning, etc.) can be understood as subserving the maximization of reward (i.e., the measure of success in the task). In our case, we claim that controlling the quality of the estimated demand model is the associated ability required to achieve reward (or revenue) maximization. RL is trusted to find the solution to all the complexities of the EWL problem from nothing more than the standard RMS raw inputs without any human guidance.

[99]: Silver, Singh, et al. (2021), "Reward is enough"

Lastly, RL also displays some practical benefits compared to heuristic methods, such as the one developed in Section 4.3. The RL agent does not need any external indication of the level of trust in the estimated demand model parameters (represented by the estimation of demand model uncertainty σ_ψ in heuristic methods). The advantage of not requiring a measure of uncertainty is that, for realistic–sized demand models having many parameters to be calibrated, computing these quantities and balancing their importance in the optimization heuristic may not be a trivial task. Furthermore, training the RL agent imposes other constraints, such as computation requirements and fine–tuning of the algorithm's hyperparameters, that may be easier to address and less time–consuming than heuristic–based methods.

4.7 Summary

Optimizing prices under an unknown demand behavior is also referred to as the earning–while–learning (EWL) problem, and, in recent years, this problem has attracted attention from academia and industry, particularly since the COVID-19 pandemic, which significantly impacted the global economy and the airline industry. We believe recovery requires effective price experimentation to learn the new and evolving demand price sensitivity.

The system's current pricing policy impacts the booking data it collects and the quality of future forecasts, raising the question of how to perform efficient price experimentation to improve the quality of future forecasts and increase long–term revenue.

The EWL problem has been investigated earlier, and proposed solutions rely on heuristic optimization. Unfortunately, many of these heuristics have no obvious extension to the single–leg problem, which requires methods to manage capacity constraints and price optimization of multiple active flights. One of the most promising approaches [67] can be efficiently adapted to the single–leg problem under the assumption of unconstrained capacity, leading to a new objective that allows the user to manually calibrate the system's emphasis to price experimentation and revenue maximization when optimizing prices.

[67]: Elreedy, Atiya, et al. (2021), "Novel pricing strategies for revenue maximization and demand learning using an exploration-exploitation framework"

The EWL problem can be formulated to the single–leg as the price optimization of several parallel flights at once (rather than each flight individually), given the observed capacity constraints and the most recent observed estimated parameters. This change in the problem definition makes it intractable with classical methods due to the large size of the state–action space and the difficulties of modeling the future distributions of the estimated demand model parameters. In contrast, reinforcement learning (RL) methods can be adapted to this new formulation. The class of actor–critic methods are the most promising ones because they use policy parameterization, which allows them to optimize across large action spaces.

Policy gradient methods approximate a stochastic parameterized policy function through stochastic gradient ascent to maximize long–term reward. In addition to policy parameterization (actor), actor–critic methods also learn the weights of a parameterized value function (critic). The critic is not used for decision–making but only for learning the policy, and it allows reducing the variance of the observed returns, easing the learning procedure.

Beyond the large state–action spaces, the EWL problem also presents a continuing nature, i.e., balancing between earning

and learning is a task that never finishes. For tasks having this nature, there is an alternative formulation of goals in Markov decision processes (MDPs) referred to as the average reward. As long as the MDP has the property of ergodicity, the value functions can be rewritten for the average reward (in contrast to the discounted formulation presented in previous chapters), which allows us to plug this formulation straightforwardly in the actor–critic framework.

We choose to model the agent with artificial neural networks (ANNs) following the long short–term memory encoder–decoder architecture with attention. This model limits the number of outputs the ANN needs to learn by sharing the weights for each component of the output vector. During the ablation studies, it was observed that the most important aspect of our algorithm is having the demand behavior estimated parameters in the representation of the state, followed by the encoder–decoder architecture, and, finally, the average reward formulation.

When evaluating RMS, heuristic method, and RL in a capacity unconstrained scenario, RL generally provides a better revenue output than the other methods while reducing errors in the estimation of the demand model parameters. When comparing RMS and RL in a constrained capacity scenario, RL is also better for revenue maximization but worse in the estimation of the demand model parameters. One possible explanation is that RL learned to price safely rather than reduce errors of the estimated demand model parameters. This strategy is fairly unexplored by past research on the EWL problem.

This chapter's results highlight the RL capability of finding better solutions than human–designed heuristics, presenting itself as an alternative path to improve RMSs. The next chapter discusses how to adapt the method present here to other issues of RM and its consequences.

Beyond Earning while Learning

5

Reinforcement learning may have the power to transform the field of revenue management. This chapter presents possible ways to adapt the method previously described to address other long–standing issues in airline RM and how researchers and practitioners can interact with such RL–based systems.

5.1 In depth view of partial observability

In the earning–while–learning problem presented in Chapter 4, the true state of the environment is not directly observable because the parameters regulating the demand behavior are unknown (and estimated from past bookings). When the system does not have access to the complete state of the environment, the MDP is said to be partially observable. Optimization under *partially observable MDPs* (POMDPs) is a research topic in the field of RL, and many studies have been conducted throughout the years [100–102]. Even though the literature around this topic is large and outside our scope, we present some key concepts essential for further understanding our work and how to improve it.

[100]: Jaeger (1998), "A short introduction to observable operator models of stochastic processes"
[101]: Thon (2018), "Spectral learning of dequential systems"
[102]: Monahan (1982), "State of the art – a survey of partially observable Markov decision processes: theory, models, and algorithms"

The RL algorithms presented in Chapters 3 and 4 rely heavily on the concept of a state, i.e., the returns of the policy and value functions are approximations of observed states. Indeed, function parametrization includes important aspects of partial observability since the parameters can be adjusted independently of state variables that are not directly observable. However, some issues need more explicit treatment of partial observability for further investigation.

Without loss of generality, let's assume that the reward signal is computed directly from observations or that the reward is one of its components. The interactions with the environment have no complete states or rewards but rather an alternating sequence of observations $O_t \in \mathcal{O}$ and actions $A_t \in \mathcal{A}$. This sequence at time t, called the *history*, is denoted by

$$H_t \doteq O_0, A_0, O_1, A_1, O_2, A_2, \ldots, A_{t-1}, O_t.$$

This sequence represents everything the agent knows about the environment without looking outside the stream of data and could be used, in principle, for accessing the unobservable true state of the environment. For the history to be useful, the system needs to transform this history into an *internal state* S_t that is provided to the policy function $\pi(S_t)$. The function that maps the history into the system's internal state is denoted by $S_t = h(H_t)$. For example, h could be the identity function, such that $h(H_t) = H_t$. Even though simple, the identity function is not a practical choice because the state would never recur and the agent would never experience the same state twice. Therefore, the function h needs to represent the history into a compact summary useful for predicting the future. Furthermore, there are also computational aspects to be considered. The amount of data captured data grows with t, and, at a certain point, it becomes large and unwieldy. For these reasons, rather than a function taking in whole histories, the system needs a function having the same effect by doing so incrementally, i.e.,

$$S_{t+1} = u(S_t, A_t, O_{t+1}), \forall t \geq 0,$$

with the first S_0 initialized arbitrarily. The above recursive function is known as the *state–update function* [27].

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

These functions, h and u, are said to have the *Markov property* if and only if two arbitrary histories H and H′ that are mapped to the same state (e.g., $h(\mathrm{H}) = h(\mathrm{H}')$) present the same probabilities for their next observation, i.e.,

$$h(\mathrm{H}) = h(\mathrm{H}') \Rightarrow \Pr\{O_{t+1} = o \mid H_t = \mathrm{H}, A_t = a\} = \Pr\{O_{t+1} = o \mid H_t = \mathrm{H}', A_t = a\}.$$

The ability of the system to predict anything depends on how strong this Markov property is. When the environment is fully observable, only the last transition of the history (i.e., $H_{t-1:t} = O_{t-1}, A_{t-1}, O_t$) is enough to identify the complete state of the environment. However, if the Markov property is only approximately satisfied, the long–term predictions can degrade dramatically, impacting the system's performance [27]. Hence, one effective way to learn when the Markov property is not strong consists of approximating a stochastic policy while maximizing the average reward [98], as presented in the previous chapter.

[98]: Singh, Jaakkola, et al. (1994), "Learning without state-estimation in partially observable Markovian decision processes"

In the case of airline RM, the system must rely on a noisy approximation of the true state obtained from past observations. Specifically, the historical database keeps track of the latest

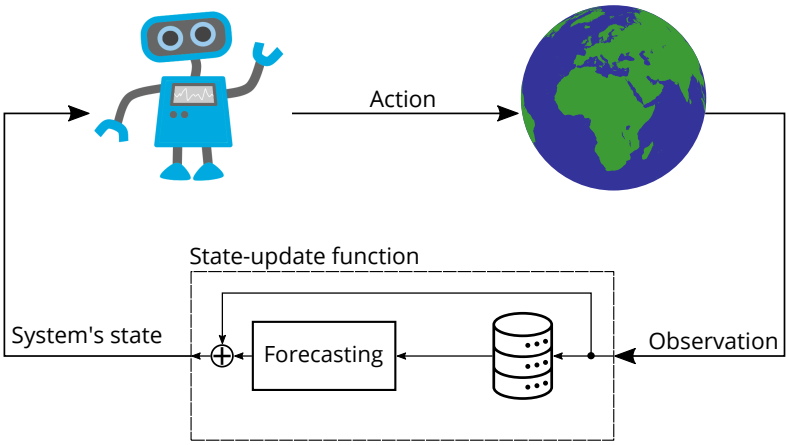


Figure 5.1: An illustration of the state–update function for Algorithm 4.

T observations and actions and transforms this history (data stream) into the estimated parameters of the demand model (compact summary). These estimated parameters depend on the true state of the environment that is not directly observable. In Algorithm 4, concatenating the latest observation with the estimated demand behavior parameters, as illustrated in Figure 5.1, acts as the state–update function, and its output behaves as the system's internal state. Taking such a perspective, one could wonder which other ways we could build this state-update function. The following sections explore how this function can be adapted to address other issues in airline RM while keeping the same algorithmic structure for training presented in Chapter 4.

5.2 Revisiting non–stationarity and competition

As discussed in Chapter 1, alongside the earning–while–learning problem, there are at least three other open issues in airline RM that can be addressed with RL: pricing under competition, self–competition (or cannibalization), and non–stationarity. This section discusses how the RL architecture proposed in Chapter 4 could be adapted to address these issues.

5.2.1 Non–stationarity

Airlines use *past* booking data to predict *future* demand when optimizing prices. However, the past may not represent the future because of *abrupt* changes in market conditions, such as pandemics, economic crises [103], and *smooth* changes, such as economic growth [18]. In other words, real–world demand is not stationary as assumed in Chapter 4 but changes over time.

[103]: Gatti Pinheiro, Fiig, et al. (2022), "Demand change detection in airline revenue management"

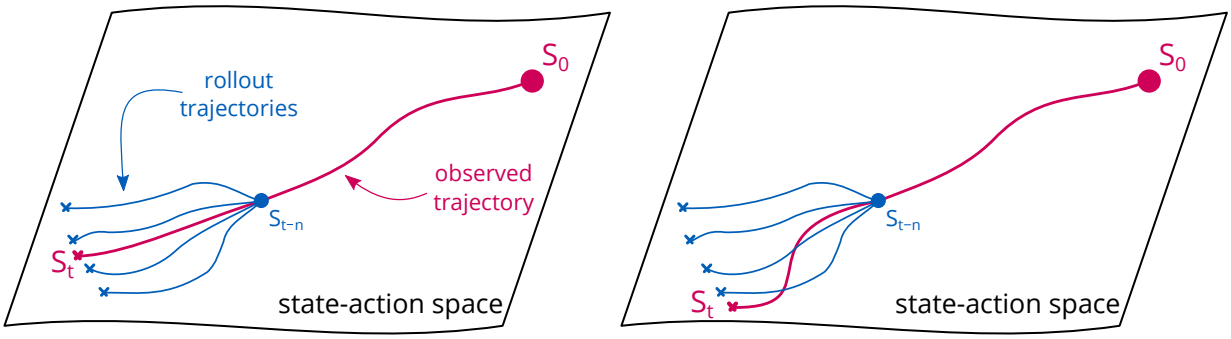
[18]: Keskin and Zeevi (2017), "Chasing demand: Learning and earning in a changing environment"

For this reason, to adapt Algorithm 4 to non–stationary demand behavior, the first and most straightforward change consists in making the true demand behavior a function of time, i.e., $\boldsymbol{\psi}_*(t)$.

But, which function $\boldsymbol{\psi}_*(t)$ should be chosen? If there is reliable information on how the demand behavior will change in the future, the functional form of such a change can be specified in the true demand behavior parameters used for training. However, in principle, future changes in the true demand behavior are unknown, and the agent may perform poorly if the real–world demand behavior changes differently than specified in training. Instead, the system needs to identify different regimes and react accordingly.

Perhaps one way to design $\boldsymbol{\psi}_*(t)$ consists in looking into how demand behavior changed in the past and reproducing such changes during training. Even though this might be a piece of a solution, there are no guarantees that past changes in demand behavior will represent future changes, e.g., the COVID-19 pandemic was unprecedented in the airline industry. Alternatively, one could generate many $\boldsymbol{\psi}_*(t)$ functions and pick among them arbitrarily during training. For example, $\boldsymbol{\psi}_*(t)$ could be a function increasing/decreasing linearly in time or any other functional form that seems relevant. The agent should be better prepared to react to unseen changes when the specified family/range of $\boldsymbol{\psi}_*(t)$ during training is large in variety and realistic in functional form. Put differently, the future is not foreseeable, but the agent can be trained in numerous fabricated scenarios, and, perhaps, it may find the real world to be similar enough to others experienced in training, bootstrapping its strategy from one of these trained scenarios.

Even though specifying $\boldsymbol{\psi}_*(t)$ is an important step, it may not be the only one needed. The system's internal state needs to be enriched so the agent can better recognize the situations it may find, i.e., the internal state must have a strong Markov property. One of the most immediate ways to enrich the system's internal state is by introducing the uncertainty of the estimated demand model parameters, which computation is presented in Chapter 4. However, the uncertainty of the demand model parameters only tells the agent about the believed quality of the estimated demand model parameters, but nothing about the nature of changes in demand behavior, such as abrupt and smooth changes, nor the intensity of such changes. In other words, when given only the demand model parameters and their corresponding uncertainty, the system cannot distinguish a large uncertainty due to the poor quality of the historical data



(a) Example of normality.

(b) Example of outlier.

Figure 5.2: The system generates trajectories (blue) from an arbitrary state in the past S_{t-n} while following its pricing policy and simulated customers generated according to the calibrated demand model. The system can compare the simulated trajectories to the observed trajectory between states S_{t-n} and S_t.

caused by the system's pricing policy from the one originated by a change in the customer behavior.

A *shock detector* is a function/module that tracks demand changes in real–time. Roughly, a shock detector compares freshly observed data with the model predictions. As the demand model is built with data from past interactions, a change in demand behavior can be detected when the newly obtained data do not fit within model predictions (i.e., low probability of observing this new data). For example, the probability of observing specific states can be computed using the model of the demand behavior, as explained in Example 3.2.4 (on page 47). A probable cause for experiencing unlikely states (or states where the probability of being experienced is below a certain threshold, such as the white region in Figure 3.5) is a change in the demand behavior, thus being a practical signal to monitor.

The shock detection mechanism described above uses only the information about states (state–space distribution), however, the information about actions can also be used (state–action space distribution). The demand shock detector is presented formally in the Appendix, but Figure 5.2 illustrates the procedure of the state–action space shock detector. The system chooses any arbitrary observed state S_{t-n} and, from this state, it simulates a set of trajectories with Monte–Carlo rollouts while using its policy and generating demand according to its estimated demand model. Trajectories in the state–action space can be summarized with the state–transition probability function and compared to the "expected" ones obtained through simulation. When the observed trajectory matches the distribution of simulated ones, as represented in Figure 5.2 (a), the shock detector concludes with the absence of a change in demand behavior. On the contrary, when the observed trajectory is too extreme for the simulated ones, as represented in Figure 5.2 (b), the shock detector concludes with

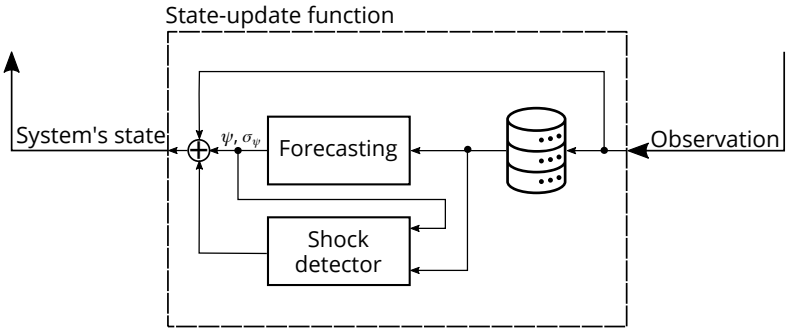


Figure 5.3: Representation of the state–update function for non–stationarity.

a change in demand behavior.

The shock detector can also provide information about the "direction" of a change in the demand behavior. For example, suppose that several active flights are observing an excess of demand that is very unlikely due to chance. When the system's capacity goes down too quickly, its natural response is to increase prices (see Figure 2.6), but note that this scenario is somewhat different from the one assumed in Chapter 2: Many flights are observing their capacities decrease faster than expected despite price increases. This phenomenon indicates a *positive* shock, where there are more customer arrivals or these customers are more willing to pay. In contrast, if the system observes many active flights having fewer bookings than expected, then the system may be experiencing a *negative* shock. Furthermore, the shock detector provides an estimate of the "intensity" of such changes because the likelihood of observing a trajectory is related to how much the demand behavior has changed.

In summary, as represented in Figure 5.3, the estimated parameters, their uncertainties, and the shock detector may consist of a rich set of signals that can help the agent evaluate the situation. The agent can be trained to recognize and interpret these signals through examples of non-stationary demand behavior included in the training, and the designer can implement these examples according to his goals and intuition.

5.2.2 Self–competition

When optimizing prices, airlines consider the independence of the demand for flights of the same leg. As discussed in Chapter 4, this is inaccurate because the booking data collected for each flight is appended to the historical database and later used for estimating the demand model parameters. However, there is a second reason why this is not true. In the real world, customers may anticipate or delay their trip to save money, forcing flights from the airline to compete for the same demand, i.e., *self–competition* (or cannibalization) [14, 104]. Throughout

[14]: Gallego and Phillips (2004), "Revenue management of flexible products"
[104]: Bront, Méndez-Diaz, et al. (2009), "A column generation algorithm for choice-based network revenue management"

this work, we deliberately assumed this was not possible to keep the problem statement simple, but, in practice, this assumption is far from realistic.

The first step to address the self–competition requires modifying the simple binary customer choice model (accept or reject the offer) assumed so far. There are many ways to design a customer choice model that accommodates complex behaviors [105], but, for illustration, consider the *multinomial discrete choice model*. This choice model defines the probability of selecting each option, let's say, option A and option B, according to the soft–max distribution

[105]: Garrow (2016), *Discrete choice modelling and air travel demand: theory and applications*

$$\Pr\{\mathrm{A} \mid \mathrm{A}, \mathrm{B};\ \boldsymbol{\omega}\} \doteq \frac{e^{h(\mathrm{A};\,\boldsymbol{\omega})}}{1 + e^{h(\mathrm{A};\,\boldsymbol{\omega})} + e^{h(\mathrm{B};\,\boldsymbol{\omega})}},$$

where $h(o;\ \boldsymbol{\omega}) \in \mathbb{R}$ is a preference function causally specified. For example, the preference function could be the dot–product between the features of the evaluated option $\mathbf{x}(o)$ and the feature weights $\boldsymbol{\omega}$, i.e., $h(o;\ \boldsymbol{\omega}) \doteq \mathbf{x}(o) \cdot \boldsymbol{\omega}$. The features of an option can be anything relevant to customers when making a choice (e.g., if customers are choosing between different models of smartphones, the features could represent memory, display size, and camera quality). The probability of choosing none of the available options is given by

$$\begin{aligned}
\Pr\{\varnothing \mid \mathrm{A}, \mathrm{B};\ \boldsymbol{\omega}\} &= 1 - \Pr\{\mathrm{A} \mid \mathrm{A}, \mathrm{B};\ \boldsymbol{\omega}\} - \Pr\{\mathrm{B} \mid \mathrm{A}, \mathrm{B};\ \boldsymbol{\omega}\} \\
&= 1 - \frac{e^{h(\mathrm{A};\,\boldsymbol{\omega})}}{1 + e^{h(\mathrm{A};\,\boldsymbol{\omega})} + e^{h(\mathrm{B};\,\boldsymbol{\omega})}} - \frac{e^{h(\mathrm{B};\,\boldsymbol{\omega})}}{1 + e^{h(\mathrm{A};\,\boldsymbol{\omega})} + e^{h(\mathrm{B};\,\boldsymbol{\omega})}} \\
&= \frac{1}{1 + e^{h(\mathrm{A};\,\boldsymbol{\omega})} + e^{h(\mathrm{B};\,\boldsymbol{\omega})}}.
\end{aligned}$$

Let's assume customers arriving for the i-th active flight follow a Poisson distribution with mean $\nu(i)$. These customers aim at purchasing the flight departing at time $i = t + \tau$, but they can anticipate, delay or abandon their trip according to the offers made for all active flights at the instant they arrive. Customers choose between flights according to the multinomial discrete choice model, and the probability of purchasing the i-th active flight can be denoted by

$$z\left(i \,\middle|\, f^{[0]}, \ldots, f^{[\mathrm{T}-1]};\ \boldsymbol{\omega}\right) = \Pr\left\{\text{purchase } i\text{-th flight} \,\middle|\, f^{[0]}, \ldots, f^{[\mathrm{T}-1]};\ \boldsymbol{\omega}\right\}.$$

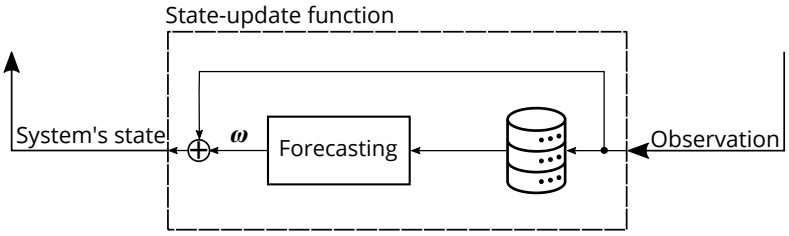


Figure 5.4: Representation of the state–update function for self–competition.

Finally, the exponential demand model (from eq. (2.2)) can be rewritten as a function of the i-th active flight as

$$d(i;\boldsymbol{\omega}) = v(i)\, z\left(i \,\middle|\, f^{[0]}, \ldots, f^{[\mathrm{T}-1]};\, \boldsymbol{\omega}\right).$$

In practice, the true parameters of the demand model $\boldsymbol{\omega}$ are unknown and need to be approximated from historical data, which may be achieved by adapting the likelihood function in eq. (2.3).

Then, the system needs to optimize its offers to maximize revenue. For the same reasons discussed in Chapter 4, this last step quickly becomes intractable to dynamic programming methods as the horizon T increases because the state–action space quickly becomes too large, requiring the development of heuristic methods to assist price optimization [106]. Instead, adapting Algorithm 4 to self–competition is straightforward. For the first step, we introduce the multinomial choice model to the true demand behavior (i.e., update the environment). Then, the new model parameters are estimated from historical bookings; these estimations are then appended to the system's internal state, as illustrated in Figure 5.4 (i.e., update the state–update function). Everything else stays unchanged: The learning system updates the learned weights through RL while generating experience according to a specified demand model in which parameters are sampled arbitrarily from a range for a predefined number of time steps.

[106]: Gallego, Ratliff, et al. (2015), "A general attraction model and sales-based linear program for network revenue management under customer choice"

5.2.3 Competition

Perhaps the most unrealistic assumption made in this work is that the airline is the only service provider (monopoly). In practice, many airlines may operate the same leg, allowing customers to select which offer best suits them. In other words, the probability that customers will book a flight from the airline depends not only on its prices but also on the offers made by the other airlines.

In a certain sense, competition is very similar to self–competition, but it presents particular characteristics and thus must be treated separately. The most remarkable difference is that the historical booking data for the competing airlines are not directly observable, i.e., airlines have access to their booking data only and not the booking data from the others, making it much harder to estimate the parameters of a dependent demand model. Price optimization is also more complex because it depends on the pricing strategy of competing airlines, which can change over time and is unknown (the other airlines' remaining capacities and pricing optimization may not be publicly available information). Furthermore, the systems of competing airlines may need some iterations of adapting strategies until convergence to a Nash equilibrium[1] [17].

1: In game theory, the Nash equilibrium is one of the common ways to define the solution in non–cooperative games, where no player has anything to gain by changing only one's own strategy.

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"

To illustrate how the system may be adapted to competition, let's assume that customers can choose among two different airlines (duopoly). For each active flight, the airline selects a fare f_{al}, while the competing airline selects a fare f_{oal} ('oal' abbreviates "other airline"). We can rewrite the purchase probability as

$$z(f_{al} \mid f_{oal}; \boldsymbol{\omega}) = \Pr\{\text{purchase at fare } f_{\text{al}} \mid f_{\text{oal}}; \boldsymbol{\omega}\}$$

where $\boldsymbol{\omega}$ represents the customer preferences. These purchase probabilities can be described with the multinomial discrete choice model or any other choice model that the designer considers relevant. In theory, the customer preference parameters $\boldsymbol{\omega}$ can be estimated from historical data by modifying the maximum likelihood estimation in eq. (2.3) or by any other supervised learning method.

For using the above demand model for optimization, the system needs to know how the competitor will price their offers in the future, which is unknown when making a choice. However, the competitor's historical prices are available to the airline and can be used to predict how, to some extent, the competitor may price its products in the future [11]. Roughly, the system can learn the parameters $\boldsymbol{\chi}$ of a pricing prediction function that approximates the competitor's pricing policy $\pi(\tau; \boldsymbol{\chi}) \approx \pi_{\text{oal}}(S_t^{\text{oal}} = (\tau, c^{\text{oal}}))$ while using the competitor's historical prices, where c^{oal} is the unknown remaining capacity of the other airline.

[11]: Fiig, Wittman, et al. (2019), "Towards a competitor-aware RMS"

Once the system computes the pricing prediction function, it must optimize its pricing policy accordingly. In principle, the system can perform price optimization with DP if the customer choice model can be mapped to the state–transition probability function, as in Chapter 2. Alternatively, integrating competition

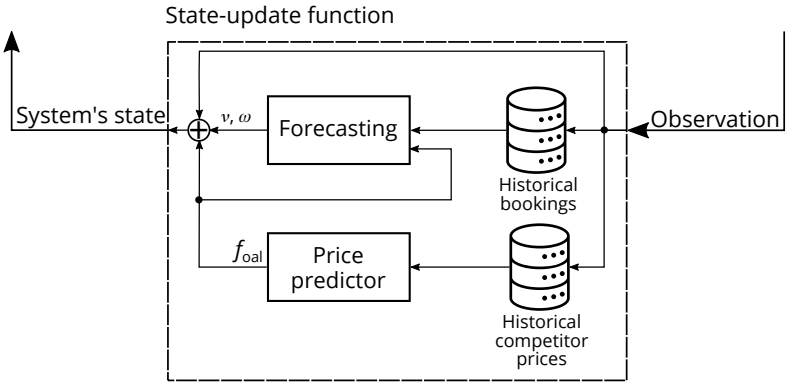


Figure 5.5: Representation of the state–update function for competition.

into Algorithm 4 requires adapting observations to include the competitor's prices and current inventory states, and extending the system's internal state to include the new demand model parameters and the predicted competitor's price, as illustrated in Figure 5.5. On the environment side, the customer choice model needs to be updated to match the desired behavior (e.g., a multinomial choice model for duopoly) and add the presence of a competitor pricing system that could be a legacy RMS or another learning agent. Similar to the previous sections, everything else stays unchanged, illustrating the simplicity and the powerful general nature of Algorithm 4.

5.3 Learning the state–update function

In the previous sections, we discussed how to adapt Algorithm 4 to many other problems of airline RM. The adaptations focused on the inputs and outputs of a state–update function, which is responsible for translating the observable components of the demand behavior into the system's internal state. The agent's performance should improve as the Markov property of this internal state becomes stronger, and the correct design of this function requires expert knowledge about the problem.

Developing and calibrating the state–update function can be difficult, requiring a significant amount of the expert's time and many trial–and–error attempts. Ideally, we would like to automate the search procedure and integrate the design of the state–update function into the learning process itself. But, how could it be achieved?

Before addressing *how*, let's first consider more deeply *why*. In artificial intelligence research, we observe the recurrent pattern that generic–learning methods, such as RL and deep learning, tend to outperform human–knowledge–based methods, such as the heuristic methods discussed in this work. For example, in speech recognition, early participants of the competition

sponsored by the *Defense Advanced Research Projects Agency* used methods that took advantage of human knowledge about speech, such as words and phonemes, contrary to the more computation–intensive generic–learning methods. Nonetheless, the learning methods won out over the human–knowledge–based methods until they became the dominant trend in the field. Similarly, early methods in computer vision searched for edges, generalized cylinders, or scale–invariant features. Today, this has been all replaced by modern ANNs that use only the notions of convolution methods. These examples illustrate that, in the short term, methods using human knowledge of the domain can display better results, but, in the long run, the only thing that matters is leveraging data and computation power. Even though these two perspectives do not need to oppose each other, they do in practice because time spent in one is time not spent in the other [107]. There is little reason to believe that the case of airline RM is any different.

[107]: Sutton (2019), *The bitter lesson*

Perhaps one of the most striking and enlightening examples of how RL can be used to learn its representations comes from Go. As presented in Chapter 3, when training an RL agent to play Go, researchers defined observations as $19 \times 19 \times 48$ image stacks, in which each point of the 19×19 Go board was represented by 48 binary or integer features designed according to what the research team believed to be important. Furthermore, researchers introduced further human support to the learning system by warming up the ANN to predict moves from expert play [35]. This version of the RL agent, later re–branded as AlphaGo Lee, shocked the field of AI when it defeated the 18-time world champion Lee Sedol. Not long after, researchers developed a new version of the agent, branded as AphaGo Zero, which bested the previous version, AlphaGo Lee, by 100 games to 0 [108]. AlphaGo Zero is different from AlphaGo Lee in many ways, but two of the most significant differences are that the agent is trained exclusively with self–play (no use of expert games) and that it uses only black and white stones as the input features. More precisely, the ANN took as input the $19 \times 19 \times 17$ image stacks of 17 binary features, in which the first 8 feature planes were the raw representation of the positions of the current player's stones in the current and past board configurations, the following 8 feature planes were similar, representing the current and past board configurations for the opponent stones, and the final feature plane had a constant value indicating the color of the current play (0 for white, 1 for black). The need for representing past board configurations and the color features was motivated by some rules of Go that forbid repetition (*ko* rule) and that White is given "compensation points" (*komi*) for not getting the first move (Black moves first), making only the current board

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

[108]: Silver, Schrittwieser, et al. (2017), "Mastering the game of go without human knowledge"

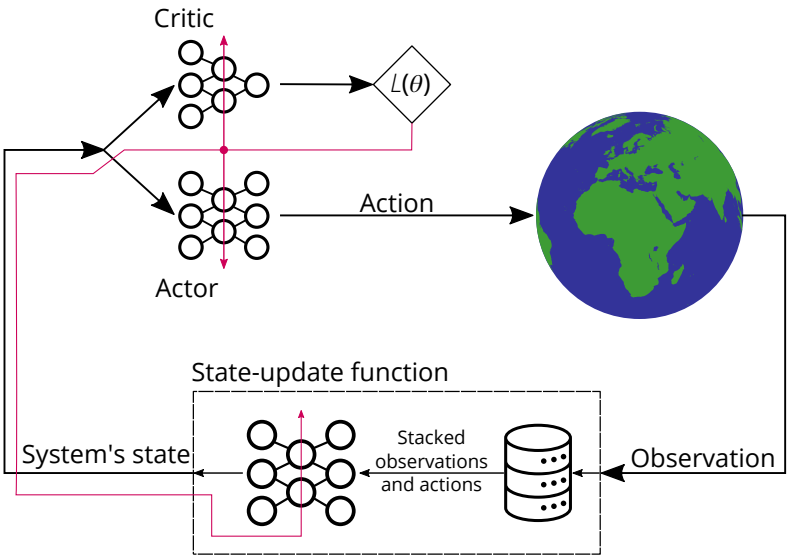


Figure 5.6: Representation of a generic state–update function using artificial neural networks. The red path illustrates backpropagation.

position not a Markov state (weak Markov property).

Instead of designing the state–update function ourselves, as illustrated in the past sections, we can imagine a system similar to AlphaGo Zero that uses an ANN to encode a stack of past observations into the summarized system's internal state. This ANN could use an encoder–decoder architecture similar to the one presented in Figure 4.11 or a powerful transformer architecture[2] [109] that can encode information over long horizons.

Consider that the system's internal state representation can be obtained through a parametrized state–updated function $S_t = u(S_{t-1}, A_{t-1}, O_t; \boldsymbol{\theta}_u)$. The agent's goal is to learn the parameters $\boldsymbol{\theta}_u$ such that the state representation S_t is useful for making predictions. The parameters of this function can be learned directly through RL by backpropagating from actor and critic networks to the same shared body (state–update function), as illustrated in Figure 5.6. The parametrized state–update function would try to learn representations (summarized history) to support the decisions made by the actor and critic networks.

To backpropagate learning to the same shared body, such as the state–update function, at the same time as updating the actor and critic networks, the system's prediction multi–objective needs to be rewritten into a single optimization objective, which can be achieved with

$$L(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_v) = L_\pi(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi) - k\, L_v(\boldsymbol{\theta}_u, \boldsymbol{\theta}_v), \tag{5.1}$$

where k is an arbitrary coefficient, $\boldsymbol{\theta}_\pi$ and $\boldsymbol{\theta}_v$ are the ANN independent weights for actor and critic, respectively. L_π is the policy gradient objective given by

2: Unlike recurrent neural networks, such as long short–term memory networks, transformers do not necessarily process data in order, i.e., they use the attention mechanism to provide a context for any position in the input sequence.

[109]: Parisotto, Song, et al. (2020), "Stabilizing transformers for reinforcement learning"

$$L_\pi(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi) = \hat{\mathbb{E}}[\delta_t \ln \pi(A_t|O_t; \boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi)],$$

and L_v is the value function objective given by

$$L_v(\boldsymbol{\theta}_u, \boldsymbol{\theta}_v) = \hat{\mathbb{E}}\left[(R_{t+1} + \gamma v(O_{t+1}; \boldsymbol{\theta}_u, \boldsymbol{\theta}_v) - v(O_t; \boldsymbol{\theta}_u, \boldsymbol{\theta}_v))^2\right].$$

This objective $L(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_v)$ can be (approximately) maximized with stochastic gradient ascent, as in Algorithm 4.

Learning the state–update function while solving the task can allow us to address the earning–while–learning, non–stationarity, self–competition, and competition problems using the same generic training architecture. Finally, the designer is left with the task of defining, on the environment side, the functional form of customer behavior that needs to be optimized.

5.4 What about forecasting?

When using ANNs to encode past observations, the task of mapping interactions with the demand into a summarized state is transferred to the learning system. In principle, performing price optimization does not require explicit forecasting. Naturally, this raises a question about the necessity of the forecasting module. However, the forecasting module is not only employed for price optimization. Anticipating the travel demand can support many managerial decisions, such as deciding whether to open or close services to specific destinations, the staffing levels at the airport (for check-in, lounge, security), or the aircraft type the airline needs to buy. Therefore, the forecasting module cannot be simply given up.

Perhaps the most immediate solution consists in maintaining a separate forecasting module that is not used for price optimization. Nonetheless, this might not be the best solution. Alternatively, we could add a new *head* to the output of the RL agent (alongside the value prediction and the policy) that is responsible for demand forecasting, as illustrated in Figure 5.7. To train this new head, update the objective function in eq. (5.1) needs to be updated to accommodate this new *auxiliary task*

$$L(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_v, \boldsymbol{\theta}_d) = L_\pi(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\pi) - k_1 L_v(\boldsymbol{\theta}_u, \boldsymbol{\theta}_v) - k_2 L_d(\boldsymbol{\theta}_u, \boldsymbol{\theta}_d),$$

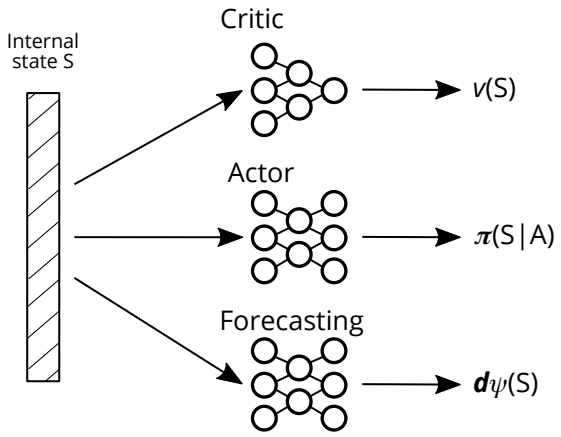


Figure 5.7: Representation of a generic neural network that shares an internal representation for price optimization, value function prediction, and forecasting tasks. In this example, the head responsible for forecasting the demand is here represented by a vector $\boldsymbol{d}_\psi$ where each component is the expected demand for each price point in the fare structure.

where L_d represents the new forecasting prediction objective, and $\boldsymbol{\theta}_d$ represents the ANN weights of the layers dedicated to forecasting. We can add as many heads as tasks one would like the system to learn, allowing the agent to predict and control many signals and not just the long–term revenue.

Why is this solution any better? One possible answer is that the ability to predict and control many signals may be a form of an environmental model [27]. For example, forecasting demand has a strong relationship with price optimization but may have a shorter delayed response than rewards and less variance between actions and outcomes. Consequently, the RL agent can use the same internal representations for predicting demand to optimize prices, potentially reducing the amount of data needed to learn the main task, which is revenue maximization. There are no specific reasons why this has to be true, but, in this case, it seems plausible. Researchers have already experimented with learning auxiliary tasks, such as predicting pixels, distribution of returns, and predicting future states [49, 74, 110]. In many cases, this approach has shown significant success, accelerating learning on the main task.

[27]: Sutton and Barto (2018), *Reinforcement learning: An introduction*

[49]: Jaderberg, Mnih, et al. (2016), "Reinforcement learning with unsupervised auxiliary tasks"
[74]: Pathak, Agrawal, et al. (2017), "Curiosity-driven exploration by self-supervised prediction"
[110]: Bellemare, Dabney, et al. (2017), "A distributional perspective on reinforcement learning"

5.5 Summary

As revenue management systems (RMSs) cannot observe the true demand behavior directly, they fall in the category of partially observable problems. Optimization of partially observable Markov decision processes is a topic of research in the field of reinforcement learning (RL), and many methods for addressing this issue have been proposed in the past.

The sequence of observations and actions can be transformed into a summarized view, named the *internal state*, as typically done by RMSs when it estimates the parameters of the demand

model. The agent should perform better in its task as stronger the Markov property of this internal state is.

A function summarizing the history of observations is referred to as the state–update function. There are many ways to build this function according to the task at hand. For example, if addressing non–stationarity, the internal state can be enriched with a shock detection signal, whereas, when addressing competition, the competitor's price prediction can be added.

Instead of manually designing the state–update function, artificial neural networks can learn representations through backpropagation and RL. By integrating the design of this function into the learning process, the search procedure is automated, thus reducing the expert's design effort.

Even though explicit forecasting and many other modules could be, in principle, eliminated from the system, this may not be desirable because some of these modules have roles beyond price optimization. Instead, the RL agent's outputs can be extended to address such auxiliary tasks. Perhaps another reason to extend the agent to auxiliary tasks is that it may improve data efficiency, as many studies in the field of RL have found.

6 Conclusions

Many fields of research, such as robotics and navigation [111–113], are being revolutionized by reinforcement learning (RL), and revenue management (RM) may be one of the next ones. Early studies bringing RL into the field of RM [17, 46, 47] focused on the model–free aspects of RL, i.e., the ability to optimize prices without relying on expert–designed demand models. Although this is an appealing idea, it is unclear whether learning online from real–world demand is possible, as discussed in Section 3.5.

Most successful applications of RL to real–world problems use a degree of a model of the environment, such as the rules of the game when training the agent to play Go [35] or the laws of physics when training the agent to control the magnetic fields of a tokamak [42]. Across all these fields, there is one constant motivational factor for using that RL: Researchers seek to address complex control problems through generic learning methods with little or no human support.

As discussed in Chapter 1, there are many open problems in the field of RM that have long relied upon expert intuition and heuristic methods, such as earning while learning (EWL), competition, self–competition, and non–stationarity. These heuristics are not easy to adapt to the sophisticated demand models of modern real–world revenue management systems (RMSs), being perhaps the main reason why these systems, to the best of our knowledge, do not implement them. We believe that heuristic methods are not to be blamed, but rather the complex nature of these problems: Addressing such complex problems requires powerful tools. Therefore, in this work, our goal is to demonstrate that an RL agent can be trained offline to address RM complex problems without expert guidance.

It was demonstrated, in Chapter 4, how to design an RL agent capable of discovering solutions to the EWL problem that delivers better revenue performance than state–of–the–art heuristic methods. Furthermore, in Chapter 5, it was described how this same RL agent could be adapted to address other open issues as well. We argue, in Section 5.3, how to design an RL agent that can learn its internal representations, further simplifying the system's design. Consequently, RM experts would be left only with the task of specifying the demand model used for training. In other words, the "model–free" aspect of RL enables price

[111]: Xie, Berseth, et al. (2018), "Feedback control for cassie with deep reinforcement learning"

[112]: Akkaya, Andrychowicz, et al. (2019), "Solving rubik's cube with a robot hand"

[113]: Bellemare, Candido, et al. (2020), "Autonomous navigation of stratospheric balloons using reinforcement learning"

[17]: Bondoux, Nguyen, et al. (2020), "Reinforcement learning applied to airline revenue management"

[46]: Kastius and Schlosser (2021), "Dynamic pricing under competition using reinforcement learning"

[47]: Shihab, Logemann, et al. (2019), "Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking"

[35]: Silver, Huang, et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

[42]: Degrave, Felici, et al. (2022), "Magnetic control of tokamak plasmas through deep reinforcement learning"

optimization of complex demand behaviors without the need for adapting the many modules of an RMS. Price optimization can be delegated to an end–to–end system that learns to solve problems exclusively by maximizing the standard reward signal (i.e., revenue maximization) without the need for any particular engineering of this signal [99].

[99]: Silver, Singh, et al. (2021), "Reward is enough"

Relying on RL for guiding the search for solutions to complex problems is perhaps the most significant change in the RM research field. RL shifts the researchers' focus from *how* to solve problems toward *what* problems need to be solved, i.e., the agent's task is to discover the solution of a problem specified by the human designer. **RL experts** could focus on investigating different ANN architectures, more adapted RL methods, enhanced computation capabilities, and changes in the modeling of the observation/action space, while **RM experts** could concentrate on designing the training environment (e.g., customer choice model) that the RL agent relies upon for training. Hopefully, such an approach may accelerate the research process in the RM domain.

The impact of RL goes beyond the research field. RL brings a new set of organizational challenges. For example, in the case of airline RM, the revenue performance would no longer be measured and optimized at the flight level but rather at the level of the whole airline network, making it difficult to assign the revenue responsibility to a specific RM analyst. Furthermore, RM analysts spend a significant amount of time monitoring, investigating, and fixing forecast mistakes made by the system. However, RL reduces the interpretability of the system choices and the flexibility of manipulating its outputs. Instead, RM analysts could focus on identifying new training situations and enriching the demand model used for training the agent, representing a significant transformation of their duties.

RL can also impact the operational aspects of RMSs. By having shorter research, tuning, and moving into production loops, RL enables a much faster response to new market situations. Furthermore, RL replaces the complex nested control structures of the RMS with a single computational unit, making it much easier to upgrade (changes in artificial neural network structure or uploading new training weights) and query (interrogate the artificial neural network for its predictions). The agent's ability to learn from generic inputs can also be a critical asset since the system could use various other alternative sources of data (such as weather forecasts) that today are often difficult to integrate into real–world RMSs.

In summary, RL may transform the field of RM profoundly. It

can accelerate research, simplify the system's architecture, and speed up deployment to production systems. This enhanced organization can improve response time to changing markets and competition.

Appendix

Journal of Revenue and Pricing Management
https://doi.org/10.1057/s41272-022-00385-8

RESEARCH ARTICLE

Demand change detection in airline revenue management

Giovanni Gatti Pinheiro[1,3] · Thomas Fiig[2] · Michael D. Wittman[2] · Michael Defoin-Platel[1] · Riccardo D. Jadanza[4]

Received: 10 September 2021 / Accepted: 1 May 2022

Abstract
Demand shocks—unobservable, sudden changes in customer behavior—are a common source of forecast error in airline revenue management systems. The COVID-19 pandemic has been one example of a highly impactful macro-level shock that significantly affected demand patterns and required manual intervention from airline analysts. Smaller, micro-level shocks also frequently occur due to special events or changes in competition. Despite their importance, shock detection methods employed by airlines today are often quite rudimentary in practice. In this paper, we develop a science-based shock detection framework based on statistical hypothesis testing which enables fast detection of demand shocks. Under simplifying assumptions, we show how the properties of the shock detector can be expressed in analytical closed form and demonstrate that this expression is remarkably accurate even in more complex environments. Simulations are used to show how the shock detector can successfully be used to identify positive and negative shocks in both demand volume and willingness-to-pay. Finally, we discuss how the shock detector could be integrated into an airline revenue management system to allow for practical use by airline analysts.

Keywords Change point detection · Demand forecast error · Airline revenue management · Demand shock · Forecasting · Markov decision process

Introduction

Motivation for shock detection in airline revenue management

Airline revenue management (RM) analysts often spend a significant portion of their time searching for and correcting forecast errors in the airline's revenue management system (RMS). These forecast errors can be costly to airlines—one study found that as little as a 10% error in an RMS demand forecast can be associated with a 1% decrease in airline revenue (Fiig et al. 2019).

Forecast errors fundamentally occur due to a mismatch between the demand model parameters assumed by the RMS forecaster and the customer behavior in the marketplace. Usually, shifts in customer behavior are automatically captured by the RMS during forecast parameter re-estimation, which typically uses a historical database consisting of departed flights. However, when customer behavior suddenly changes, the RMS can struggle to adapt quickly, since it takes time for the new behavior to enter the historical database and be detected by the parameter re-estimation.

We refer to these sudden, abrupt changes in customer behavior as *demand shocks*. Demand shocks vary in intensity and can occur at the macro- or micro-level. The COVID-19 pandemic is one example of a highly impactful macro-level demand shock that affected demand across a wide range of flights and origin–destination (O&D) markets, while micro-level demand shocks affecting a handful of flights or markets frequently occur due to entry or exit of a competitor, special events such as conferences, concerts, or sporting competitions, changes in airline schedules, etc., that were not already anticipated and corrected by the airline analyst.

Airline analysts typically identify and address demand shocks via relatively simplistic alerting mechanisms. For

✉ Michael D. Wittman
Michael.WITTMAN@amadeus.com

1 Amadeus S.A.S., Avenue Jack Kilby, 06270 Villeneuve-Loubet, France

2 Amadeus IT Group, Lufthavnsboulevarden 14, 2. tv, 2770 Kastrup, Denmark

3 University of Nice Sophia-Antipolis, Nice, France

4 Enerbrain SRL, Strada alla Villa d'Agliè 26, 10132 Turin, Italy

Published online: 06 August 2022

example, Weatherford (2019) describes how an analyst might set an alert to trigger if certain criteria for a flight departure date, such as current load factor (LF), falls above or behind a predefined threshold (e.g., greater than ±5 p.p. compared to the previous year) at a given time prior to departure. If an individual flight is alerted, the analyst would then apply a demand intervention to adjust the forecast for that flight. Vinod (2021) also describes a similar workflow where analysts define alerts by comparing key performance indicators (KPIs) from the RMS to predefined thresholds, conduct a root-cause analysis, and then apply interventions to forecasting or availability in response to a triggered alert.

These methods for detecting demand shocks face several limitations. First, they are often quite rudimentary and rely on imprecise heuristics or rules of thumb. Since analysts are not provided with guidance on how to set the alert thresholds, they may either miss impactful shocks (false negatives, Type II error) or be overwhelmed with alerts that, after investigation, turn out to be normal behavior (false positives, Type I error).

Additionally, these threshold-based approaches often evaluate flights one at a time, ignoring wider-scale demand shocks that affect multiple departure dates or markets at the same time. They also do not directly consider the effect of offered prices on demand behavior. For example, they may alert an analyst to a flight with a very high current load factor without considering whether the prices offered for that flight were higher or lower than the previous year. In contrast, our method considers the offered prices for each flight when determining whether or not a demand shock has occurred.

Finally, traditional approaches to shock detection often consider KPIs taken at a single snapshot when the alerts were generated. Our method utilizes all accessible information—bookings and demand forecast given the control policy—across the entire booking horizon of each flight. Our approach also aggregates data across multiple active flights, allowing for faster and more accurate detection of shocks. Since analysts are often responsible for hundreds or thousands of flight departure dates at a time, this approach allows for greater efficiency and less time spent identifying demand shocks.

Contributions

In this paper, we introduce a science-based framework for demand shock detection that aims to improve airline analysts' ability to identify sudden changes in demand. Our detector is based on well-known approaches for statistical hypothesis testing which we have adapted for the shock detection problem in airline revenue management. Given an observed set of booking activity for one or more active (non-departed) flights, we compute the log likelihood that those observations occurred given the offered prices and the RMS's demand forecast. If the log likelihood—assuming no shock—deviates from a calculated acceptance range, this indicates a poor model description by the forecast parameters and leads to the conclusion that a demand shock has occurred.

We show how the statistical properties of the shock detector, such as Type I error, Type II error, time since shock, etc., can be described in analytical closed form under simplifying assumptions about the demand environment and RMS policy. We find that the properties that we derive also generalize well to more complex environments with capacity constraining or time-dependent willingness-to-pay. We then show that the properties of the shock detector based on simulations can be accurately predicted from the analytical closed form expression, even in these complex environments.

Finally, we demonstrate how the shock detector outputs could be used in practice via an alert center application to allow for efficient prioritization of shocks for investigation.

Literature review

Academic research relating to detecting change in stochastic processes—so-called Change Point Detection (CPD)—has been conducted for nearly one hundred years. As described in a literature review by Lai (1995), the first CPD mechanisms date back to the 1930s and have been frequently used in manufacturing and quality control applications to detect systematic shifts in time series data. Table 1 summarizes some of the relevant literature in the field.

We distinguish between CPD in an online and offline setting. In online testing, the dataset is not available upfront but is gradually collected over time. For every new observation, a test is performed. If no change is detected, we continue to the next time step, while if a change is detected, we stop and raise an alarm. The online form is not of main interest in this paper because for our purpose, the full dataset of booking activity on active flights is given up front.

Basseville and Nikiforov (1993) provide a theoretical framework of many CPD mechanisms, including the well-known Cumulative Sum (CuSum) method, which is available in both online and offline forms. CuSum works by accumulating deviations between observations and their expectations and identifying a change if the accumulated deviations become too extreme with respect to a predefined threshold. CuSum is frequently used when analyzing time series data to identify moments when the underlying demand generating process appears to have changed.

As Besbes and Zeevi (2011) point out, many CPD mechanisms assume that the post-shock behavior is known, reducing the problem to identifying the shock as quickly as possible. We will discuss shock detection with and without known post-shock demand behavior. Classical CPD mechanisms

Table 1 Selected literature on change point detection

Paper	Subject	Capacity constraint?	Time horizon	# of shocks	Post-shock params known?	Demand model
Besbes and Zeevi (2011)	"Learning while earning"	No	Infinite	One	Yes	Parametric
Garivier and Moulines (2011)	"Learning while earning"	No	Infinite	Multiple	No	Parametric
Broder and Rusmevichientong (2012)	"Learning while earning"	No	Infinite	One	Yes	Non-parametric
Besbes and Sauré (2014)	"Learning while earning"	Yes	Finite	One	Yes	Parametric
den Boer (2015)	"Learning while earning"	No	Infinite	Multiple	No	Parametric
Keskin and Zeevi (2017)	"Learning while earning"	Yes	Infinite	Multiple	No	Parametric
den Boer and Keskin (2020)	"Learning while earning"	Yes	Finite	Multiple	No	Parametric
Keller and Rady (1999)	MDP demand	No	Infinite	One	Yes	Parametric
Aviv and Pazgal (2005)	MDP demand	No	Finite	Multiple	Yes (in exp.)	Parametric
Hadoux et al. (2014)	Change−point detection	N/A	Both	One	Yes	Non-parametric
This paper	Shock detection	Yes	Finite	One	No	Parametric

also assume that all samples are drawn from the same underlying distribution, while in the airline RM problem, the sample distribution is dependent on the states in state space which have been visited. Our methodology addresses this problem by calculating the probability of observing a particular trajectory of state-action pairs in state-action space.

Also related to our setting is a series of papers in the operations research literature that consider online "learning while earning" under conditions of demand uncertainty. In these papers, a retailer sets prices for a product that exhibits unknown demand behavior, and periodically estimates the parameters of the demand model from sales data. The retailer's goal is to learn the demand behavior as quickly as possible to maximize long-term revenue by charging the optimal price. Common among many "learning while earning" papers is that the selling period is indefinite, and that the retailer continues to collect information about the environment in perpetuity. The retailer must then decide which historical data to use in their estimation of customer behavior, since old data may have been collected under a different demand model.

Besbes and Zeevi (2011) and Broder and Rusmevichientong (2012) consider scenarios where the demand behavior undergoes a single demand shock. The retailer knows both the pre-shock and post-shock demand behavior but does not know when the shock occurs. They describe how price experimentation strategies can help the retailer identify the time of the shock and maximize the long-term revenue.

"Learning while earning" can also be applied in environments with multiple demand shocks. Garivier and Moulines (2011) describe how a multi-armed bandit approach can be used to perform price experimentation in order to minimize revenue regret in such a setting. Den Boer (2015) considers a dynamic pricing problem with a more complex demand model with parameters that change continuously over time. Keskin and Zeevi (2017) describe several methods for estimating demand behavior in an environment with multiple demand shocks while assuming a given limit of how much the demand parameters can change from one period to the next.

Few "learning while earning" papers consider a setting similar to airline RM where capacity is constrained and the selling horizon is finite. An exception is Besbes and Sauré (2014), who study a situation in which a demand forecast model experiences a single demand shock. The retailer does not know the time of the demand shock nor the post-shock behavior, but the post-shock behavior is revealed to the retailer at the time of the shock. The goal is to set a pricing policy to maximize revenue given an unknown shock that will be revealed at some point in future. den Boer and Keskin (2020) also review a dynamic pricing problem with a finite selling horizon. Their demand function allows for multiple discontinuity points and unknown pre- and post-shock demand behavior. Their focus is not on shock detection, but rather on theoretically constructing a pricing policy that incorporates the possibility of demand discontinuities.

Our work is also related to a series of papers that describe the demand change process as a Markov process. Keller and Rady (1999) study a setting where demand shifts between two known linear demand functions. The retailer knows the demand functions but does not know which demand function is active. Aviv and Pazgal (2005) describe an environment with Poisson demand where the demand function fluctuates between multiple "core states" with different demand behavior via a Markov process. They use this framework to describe static environments, those with decreasing demand, and those with increasing demand. The retailer bases their prices on a partial observation of the Markov Decision Process (MDP), by assuming a prior belief of the current core state.

Perhaps most similar to our work is the paper by Hadoux et al. (2014). They describe how the CuSum method can be adapted in an online setting to detect changes in state transitions in an MDP. Their method is not directly applicable in the airline RMS setting, since they do not consider how a change intersects multiple flight departures at different times during an episode. In contrast, our shock detection method can detect a shock that simultaneously affects all flights across a given market. Further, their paper assumes known post-shock parameters, whereas we also consider the most relevant case for RMS of unknown post-shock parameters.

Problem formulation

We consider the optimization problem for a single flight leg without overbooking and cancelations. The flight has capacity C. The booking horizon is divided into *time steps* $t = 0, \dots, T$, where T is departure (e.g., if the time steps represent days to departure for one year ahead, then $T = 365$). The time step denotes the left end point of the *time interval* $[t, t+1[, t = 0, \dots, T-1$. We assume the fare structure to be fenceless (that is, all classes in the fare structure have identical restrictions and all customers will buy-down to the lowest available class) with m equidistant fare levels (price points) $f_0 < \cdots < f_{m-1}$ in increasing order. This assumption on the fare structure is introduced purely for simplicity in the analysis; the methodology can be extended to other restricted, semi-restricted, or fare family fare structures without loss of generality.

Suppose that demand arrives following a negative exponential demand model $d_\theta(f) = \lambda e^{-\gamma(f/f_0 - 1)}$, where $\theta = (\lambda, \gamma)$, $\lambda > 0, \gamma > 0$ are the demand parameters that represent the demand volume per time step and the willingness-to-pay, respectively. In practice, RMS is unaware of the true demand parameters $\theta = (\lambda, \gamma)$ and instead estimates the parameters $\widehat{\theta} = (\widehat{\lambda}, \widehat{\gamma})$ using historical booking data. This negative exponential demand model has been extensively studied in the literature (e.g., Gallego and van Ryzin 1994; Fiig et al. 2010), but the demand model could in practice take any functional form.

The single flight optimization problem can be represented as a finite time Markov Decision Process (MDP) (Talluri and van Ryzin 2005). Formally, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, r, p)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function which in our problem takes a finite number of values corresponding to multiples of the price points in the fare structure, and $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function. In this setting, we can define the state space as the set of pairs $s_t = (c, t)$ of remaining capacity $c = 0, \dots, C$ and time steps $t = 0, \dots, T$, the action space

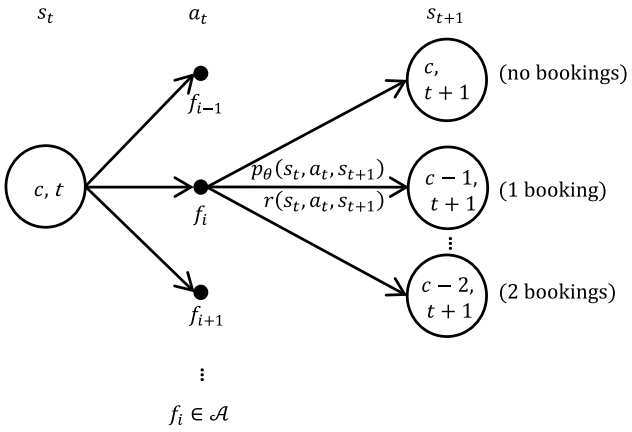


Fig. 1 Single resource MDP back-up diagram. The system is initially in state s_t. The agent chooses an action a_t that causes the system to transition with probability $p_\theta(s_t, a_t, s_{t+1})$ into a reachable subsequent state s_{t+1}, yielding a reward $r(s_t, a_t, s_{t+1})$.

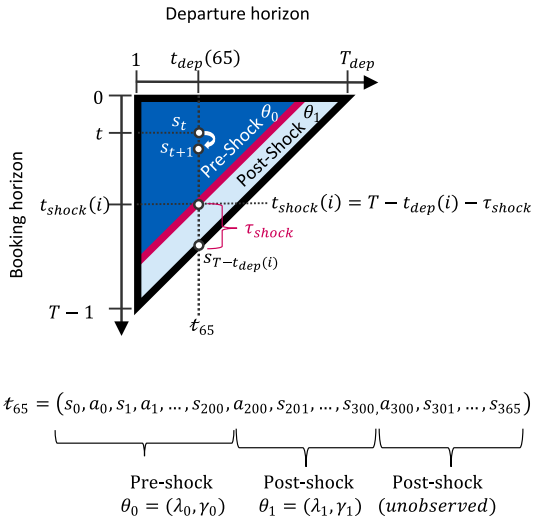
$\mathcal{A} = \{f_0, \dots, f_{m-1}\}$ as the set of possible price points, the state transition probability function as the probability of receiving a specific number of bookings (zero or more) in a time step, and the reward function as the revenue collected after transitioning to a new state.

The state transition process can be represented as follows: at time t, the system is in state $s_t = (c, t)$, the agent (RMS) selects an action (fare) $a_t \in \mathcal{A}$ according to a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$. The system enters into a subsequent state s_{t+1} with probability $p_t = p_\theta(s_t, a_t, s_{t+1})$, where we use the index θ to indicate that the transition probability depends on the demand parameters. The environment then returns an immediate reward $r_t = r(s_t, a_t, s_{t+1})$. The process is repeated until termination at $t = T$, as shown in Fig. 1. In the following, we will use an asterisk to denote optimality.

The objective of the agent is to select the optimal policy $\pi^*(s_t)$ that maximizes the expectation of the sum of future rewards until departure, $v^*(s_t) = \mathbb{E}_{\pi^*}\left[\sum_{k=t}^{T-1} r_k\right]$. The optimal policy can be extracted from the *state-action value function* $q^*(s_t, a_t)$ which is the revenue to go from state s_t, given the agent takes an action a_t and then acts following the optimal policy until termination. The Dynamic Program for the state-action value function and its relation to the *value function* $v^*(s_t)$ is given below.

$$q^*(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} p_\theta(s_t, a_t, s_{t+1}) \times \left[r(s_t, a_t, s_{t+1}) + \max_{a_{t+1}} q^*(s_{t+1}, a_{t+1}) \right],$$

where $s_t \in \mathcal{S}, a_t \in \mathcal{A}$, and $t = 0, \dots, T-1$. The optimal state value function $v^*(s_t) = \max_{a_t} q^*(s_t, a_t)$ is given employing the revenue maximizing action $\pi^*(s_t) = \operatorname{argmax}_{a_t} q^*(s_t, a_t)$.



Area	Symbol	Description
Booking horizon	t	Time step, $t \in \{0, \dots, T-1\}$ refers to booking interval $[t, t+1[$, where T is departure.
	T	T represents departure time
	τ_{shock}	Number of time steps since shock occurred
	$t_{shock}(i)$	Time step at which the demand shock occurred for trajectory i.
Departure horizon	$t_{dep}(i)$	Departure day $t_{dep}(i) \in \{1, \dots, T_{dep}\}$ for trajectory i.
	T_{dep}	The departure date furthest away. We assume $T_{dep} = T$
Trajectories	t_i	Trajectory t_i indexed by $i \in \{1, \dots, N\}$
	$\mathcal{T}$	Trajectory set $\mathcal{T} = \{t_1, \dots, t_N\}$
	$\mathcal{T}^{obs}$	Observed trajectory set based on actual flights

Fig. 2 Two-dimensional time aspects of RMS historic database for a given flight. The triangle contains live sales data. Prior to the shock, the data are generated according to the pre-shock parameters, while after the shock (represented by the diagonal line) the data are generated according to the post-shock parameters

Anatomy of a demand shock

We define a trajectory $t = (s_0, a_0, s_1, a_1, \dots, s_t, a_t, s_{t+1}, \dots)$ as the path a flight takes through state-action space. Hence, a trajectory is a sequence of state-action pairs which records the offered prices and the subsequent bookings received in each time step t. Once a trajectory reaches its terminal state s_T, the flight departs. Note that we omit the rewards in our definition of the trajectory, since in our setting there is a deterministic reward of moving from one state s_t to the successor state s_{t+1}: namely, $r_t = (c_t - c_{t+1})a_t$, where $c_t - c_{t+1}$ is the change in remaining capacity.

While a trajectory represents the path of a single flight through state-action space, it is also of interest to consider a collection of multiple trajectories—for example, multiple departure dates of a flight that departs once per day. We define the *departure horizon* $1, \dots, T_{\text{dep}}$ as the set of departure dates (indexed by days to departure) for active flights, where T_{dep} is the furthest active departure date from today's date (e.g., $T_{\text{dep}} = 365$ if flights are available for sale one year out).

In the remainder of the paper, and without loss of generality, we assume for ease of exposition that the booking horizon and departure horizon have the same cardinality ($T = T_{\text{dep}}$), and that the time step is one day.

We define a *trajectory set* $\mathcal{J}$ as a set of one or more trajectories t_i, $i = 1, \dots, N$ where N is the cardinality of the trajectory set. The departure date of trajectory t_i is denoted $t_{\text{dep}}(i)$, where $t_{\text{dep}} : \{1, \dots, N\} \to \{1, \dots, T_{\text{dep}}\}$. This notation allows us to represent any flight schedule or aggregation of multiple flights per departure day in the trajectory set.

The triangle in Fig. 2 illustrates a trajectory set containing a total of T_{dep} future departure dates (e.g., one year). The horizontal axis indexes the departure horizon (departure dates) and the vertical axis indexes the booking horizon (time steps). Each trajectory (active flight) in the trajectory set can be represented as a vertical line in the triangle; one such trajectory t_{65}, representing a flight that departs in 65 days from now, is illustrated in the figure.

We consider a demand shock that simultaneously affects the entire trajectory set as follows: prior to the demand shock, the state transition probabilities for all trajectories are governed by a vector of pre-shock demand parameters $\theta_0 = (\lambda_0, \gamma_0)$, which are assumed to be known to the RMS. This region is shown in the dark blue area on the left side of the triangle marked "pre-shock." After the demand shock, the state transition probabilities for all trajectories are governed by a vector of post-shock demand parameters $\theta_1 = (\lambda_1, \gamma_1)$, which are unknown to the RMS. This region as shown in the light blue area on the right side of the triangle marked "post-shock." The demand shock occurs at the separation between the pre-shock and post-shock regions, which is shown as the red diagonal line.

Let τ_{shock} refer to the number of time steps that have elapsed since the demand shock occurred, which is also unknown to the RMS. Note that τ_{shock} is not a fixed quantity—it increases with time. At the onset of a shock, $\tau_{\text{shock}} = 0$. All active flights are not yet impacted by the shock (Fig. 2, the dark blue area covers fully the triangle). As system time progresses, τ_{shock} increases and more and more of the active flights' history are impacted by the

shock until eventually at $\tau_{\text{shock}} = T_{\text{dep}}$ all of the active flight data are generated under the post-shock parameters (Fig. 2, the light blue area covers fully the triangle).

For trajectory $\mathcal{t}_i$, let $t_{\text{shock}}(i) = T - t_{\text{dep}}(i) - \tau_{\text{shock}}$ be the time step at which the demand shock occurred for that trajectory. The state transition probabilities for trajectory $\mathcal{t}_i$ are as follows:

$$p_\theta(s_t, a_t, s_{t+1}) = \begin{cases} p_{\theta_0}(s_t, a_t, s_{t+1}) & \text{for} \quad t < t_{\text{shock}}(i) \\ p_{\theta_1}(s_t, a_t, s_{t+1}) & \text{for} \quad t \geq t_{\text{shock}}(i) \end{cases}$$

As a concrete example, consider the situation where we have one flight per departure day, $T = T_{\text{dep}} = 365$, and $\tau_{\text{shock}} = 100$. For trajectory $\mathcal{t}_{65}$, $t_{\text{dep}}(65) = 65$ and $t_{\text{shock}}(65) = 200$. Hence, we see that for $t < 200$ we follow the pre-shock parameters, while for $t \geq 200$ we follow the post-shock parameters.

Methods for demand change detection

In this section, we provide a framework for detecting demand shocks. This framework provides a science-based approach to detecting anomalous flights in practice, as we will discuss in the section "Practical implementation of the shock detector."

In this section, we will first assume for the purpose of deriving the statistical properties of the shock detector that the post-shock demand parameters $\theta_1 = (\lambda_1, \gamma_1)$ are known. Subsequently, we discuss the case of unknown post-shock parameters.

Let $\mathcal{J}^{\text{obs}}$ represent an *observed trajectory set*—the set of active flights for which we wish to detect whether a demand shock has occurred. We construct a hypothesis test for the occurrence of a shock as follows:

$$H_0 : \theta = \theta_0 \ \forall i, t$$

$$H_{\tau_{\text{shock}}} : \begin{cases} \theta = \theta_0 \text{ for } t < t_{\text{shock}}(i) & \forall i \\ \theta = \theta_1 \text{ for } t \geq t_{\text{shock}}(i) & \forall i \end{cases}$$

Note that the alternative hypothesis is indexed by τ_{shock}, since the time step $t_{\text{shock}}(i) = T - t_{\text{dep}}(i) - \tau_{\text{shock}}$ at which the shock occurs for each trajectory in $\mathcal{J}^{\text{obs}}$ is dependent on τ_{shock}.

To test $H_{\tau_{shock}}$ against H_0, we compute the log likelihood of obtaining the observed trajectory set under each hypothesis. First, we compute the likelihood $\mathcal{L}_0(\mathcal{t}_i)$ of observing a single trajectory $\mathcal{t}_i$ assuming no shock has occurred while following a deterministic policy $a_t \sim \pi^*(s_t)$:

$$\mathcal{L}_0(\mathcal{t}_i) = \prod_{t=0}^{T-t_{dep}(i)-1} p_{\theta_0}(s_t, a_t, s_{t+1})$$

Analogously, we compute $\mathcal{L}_{shock}(\mathcal{t}_i)$, the likelihood of a trajectory given a shock that occurred τ_{shock} time steps ago. Recall that if there is a shock, the state transition probabilities will be governed by demand parameters θ_0 until time $t_{\text{shock}}(i) - 1 = T - t_{\text{dep}}(i) - \tau_{\text{shock}} - 1$ and demand parameters θ_1 thereafter.

$$\mathcal{L}_{shock}(\mathcal{t}_i) = \prod_{t=0}^{t_{shock}(i)-1} p_{\theta_0}(s_t, a_t, s_{t+1}) \times \prod_{t=t_{shock}(i)}^{T-t_{dep}(i)-1} p_{\theta_1}(s_t, a_t, s_{t+1})$$

We extend the likelihood computations from single trajectories to the observed trajectory set $\mathcal{J}^{\text{obs}}$ by taking the product of likelihood functions over the trajectories in the set:

$$\mathcal{L}_0(\mathcal{J}^{obs}) = \prod_{\mathcal{t} \in \mathcal{J}^{obs}} \mathcal{L}_0(\mathcal{t})$$

and

$$\mathcal{L}_{shock}(\mathcal{J}^{obs}) = \prod_{\mathcal{t} \in \mathcal{J}^{obs}} \mathcal{L}_{shock}(\mathcal{t}),$$

where we have suppressed the departure day index for readability.

We then compute the likelihood ratio test statistic D (*deviance*) by comparing the log likelihood of the observed trajectory set under the null hypothesis and under the alternative hypothesis.

$$\frac{D}{2} = \log \frac{\mathcal{L}_{\text{shock}}(\mathcal{J}^{\text{obs}})}{\mathcal{L}_0(\mathcal{J}^{\text{obs}})} = \ell_{\tau_{\text{shock}}}(\mathcal{J}^{\text{obs}}) - \ell_0(\mathcal{J}^{\text{obs}}),$$

where $\ell(\mathcal{J}^{\text{obs}}) = \log \mathcal{L}(\mathcal{J}^{\text{obs}})$ denotes the corresponding log-likelihood function. Large values of the deviance indicate a poor model description and thus lead to a rejection of H_0. To determine the critical region for the deviance, we need to determine its sampling distribution. This can be done in closed form for a simple MDP, as we show below.

Closed form expressions for the log-likelihood functions

Consider again the negative exponential model $d_\theta(f)$, with pre-shock and post-shock parameters $\theta_0 = (\lambda_0, \gamma_0)$ and $\theta_1 = (\lambda_1, \gamma_1)$, respectively. Assume no capacity constraining and that the timesteps are sufficiently close that we can ignore multiple bookings in a time step. These assumptions simplify the mathematical derivation, and we will discuss in the section "Shock detection in the general case—parametric

bootstrapping" how the framework can be extended in the general case.

Given the simplifying assumptions, we can compute closed form expressions for the log-likelihood functions.

The optimal pre-shock policy in all states is $\pi^*_{\theta_0} = f_0/\gamma_0$. Let $p_0 = d_{\theta_0}\left(\pi^*_{\theta_0}\right)$ and $p_1 = d_{\theta_1}\left(\pi^*_{\theta_0}\right)$. The transition probabilities under the pre- and post-shock environments are thus: pre-shock: $p_{\theta_0} = p_0$ (booking) and $p_{\theta_0} = 1 - p_0$ (no booking); post-shock: $p_{\theta_1} = p_1$ (booking) and $p_{\theta_1} = 1 - p_1$ (no booking). Let n_0, n_1 denote the number of state transitions in the pre-shock and post-shock regions of $\mathcal{J}^{\text{obs}}$, respectively, and let x_0, x_1 denote the bookings in the pre-shock and post-shock region of $\mathcal{J}^{\text{obs}}$, respectively.

The likelihood function $\mathcal{L}_{\text{shock}}\left(\mathcal{J}^{\text{obs}}\right)$ and the corresponding log-likelihood function and its sample distribution can be computed:

$$\mathcal{L}_{\text{shock}}\left(\mathcal{J}^{\text{obs}}\right) = p_0^{x_0}\left(1-p_0\right)^{n_0-x_0} p_1^{x_1}\left(1-p_1\right)^{n_1-x_1}$$

$$\ell_{\text{shock}}\left(\mathcal{J}^{\text{obs}}\right) = \sum_{i=0}^{1} x_i \log\left(\frac{p_i}{1-p_i}\right) + n_i \log(1-p_i)$$

Note that this expression for the log-likelihood includes the no-shock log-likelihood as a special case. Indeed, the no-shock case is obtained by setting the post-shock and pre-shock booking probability equal $p_1 = p_0$.

Until now we have considered the likelihood of observing a specific set of trajectories. Now we change point of view and focus on the sample distribution of the log-likelihood function. Thus, we consider the log-likelihood function depending on the random variables X_0 and X_1.

$$\ell_{\text{shock}}\left(\mathcal{J}^{\text{obs}}\right) = \sum_{i=0}^{1} X_i \log\left(\frac{p_i}{1-p_i}\right) + n_i \log(1-p_i)$$

Using $X_i \sim \text{Bin}(n_i, p_i)$, $i = 0, 1$ we can approximate these distributions with Normal distributions. Observe now that the expression for the log-likelihood function is linear of the form $aX_0 + bX_1 + c$, where X_0 and X_1 are normally distributed and a, b, c are constants. Hence, the log-likelihood function also becomes normally distributed $\ell_{\text{shock}}\left(\mathcal{J}^{\text{obs}}\right) \sim N\left(\mu_{\text{shock}}, \sigma^2_{\text{shock}}\right)$ where

$$\mu_{\text{shock}} = \sum_{i=0}^{1} n_i p_i \log\left(\frac{p_i}{1-p_i}\right) + n_i \log(1-p_i)$$

$$\sigma^2_{\text{shock}} = \sum_{i=0}^{1} n_i p_i \left(1-p_i\right)\left(\log\left(\frac{p_i}{1-p_i}\right)\right)^2$$

Similarly, we obtain $\ell_0\left(\mathcal{J}^{\text{obs}}\right) \sim N\left(\mu_0, \sigma_0^2\right)$, with

$$\mu_0 = (n_0 + n_1)\left[p_0 \log\left(\frac{p_0}{1-p_0}\right) + \log(1-p_0)\right]$$

$$\sigma_0^2 = (n_0 + n_1) p_0 \left(1-p_0\right)\left(\log\left(\frac{p_0}{1-p_0}\right)\right)^2$$

Let m be the number of flights per departure day in $\mathcal{J}^{\text{obs}}$ (which can also be fractional, e.g., every second day). The sample size becomes $N = mT$, and the total number of state transitions $n_0 + n_1 = NT/2$, which is the area of the triangle of active flights. Analogously, the number of state transitions in the pre-shock area becomes $n_0 = \frac{m\left(T-\tau_{\text{shock}}\right)^2}{2} \approx \frac{NT}{2} - N\tau_{\text{shock}}$, and $n_1 \approx N\tau_{\text{shock}}$ for $\tau_{\text{shock}} \ll T$. Note that n_0, n_1 and N are integers, thus the relations above are approximations.

$$\mu_{\text{shock}} - \mu_0 \approx N\tau_{\text{shock}} \times \left[\log\left(\frac{1-p_1}{1-p_0}\right) + p_1 \log\left(\frac{p_1}{1-p_1}\right) - p_0 \log\left(\frac{p_0}{1-p_0}\right)\right]$$

Thus, the separation between the two Normal distributions grows proportionally to sample size and time since shock.

$$\sigma^2_{\text{shock}} \approx \sigma_0^2 = \frac{NT}{2} p_0 \left(1-p_0\right)\left[\log\left(\frac{p_0}{1-p_0}\right)\right]^2$$

That the variance $\sigma^2_{\text{shock}} \approx \sigma_0^2$ follows from $n_1 \ll n_0$ (assuming $\tau_{\text{shock}} \ll T$, which holds for shocks that are detectable within a reasonable time frame of at most a few months). Further, it follows from the above expression that the variance grows proportionally with sample size.

Closed form expressions for the statistical power

In this section, we continue with the simplified analytical model and compute the statistical power of the shock detector. Let $\phi(x)$ and $\Phi(x)$ denote the pdf and cdf of the standard Normal distribution. Assume $\mu_{\text{shock}} > \mu_0$ (similar expressions can be obtained for $\mu_{\text{shock}} < \mu_0$). We choose a significance level α (one-sided). Let $z_{1-\alpha} = \Phi^{-1}(1-\alpha)$ denote the quantile. The acceptance region of the one-sided hypothesis test is shown as the $(1-\alpha)100\%$ confidence interval and the rejection region is correspondingly shown as the complement, as illustrated in Fig. 3.

The power Pow of the statistical test is the probability of rejection of H_0. Hence, the power is computed as the shaded area, c.f. Figure 3 panel (a).

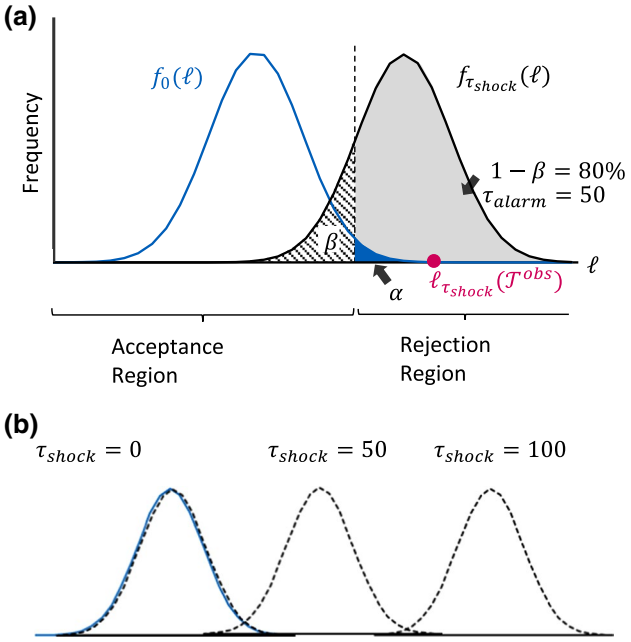


Fig. 3 Log-likelihood distributions generated for the null hypothesis $f_0(\ell)$ and the alternative hypothesis $f_{\text{shock}}(\ell)$ at $\tau_{\text{shock}} = 50$. Parameters: $\theta_0 = (\frac{70}{365}, 0.447)$ and $\theta_1 = (\frac{70}{365}, 0.555)$

$$\begin{aligned}\text{Pow} &= \Pr(\ell_{\text{shock}}(\mathcal{J}^{\text{obs}}) > \mu_0 + z_{1-\alpha}\sigma_0) \\ &= \int_{\mu_0 + z_{1-\alpha}\sigma_0}^{\infty} \frac{1}{\sigma_{\text{shock}}} \phi\left(\frac{x - \mu_{\text{shock}}}{\sigma_{\text{shock}}}\right) dx \\ &= 1 - \Phi\left(\frac{\mu_0 - \mu_{\text{shock}} + z_{1-\alpha}\sigma_0}{\sigma_{\text{shock}}}\right)\end{aligned}$$

Finally, define the *alarm time* τ_{alarm} as the expected number of days after a shock at which the statistical power of the detector reaches a given threshold $1 - \beta$.

$$\tau_{\text{alarm}} = \text{argmin}_{\tau_{\text{shock}}}[\text{Pow} \geq 1 - \beta]$$

To understand how the shock evolves to affect the data over time, see Fig. 3 panel (b), which illustrates how the shock distribution (black dashed curve) propagates from left to right with increasing τ_{shock}. Also shown is the no-shock distribution (blue curve), which remains static.

At the onset of a shock, $\tau_{\text{shock}} = 0$, $\mu_{\text{shock}} = \mu_0$ and $\sigma_{\text{shock}} = \sigma_0$, and the statistical power $\text{Pow} = \alpha$. Detection at this point will be entirely random. At the other extreme, $\tau_{\text{shock}} = T_{\text{dep}}$, the shock distribution is shifted to the right and the statistical power $\text{Pow} \approx 1$, which means all shocks will eventually be identified successfully.

In between these two extremes (for example, at $\tau_{\text{shock}} = 50$), the shock distribution has been shifted such that the statistical power reaches the predefined threshold: $\text{Pow} = 1 - \beta$. This defines the alarm time.

We can obtain an analytical expression for the alarm time by inverting the power equation employing the relationship $1 - \Phi(x) = \Phi(-x)$ and inserting the analytical expressions for $\mu_{\text{shock}} - \mu_0, \sigma_{\text{shock}}, \sigma_0$:

$$\tau_{\text{alarm}} \approx g(p_0, p_1) \frac{z_{1-\alpha} + \Phi^{-1}(1-\beta)}{\sqrt{N/T}}$$

$$\begin{aligned}g(p_0, p_1) &= \frac{\sqrt{p_0(1-p_0)\left[\log\left(\frac{p_0}{1-p_0}\right)\right]^2/2}}{\left|\log\left(\frac{1-p_1}{1-p_0}\right) + p_1\log\left(\frac{p_1}{1-p_1}\right) - p_0\log\left(\frac{p_0}{1-p_0}\right)\right|} \\ &\approx \frac{\sqrt{p_0(1-p_0)/2}}{|p_1 - p_0|} + O(p_1 - p_0)^0\end{aligned}$$

where $g(p_0, p_1)$ is a *shock scenario-specific factor* which depends on the pre-shock booking probability p_0 and the post-shock booking probability p_1.[1] The approximate expression is obtained by a series expansion around p_0 to first order.

Thus, we have computed a closed form relationship between alarm time (τ_{alarm}), shock scenario-specific factor (g), significance level (α), statistical power ($1 - \beta$), and sample size (N). Below we will discuss their interdependence:

- **Shock scenario-specific factor** (g) From the analytical expression, observe that $g(p_0, p_1)$ is a hyperbola where the alarm time is inversely proportional to shock size expressed as $|p_1 - p_0|$. This makes intuitive sense. Small shocks will have a high $g(p_0, p_1)$ factor and be hard to detect, while large shocks will have a low $g(p_0, p_1)$ factor and be easy to detect.
- **Significance level** (α) The significance level α expresses the probability of incorrectly flagging a shock when in fact no shock has occurred (false positive, so-called Type I error). For example, if we choose $\alpha = 0.05$, then 5% of flagged shocks will be false alerts. Decreasing the value of α will result in less shocks being flagged, while at the same time also leading to less false positives.
- **Statistical power** ($1-\beta$) The statistical power expresses the probability of correctly identifying a shock that has occurred. If we choose e.g., $(1 - \beta) = 80\%$, it implies that given that there has been a shock, we correctly identify 80% of the trajectory sets as having experienced a shock,

[1] The absolute value of the denominator allows us to cover both cases $\mu_{\text{shock}} > \mu_0$ and $\mu_{\text{shock}} < \mu_0$ in the same expression.

while we incorrectly classify the remaining 20% as no shock (so-called Type II error). All else equal, increasing the power results in an increase in alarm time.

- **Sample size** (N) The alarm time scales with the inverse square root of the sample size, which can be used to set the appropriate level of aggregation. Increasing the sample size leads to faster detection but at the same time loses granularity. Therefore, it could be an option to configure multiple shock detectors (possibly overlapping) at various sample sizes and significance levels to detect shocks of different sizes and granularities. For example, the detectors could focus on lower-impact shocks that affect a wide range of markets, or larger-impact shocks affecting a single flight or a single market.
- **Alarm time** (τ_{alarm}) The alarm time is the expected time delay before a shock is detected at a specified statistical power. The alarm time depends on the factors discussed and can, for a given shock scenario, be controlled by setting appropriate values for significance level, statistical power, and sample size.

End-to-end shock detection example

In this example, we describe how each of the properties above can be calculated for a specific demand shock. We consider a market AAA-BBB where we have 1 flight per day for one year out, and a one-year booking horizon. Demand arrives following a negative exponential demand model $d_\theta(f) = \lambda e^{-\gamma(f/f_0-1)}$, with pre-shock parameters $\theta_0 = (\lambda_0, \gamma_0) = (\frac{70}{365}, 0.447)$. We will consider the case where a shock has occurred $\tau_{\text{shock}} = 50$ days ago with post-shock parameters $\theta_1 = (\lambda_1, \gamma_1) = (\frac{56}{365}, 0.447)$, corresponding to a negative shock in the arrival rate of 20%. Using the notation from above, we can summarize:

Known to RMS:

- Sample size: $N = 365$
- Pre-shock: $\theta_0 = (70/365, 0.447) \Rightarrow p_0 = 0.1121$
- Pax expected (no-shock): $NTp_0/2 = 7467$

Unknown to RMS:

- Time since shock: $\tau_{\text{shock}} = 50$
- Post-shock: $\theta_1 = (56/365, 0.447) \Rightarrow p_1 = 0.0897$
- Number of state transitions in post-shock region: $n_1 = N\tau_{\text{shock}} = 20440$
- Number of state transitions in pre-shock region: $n_0 = \frac{NT}{2} - n_1 \approx 46172$

Generation of the trajectory set $\mathcal{J}^{obs}$:

Let $Z_i(t)$ be the number of bookings for trajectory t_i at time t. Then $Z_i(t)$ is Bernoulli distributed with $Z_i(t) \sim B(p_0)$ for $0 \le t < t_{\text{shock}}(i)$ and $Z_i(t) \sim B(\hat{p}_1)$ for $t_{\text{shock}}(i) \le t < T - i$, where $t_{\text{shock}}(i) = T - t_{\text{dep}}(i) - \tau_{\text{shock}}$ as before denotes the separation in time between the pre-shock and post-shock region. Given estimates of post-shock parameters and time since shock $\hat{\tau}_{\text{shock}}$, we compute the number of state transitions $n_1 = N\hat{\tau}_{\text{shock}}$, $n_0 = \frac{NT}{2} - N\hat{\tau}_{\text{shock}}$ and aggregate the number of bookings in the two regions (which depend on the estimates):

$$x_0 = \sum_{i=1}^{T} \sum_{t=0}^{T-i-\hat{\tau}_{\text{shock}}-1} z_i(t),\ x_1 = \sum_{i=1}^{T} \sum_{t=T-i-\hat{\tau}_{\text{shock}}}^{T-i-1} z_i(t)$$

Maximum likelihood estimation:

Inserting these values into $\ell_{\text{shock}}(\mathcal{J}^{\text{obs}})$ provides the maximum likelihood estimates (MLE):

$$\hat{\tau}_{\text{shock}}^{MLE}, \hat{\theta}_1^{MLE} = \text{argmax}_{\hat{\tau}_{\text{shock}}, \theta_1} \left(\ell_{\text{shock}} \left(\mathcal{J}^{\text{obs}} \right) \right)$$

We assume that the shock has only affected the demand volume (i.e., $\gamma_1 = \gamma_0$), the reason being that we are unable to estimate the price elasticity parameter because we generated all data with constant price, corresponding to the optimal price without capacity constraining, and for simplicity, we drop the "MLE" superscript.

Observations and estimations:

- Observed total bookings: $x_0 + x_1 = 7146$
- Current impact: $2(x_0 + x_1)/(p_0 NT) - 1 = -4.3\%$
- MLE post-shock: $\hat{\theta}_1 = (57.05/365, 0.447) \Rightarrow \hat{p}_1 = 0.0914$
- MLE volume impact: $(57.05 - 70)/70 = -18.5\%$
- MLE time since shock: $\hat{\tau}_{\text{shock}} = 56$

Shock detection computations:

- Shock specific factor: $g(p_0, \hat{p}_1) \approx 10.76$
- p-value $< 10^{-4}$ (probability of no shock—Type I error)
- Power: $(1 - \beta) = \Phi\left(\hat{\tau}_{\text{shock}} \sqrt{N/T} / g(p_0, \hat{p}_1) - z_{1-\alpha} \right) = 99.9\%$ for $\alpha = 0.01$
- Alarm time: $\tau_{\text{alarm}} = 34$ days (for $\alpha = 0.01, 1 - \beta = 0.8$)

To summarize, applying the shock detection framework, we have estimated that a demand shock occurred $\hat{\tau}_{\text{shock}} = 56$ days ago, causing a volume impact of -18.5% compared to the pre-shock parameters. Since the shock has not fully propagated across all days to departure, we observe only a -4.3% reduction in bookings compared to expectations. This information allows the airline analyst to proactively adjust

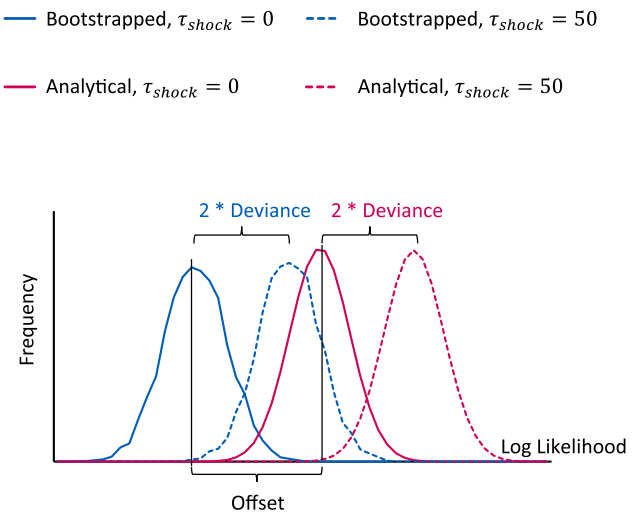


Fig. 4 Log-likelihood sample distributions for the analytical model and parametric bootstrapping shock detectors for $\tau_{\text{shock}} = 0$ and $\tau_{\text{shock}} = 50$. Parameters: $\theta_0 = (\frac{70}{365}, 0.447)$ and $\theta_1 = (\frac{70}{365}, 0.555)$

the demand forecast in line with the customer behavior after the demand shock.

Finally, to compare with methods currently used in the industry, we compute the time to shock detection if we were to alert flights individually. Using the same pre-shock and post-shock parameters and inserting $N = 1$, it would take one year ($\tau_{\text{alarm}} = 365$ days) to reach a power of $1 - \beta \approx 0.8$ at $\alpha = 0.1$. If we instead use the entire trajectory set of $N = 365$ flights, as done in the example above, we can detect the shock in $\tau_{\text{alarm}} = 34$ days with the same statistical power at a much stronger significance level ($\alpha = 0.01$), providing both significantly faster time to detection and improved accuracy.

Shock detection in the general case—parametric bootstrapping

The analytical results in the previous sections made two important assumptions: that the state transition probabilities are state independent and that at most one booking occurred at each time step. When the transition probabilities are state dependent (that is, the offered price depends on the state) and multiple arrivals can occur in a time step, we can no longer express the sample distribution in closed form. In this case, we use parametric bootstrapping (Efron and Tibshirani 1986) to construct the log-likelihood sample distribution used in the shock detector.

To perform the bootstrapping, we create an ensemble of independent "virtual copies" $\{\mathcal{J}^1, \ldots, \mathcal{J}^K\}$ of the observed trajectory set $\mathcal{J}^{\text{obs}}$. Each virtual copy matches the dimensions (departure dates, booking horizon, cardinality) of $\mathcal{J}^{\text{obs}}$, and is initialized empty. For each virtual copy $\mathcal{J}^k$, trajectories are then generated by following a random walk through state-action space assuming the *a priori* (pre-shock) demand parameters and the RMS policy $\pi^*_{\theta_0}$, which is constructed from solving the Dynamic Program as described previously.

We then compute observations $\ell_k = \ell_0(\mathcal{J}^k)$ of the log-likelihood function for each trajectory set $\mathcal{J}^k \in \{\mathcal{J}^1, \ldots, \mathcal{J}^K\}$ using the state transition probabilities corresponding to the no-shock case. Let $f_0(\ell)$ denote the "empirical" probability distribution functions (epdf) obtained from the ensemble (not to be confused with fare levels, that are also denoted by f). Analogously, we compute the epdf for the post-shock demand parameters, which we denote $f_{\text{shock}}(\ell)$. We will return to these distributions below, and in Fig. 4.

Further let $F_0(\ell)$ denote the corresponding empirical cumulative distribution function (ecdf). In this way, the ensemble produces a sample distribution of the log-likelihood function. As before, we determine the acceptance region of the one-sided hypothesis test as the $(1-\alpha)100\%$ confidence interval $]-\infty, F_0^{-1}(1-\alpha)]$ from the sample distribution $F_0(\ell)$.

To perform the hypothesis test, we evaluate the log likelihood, $\ell_0(\mathcal{J}^{\text{obs}})$ of the *observed* trajectory set $\mathcal{J}^{\text{obs}}$ for the active flights. If $\ell_0(\mathcal{J}^{\text{obs}})$ falls outside of the acceptance region, we reject the null hypothesis and flag that a demand shock has occurred.

In the section "Closed form expressions for the log-likelihood functions," we determined closed form expressions for the log-likelihood sample distribution $\ell_0 \sim N(\mu_0, \sigma_0^2)$, and $\ell_{\text{shock}} \sim N(\mu_{\text{shock}}, \sigma_{\text{shock}}^2)$, for the pre- and post-shock distributions, respectively. These normally distributed pdfs can be compared to the corresponding epdfs $f_0(\ell)$ and $f_{\text{shock}}(\ell)$ obtained using parametric bootstrapping, as explained above.

Figure 4 shows one such example of comparing the pdfs (analytical model) and epdfs (parametric bootstrapping), under identical demand parameters at $\tau_{\text{shock}} = 0$ and $\tau_{\text{shock}} = 50$. Note that the distributions from the analytical model are shifted with respect to the empirical distributions due to the differences in demand assumptions. This offset implies that we cannot rely solely on the analytical model to determine the critical regions.

Importantly, however, the offset between the pdf and the epdf appears to be constant independent of demand parameters in the considered scenario and with the assumptions made so far. This also means that the *deviance* metric that measures the difference between the log-likelihood functions under the pre-shock and post-shock demand parameters (i.e., the difference between the two epdfs) can accurately be approximated using the analytical model (i.e., the difference between the two pdfs). This implies that even though we

cannot use the analytical model to construct explicit values for the critical regions, we can trust the analytical closed form expression for the power (because it relies only on the difference between the no-shock and shock distributions), without having to resort to an entirely simulation-based approach.

Simulation studies

In this section, we validate the theoretical properties of the shock detector. We construct a simulation environment in which bookings arrive according to a specified pre-shock demand model until a demand shock occurs at a predetermined time. Both the shock impact and the time since shock are unknown to the detector. We then measure the number of days necessary for the detector to identify the shock. For each scenario, the shock detection performance computed using the analytical model is included for reference.

In our simulation environment, we retain the fenceless fare structure, capacity, and negative exponential demand model from previous sections while representing the willingness-to-pay parameter γ in terms of the more widely used business term $frat5 = 1 + \log(2)/\gamma$. The $frat5$ represents the ratio of the lowest fare at which half of the demand will buy up (Belobaba and Hopperstad 2004). The underlying pre-shock demand parameters are $\theta_0 = (\lambda_0, frat5_0) = \left(\frac{70}{365}, 2.55\right)$, which under the optimal policy π_0^* produces an expected load factor of 82%.

We evaluate the performance of the detector under various positive and negative shocks in demand volume and willingness-to-pay, which are shown in Table 2. We use a trajectory set $\mathcal{J}^{\text{obs}}$ consisting of $N = 365$ flights. We set the confidence level to 96% ($\alpha = 0.04$). Then, we measure the statistical power of the shock detector across 2000 independent simulations. The results are shown in Fig. 5, where the alarm time τ_{alarm} at which the statistical power reaches 80% is marked for each scenario.

Table 2 Demand shock scenarios evaluated in the simulation studies

Scenario	Demand shock parameters		
	Shock type	λ_1	$frat5_1$
(a)	Negative	0.192	1.8
(b)	Positive	0.192	3.3
(c)	Negative	0.123	2.55
(d)	Positive	0.260	2.55
(e)	Negative	0.192	1.8
(f)	Both	0.123 to 0.260 (10 steps)	2.55
(g)	Both	0.192	1.8 to 3.3 (10 steps)

Overall, we see that the statistical power exhibits an S-shape evolution from a power level of $\text{Pow}(\tau_{\text{shock}} = 0) = \alpha$ to $\text{Pow}(\tau_{\text{shock}} = 100) \approx 1$. This behavior can intuitively be observed by looking at the overlap between the underlying no-shock and shock distributions at various τ_{shock} shown for scenario (c).

As predicted in the section "Shock detection in the general case—parametric bootstrapping" the analytical closed form expression for the statistical power provides a very accurate approximation of the power of the detector, even though the demand assumptions differ between the two models. We also observe that negative shocks in demand volume and willingness-to-pay can be detected more quickly than positive shocks. For example, in scenario (a), a negative shock in willingness-to-pay can be detected in about two weeks with 80% power, while an equivalent positive shock in scenario (b) takes one month to detect. This is because a negative shock in willingness-to-pay will result in a sudden loss of bookings received at higher price points, which is easier for the detector to identify compared to the slight increase of bookings associated with a positive shock in willingness-to-pay.

Next, in Fig. 6, we investigate the sensitivity of the shock detector with respect to the number of flights in the trajectory set. For simplicity, we assume that fight schedules are equally spaced across the year (for example, a departure once a week, twice a week, etc.).

Adding more flights dramatically improves performance of the detector, and the scaling law $\tau_{\text{alarm}} \propto 1/\sqrt{N}$ proven by the analytical model can clearly be seen in Fig. 6. Note again that the analytical model provides an accurate description of the power of the detector as a function of sample size, although the analytical model predicts a slightly faster shock detection than is realized by the detector.

For our last experiment, we study how τ_{alarm} varies with the intensity of the shock. We evaluate five positive and five negative shocks with different intensities for both volume and willingness-to-pay. Figure 7 presents the findings.

The pre-shock settings are shown at the center of each chart's horizontal axis. The region to the right of the center line shows positive shocks of increasing intensity, and the region to the left shows negative shocks of increasing intensity. The alarm time τ_{alarm} is shown in the left axis of each panel. Again, the analytical model closely approximates the performance of the detector in all shock scenarios. Furthermore, as shocks get more extreme, the less time it takes to detect them.

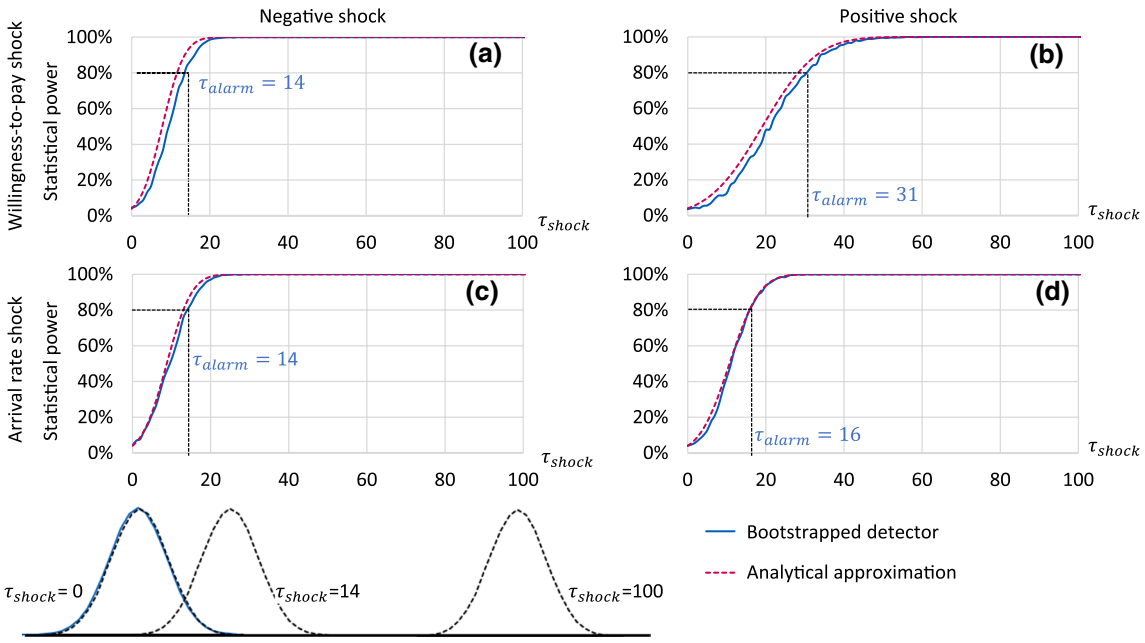


Fig. 5 Statistical power as a function of days since shock τ_{shock} for the bootstrapped shock detector and analytical model for scenarios (a)–(d) from Table 2

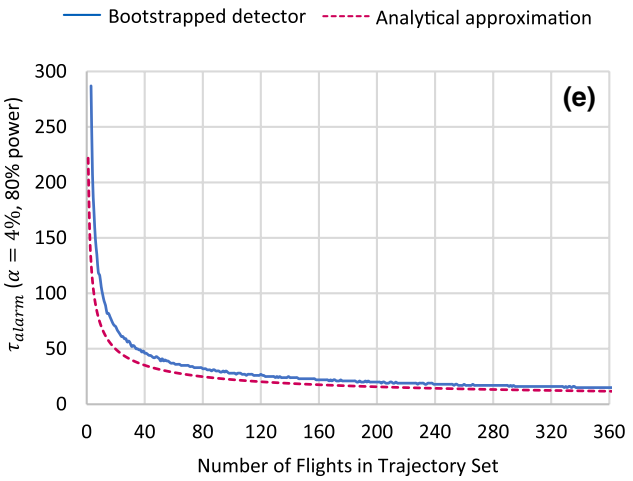


Fig. 6 Average days to detection, τ_{alarm} estimated from when the statistical power of the test reaches 80%: $1 - \beta(\tau_{\text{alarm}}) = 0.8$ as a function of number of flights in the trajectory set for scenario (e)

Practical implementation of the shock detector

In this section, we discuss how the theoretical shock detector mechanism described in previous sections could be used in practice via an alert center application in an RMS. Figure 8 shows how the shock detector can be integrated into a typical RMS process flow of demand forecasting, optimization and availability control.

Following the flow in Fig. 8, incoming bookings and RMS controls (i.e., offered prices for active flights) are stored in the active flight/flow database. Note that RMS controls are stored even if no bookings are observed. Once a flight departs, it moves from the live flight/flow database to the departed flight/flow database. These data are used by the forecasting module, which estimates the forecast parameters and computes the demand forecasts for all possible control policies (Fiig et al. 2014). Finally, the optimization module calculates bid prices used in the availability calculation to determine the RMS controls.

The input data for the shock detection algorithm—the trajectory set $\mathcal{J}^{\text{obs}}$—can be compiled from the active flight database and the forecasting module. This corresponds to the transition probabilities $p_\theta(s_t, a_t, s_{t+1})$ that are used to compute the observed value of the log-likelihood function assuming no shock $\ell_0(\mathcal{J}^{\text{obs}})$. As discussed in the section "Shock detection in the general case—parametric bootstrapping," we can then employ parametric bootstrapping to construct the empirical distribution function $F_0(\ell)$, which is used to construct acceptance and rejection regions used to detect a shock affecting $\mathcal{J}^{\text{obs}}$.

The data produced from the demand shock detection mechanism could be used as part of an alert center dashboard to draw the analyst's attention to aggregations of flights that

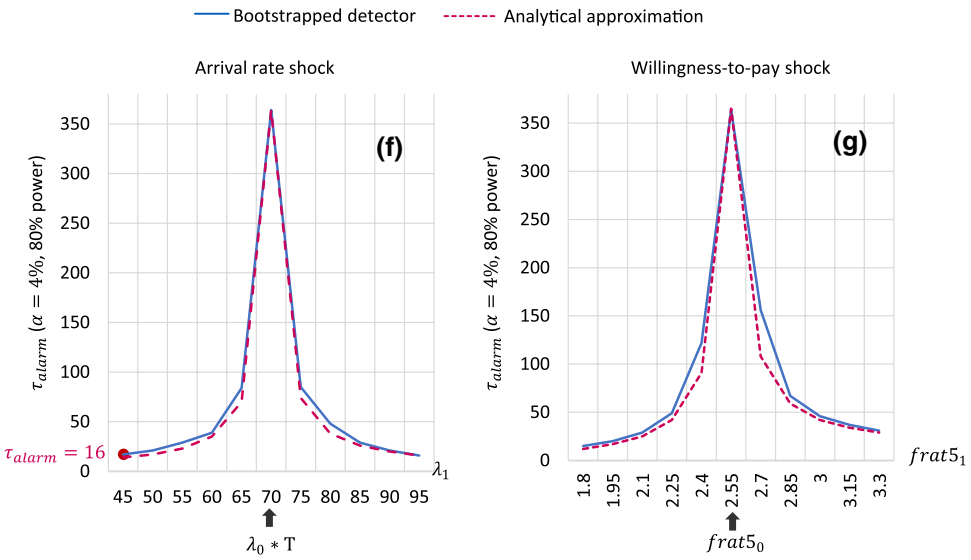


Fig. 7 Average days to detection, τ_{alarm} estimated from when the statistical power of the test reaches 80%: $1 - \beta(\tau_{\text{alarm}}) = 0.8$ as a function of shock intensity for scenarios (f) and (g)

have experienced a demand shock. The alert center would operate as an offline process that daily or weekly scans the airline's entire network to provide lists of O&D markets, regions, or other aggregations of flights that experienced a shock. The shock detector can be configured through a variety of criteria—for example, the shock impact or Type I and Type II error levels—to alert users to shocks with significant business relevance while limiting the number of false alerts.

The shock detector can also generate additional KPIs to display along with the list of markets or other aggregations that have experienced a shock. In Fig. 8, we have illustrated some of this information, such as the *demand shock impact*, *time since shock*, *expected bookings, observed bookings*, *p-value*, and *statistical power*, following the End-to-end shock detection example presented previously.

Such information would allow airline analysts to sort and rank shocks in order of recency or revenue impact, allowing analysts to prioritize their shock investigation and decide on corrective actions. In this example where the shock detector has identified an 18% reduction in demand volume, the analyst may choose to apply an intervention to lower the demand forecast for future flights in the market, to ensure that the RMS does not overprotect capacity.

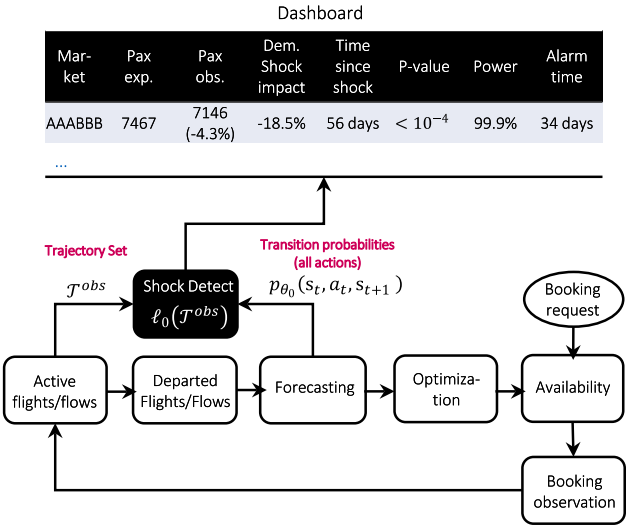


Fig. 8 Implementation of shock detection in practice

Extensions and future work

In this paper, we have made several simplifying assumptions for ease of exposition. For example, we have formulated the detection methodology for a single flights or multiple flights, but not across O&D markets. However, the concepts and methodologies generalize easily to the O&D market level. The transition probabilities at the traffic flow level are one of the byproducts of a conditional demand forecast (Fiig et al. 2014) which the airline would already have computed in order to perform the network optimization. The offered prices (actions) for each traffic flow are also available from the historical booking data for the trajectory. The shock detector would then simply use the relevant transition probabilities and offered prices associated with each traffic flow when computing the log likelihood of a trajectory, and when generating the trajectories used in parameter bootstrapping.

Additionally, there are several avenues of future work that could be explored to improve the performance of the shock detector. First, the speed of shock detection could be further

improved from the method presented here. One possibility would be to apply a weight decay to the state transition probabilities $p_\theta(s_t, a_t, s_{t+1})$ such that more recent observations are weighted more in the calculation of the log-likelihood function. Alternatively, the shock detection mechanisms could focus on transitions from only the most recent sales dates, as opposed to the set of all transitions in the trajectory set. Such an idea has recently been successfully applied to adapt the RMS forecast to widespread demand changes caused by the COVID-19 pandemic (Fiig et al. 2020).

Finally, we have shown that the analytical model well represents the performance of the bootstrapped shock detector in a case with a simple demand model. It is also worth investigating whether these properties continue to hold in more complex cases, including across multiple fare families and time-dependent demand volumes or willingness-to-pay.

Conclusions

Sudden, unobservable changes in customer behavior can create significant demand forecast error, which can be costly to airlines. Identifying these demand shocks can be a time-consuming process for airline analysts, who often set arbitrary performance thresholds to flag flights or markets with abnormal booking behavior. Analysts often struggle to determine appropriate thresholds to accurately identify meaningful demand shocks while limiting the amount of false positive alerts.

In this paper, we introduced a science-based framework for demand shock detection. Our shock detector computes the log likelihood that a set of active flights followed the observed trajectories in state-action space, assuming that demand was governed by the *pre-shock* forecast parameters. If the log likelihood of the trajectory set falls outside an acceptance region, this indicates a poor model description by the forecast parameters and thus leads to the conclusion that a shock has occurred.

We demonstrated how with simple state-independent transition probabilities, we could analytically compute a closed form relationship between alarm time (τ_{alarm}), shock scenario-specific factor (g), significance level (α), statistical power ($1 - \beta$), and sample size (N). For more complex environments with state-dependent transition probabilities, we demonstrated how a parametric bootstrapping approach could be used to construct the acceptance region for the shock detector. We found through simulations that the shock detector exhibited the expected statistical properties and was well described by the analytical model.

We described how our framework could be integrated into an RMS and used to display an estimate of shock impact and time since shock to airline analysts in an alert center application, allowing the airline analyst to easily and quickly identify shocks and prioritize them. This limits the burden on airline analysts and allows them to focus their energy on responding to changes in demand, rather than on merely identifying such changes.

Acknowledgements We thank Roger Härdling for sharing his knowledge on statistical testing methodologies.

References

Aviv, Y., and A. Pazgal. 2005. A partially observed markov decision process for dynamic pricing. *Management Science* 51 (9): 1400–1416.

Basseville, M. and I.V. Nikoforov. 1993. Detection of Abrupt Changes. Prentice Hall.

Belobaba, P.P. and C. Hopperstad. 2004. Algorithms for revenue management in unrestricted fare markets. In *Proceedings of the INFORMS section on revenue management*, Cambridge, MA.

Besbes, O., and D. Sauré. 2014. Dynamic pricing strategies in the presence of demand shifts. *Manufacturing & Operations Service Management* 16 (4): 513–528.

Besbes, O., and A. Zeevi. 2011. On the minimax complexity of pricing in a changing environment. *Operations Research* 59 (1): 66–79.

Broder, J., and P. Rusmevichientong. 2012. Dynamic pricing under a general parametric choice model. *Operations Research* 60 (4): 965–980.

den Boer, A.V. 2015. Tracking the market: Dynamic pricing and learning in a changing environment. *European Journal of Operational Research* 247: 914–927.

den Boer, A.V., and N.B. Keskin. 2020. Discontinuous demand functions: Estimation and pricing. *Management Science* 66 (10): 4516–4534.

Efron, B., and R. Tibshirani. 1986. Bootstrap method for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science* 1 (1): 54–77.

Fiig, T., K. Isler, C. Hopperstad, and P. Belobaba. 2010. Optimization of mixed fare structures: Theory and applications. *Journal of Revenue and Pricing Management* 9 (1): 152–170.

Fiig, T., R. Härdling, S. Pölt, and C. Hopperstad. 2014. Demand forecasting and measuring forecast accuracy in general fare structures. *Journal of Revenue and Pricing Management* 13 (6): 413–439.

Fiig, T., L.R. Weatherford, and M.D. Wittman. 2019. Can demand forecast accuracy be linked to airline revenue? *Journal of Revenue and Pricing Management* 18 (4): 291–305.

Fiig, T., M.D. Wittman, L. Andersen, C. Föcker, R. Härdling, T. Tofteby, C. Trescases, and L. Zannier. 2020. Revenue management forecasting in times of change: Addressing the need for speed. Presented at the 2020 AGIFORS annual symposium.

Gallego, G., and G. van Ryzin. 1994. Optimal dynamic pricing of inventories with stochastic demand over finite horizons. *Management Science* 40 (8): 999–1020.

Garivier, A., and E. Moulines. 2011. On upper-confidence bound policies for switching Bandit problems. In *ALT 2011: Algorithmic Learning Theory*, 174–188.

Hadoux, E., A. Beynier, and P. Weng. 2014. Sequential decision-making under non-stationary environments via sequential change-point detection. In *Proceedings of the 2014 learning over multiple contexts (LMCE)*, Nancy, France, 1–13.

Keller, G., and A. Rady. 1999. Optimal experimentation in a changing environment. *The Review of Economic Studies* 66 (3): 475–507.

Keskin, N.B., and A. Zeevi. 2017. Chasing demand: Learning and earning in a changing environment. *Mathematics of Operations Research* 42 (2): 277–307.

Lai, T.L. 1995. Sequential changepoint detection in quality control and dynamical systems. *Journal of the Royal Statistical Society B* 57 (4): 613–644.

Talluri, K.T., and G.J. van Ryzin. 2005. *The theory and practice of revenue management*. New York: Springer.

Vinod, B. 2021. An approach to adaptive robust revenue management with continuous demand management in a COVID-19 era. *Journal of Revenue and Pricing Management* 20 (1): 10–14.

Weatherford, L. 2019. Performance of dynamic user influence strategies in PODS under seasonality and system volatility. *Journal of Revenue and Pricing Management* 18: 2–26.

Giovanni Gatti Pinheiro is a Ph.D. student in Reinforcement Learning applied to Revenue Management in the University of Nice Sophia-Antipolis.

Thomas Fiig is a Director, Chief Scientist at Amadeus, where he is responsible for revenue management strategy and scientific methodologies. He holds a Ph.D. in Theoretical Physics and Mathematics and a BA in Finance from the University of Copenhagen, Denmark. He has published several articles, recently focused on methodologies for origin-and-destination forecasting and optimization of simplified fare structures and dynamic pricing.

Michael D. Wittman is an Expert, Revenue Management Science & Research at Amadeus, where he works on developing new models for forecasting and optimization in revenue management and dynamic pricing. He holds a Ph.D. in Air Transportation Systems from MIT and is the author of a dozen articles published in peer-reviewed academic journals. His work has won awards and recognition from AGIFORS and INFORMS.

Michael Defoin-Platel is a Head of Machine Learning Services at Amadeus. He received a Ph.D. in Artificial Intelligence from the University of Nice. He has been conducting academic research in fundamental and applied AI during 10 years before moving to the industry. He is now responsible for the development of AI in Amadeus products.

Riccardo D. Jadanza is a Head of Artificial Intelligence Research, leader and coordinator of the data scientist "Brain" team at Enerbrain, where he works mainly on studying, developing, and implementing Machine Learning techniques for energy management optimization in HVAC systems in all kinds of buildings. He holds a Ph.D. in Mathematics from the Polytechnic University of Torino and has previously worked in the field of Complex Systems and in the Revenue Management division at Amadeus.

Bibliography

Here are the references in citation order.

[1] Kalyan T. Talluri and Garrett J. Van Ryzin. *The theory and practice of revenue management*. Vol. 1. Springer, 2004 (cited on pages 1, 9, 20, 53).

[2] Thomas Fiig, Umit Cholak, Mathilde Gauchet, and Benjamin Cany. "What is the role of distribution in revenue management?–Past and future". In: *Journal of Revenue and Pricing Management* 14.2 (2015), pp. 127–133 (cited on page 2).

[3] Amine Dadoun, Michael Defoin-Platel, Thomas Fiig, Corinne Landra, et al. "How recommender systems can transform airline offer construction and retailing". In: *Journal of Revenue and Pricing Management* 20.3 (2021), pp. 301–315 (cited on page 2).

[4] Amine Dadoun, Rapahël Troncy, Michael Defoin-Platel, and Gerardo Ayala Solano. "Predicting your next trip: A knowledge graph-based multi-task learning approach for travel destination recommendation". In: *Workshop on Recommenders in Tourism* (2021), pp. 23–38 (cited on page 2).

[5] Thomas Fiig, Roger Härdling, Stefan Pölt, and Craig Hopperstad. "Demand forecasting and measuring forecast accuracy in general fare structures". In: *Journal of Revenue and Pricing Management* 13.6 (2014), pp. 413–439 (cited on pages 3, 96).

[6] William L. Cooper, Tito Homem-de-Mello, and Anton J. Kleywegt. "Models of the spiral-down effect in revenue management". In: *Operations research* 54.5 (2006), pp. 968–987 (cited on page 3).

[7] Thomas Fiig, Karl Isler, Craig Hopperstad, and Peter P. Belobaba. "Optimization of mixed fare structures: Theory and applications". In: *Journal of Revenue and Pricing Management* 9.1 (2010), pp. 152–170 (cited on page 3).

[8] Qiang Meng, Hui Zhao, and Yadong Wang. "Revenue management for container liner shipping services: Critical review and future research directions". In: *Transportation Research* 128 (2019), pp. 280–292 (cited on page 4).

[9] Sheryl E. Kimes and Jeannette Ho. "Implementing Revenue Management in Your Restaurants: A Case Study with Fairmont Raffles Hotels International". In: (2019) (cited on page 4).

[10] Adrien Bouchet, Michael Troilo, and Brian R. Walkup. "Dynamic pricing usage in sports for revenue management". In: *Managerial Finance* (2016) (cited on page 4).

[11] Thomas Fiig, Michael D. Wittman, and Clement Trescases. "Towards a competitor-aware RMS". In: Oct. 2019 (cited on pages 4, 112).

[12] Ravi Kumar, Wei Wang, Ahmed Simrin, Sivarama Krishnan Arunachalam, et al. "Competitive revenue management models with loyal and fully flexible customers". In: *Journal of Revenue and Pricing Management* 20.3 (2021), pp. 256–275 (cited on page 4).

[13] Pranjal Pande. *What are the world's busiest air routes right now?* June 2020. URL: https://simpleflying.com/busiest-air-routes-june-2020/ (cited on page 4).

[14] Guillermo Gallego and Robert Phillips. "Revenue management of flexible products". In: *Manufacturing & Service Operations Management* 6.4 (2004), pp. 321–337 (cited on pages 4, 109).

[15] Andrew McLennan. "Price dispersion and incomplete learning in the long run". In: *Journal of Economic dynamics and control* 7.3 (1984), pp. 331–347 (cited on pages 6, 63).

[16] Ningyuan Chen and Guillermo Gallego. "A primal-dual learning algorithm for personalized dynamic pricing with an inventory constraint". In: *Mathematics of Operations Research* (2022) (cited on pages 6, 63, 65).

[17] Nicolas Bondoux, Anh Quan Nguyen, Thomas Fiig, and Rodrigo Acuna-Agost. "Reinforcement learning applied to airline revenue management". In: *Journal of Revenue and Pricing Management* 19.5 (2020), pp. 332–348 (cited on pages 6, 15, 22, 53, 54, 112, 119).

[18] N. Bora Keskin and Assaf Zeevi. "Chasing demand: Learning and earning in a changing environment". In: *Mathematics of Operations Research* 42.2 (2017), pp. 277–307 (cited on pages 6, 64, 106).

[19] Giovanni Gatti Pinheiro, Michael Defoin-Platel, and Jean-Charles Régin. "Outsmarting human design in airline revenue management". In: *Algorithms* 15.5 (2022), p. 142 (cited on pages 8, 21, 71, 73, 85, 89, 91–94, 101).

[20] Omar Besbes and Assaf Zeevi. "On the (surprising) sufficiency of linear models for dynamic pricing with demand learning". In: *Management Science* 61.4 (2015), pp. 723–739 (cited on page 11).

[21] Guillermo Gallego and Garrett Van Ryzin. "Optimal dynamic pricing of inventories with stochastic demand over finite horizons". In: *Management science* 40.8 (1994), pp. 999–1020 (cited on pages 11, 15, 20, 22, 23).

[22] Thomas Fiig, Larry R. Weatherford, and Michael D. Wittman. "Can demand forecast accuracy be linked to airline revenue?" In: *Journal of Revenue and Pricing Management* 18.4 (2019), pp. 291–305 (cited on pages 12, 96).

[23] Larry R. Weatherford. "The history of forecasting models in revenue management". In: *Journal of Revenue and Pricing Management* 15.3 (2016), pp. 212–221 (cited on pages 12, 58).

[24] Jeffrey P. Newman, Mark E. Ferguson, Laurie A. Garrow, and Timothy L. Jacobs. "Estimation of choice-based models using sales data from a single firm". In: *Manufacturing & Service Operations Management* 16.2 (2014), pp. 184–197 (cited on page 14).

[25] Travis E. Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006 (cited on page 14).

[26] Peter P. Belobaba and Craig Hopperstad. "Algorithms for revenue management in unrestricted fare markets". In: *Meeting of the INFORMS Section on Revenue Management, Massachusetts Institute of Technology, Cambridge, MA*. 2004 (cited on page 20).

[27] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cited on pages 21, 30, 31, 35–44, 51, 52, 74–76, 79, 81, 82, 100, 105, 117).

[28] *IEEE Standard for floating-point arithmetic*. Tech. rep. Piscataway, NJ, USA: IEEE, 2019 (cited on page 25).

[29] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature methods* 17.3 (2020), pp. 261–272 (cited on page 25).

[30] R. Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957 (cited on page 30).

[31] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, et al. "An algorithmic perspective on imitation learning". In: *arXiv:1811.06711* (2018) (cited on page 35).

[32] Elliot A. Ludvig, Richard S. Sutton, and E. James Kehoe. "Stimulus representation and the timing of reward-prediction errors in models of the dopamine system". In: *Neural computation* 20.12 (2008), pp. 3034–3054 (cited on page 35).

[33] Yuji Takahashi, Geoffrey Schoenbaum, and Yael Niv. "Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model". In: *Frontiers in neuroscience* 2 (2008), p. 14 (cited on page 35).

[34] Momchil S. Tomov, Eric Schulz, and Samuel J. Gershman. "Multi-task reinforcement learning in humans". In: *Nature Human Behaviour* 5.6 (2021), pp. 764–773 (cited on page 35).

[35] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489 (cited on pages 36, 47, 51, 75, 114, 119).

[36] Richard S. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44 (cited on page 37).

[37] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989) (cited on page 41).

[38] Johannes Heinrich and David Silver. "Deep reinforcement learning from self-play in imperfect-information games". In: *arXiv:1603.01121* (2016) (cited on page 43).

[39] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354 (cited on pages 43, 51).

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, et al. "Playing atari with deep reinforcement learning". In: *arXiv:1312.5602* (2013) (cited on pages 49, 51).

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533 (cited on page 49).

[42] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, et al. "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature* 602.7897 (2022), pp. 414–419 (cited on pages 52, 119).

[43] Abhuit Gosavi, Naveen Bandla, and Tapas K. Das. "A reinforcement learning approach to airline seat allocation for multiple fare classes with overbooking". In: *Special issue on advances on large-scale optimization for logistics, production and manufacturing systems* (2002) (cited on page 53).

[44] Abhijit Gosavi. "A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis". In: *Machine Learning* 55.1 (2004), pp. 5–29 (cited on page 53).

[45] Ryan J. Lawhead and Abhijit Gosavi. "A bounded actor-critic reinforcement learning algorithm applied to airline revenue management". In: *Engineering Applications of Artificial Intelligence* 82 (2019), pp. 252–262 (cited on page 53).

[46] Alexander Kastius and Rainer Schlosser. "Dynamic pricing under competition using reinforcement learning". In: *Journal of Revenue and Pricing Management* (2021), pp. 1–14 (cited on pages 53, 119).

[47] Syed Arbab Mohd Shihab, Caleb Logemann, Deepak-George Thomas, and Peng Wei. "Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking". In: *arXiv:1902.06824* (2019) (cited on pages 53, 54, 119).

[48] Jon Ham. *Know your worth: valuing new pricing policies with reinforcement learning*. AGIFORS 61st Annual Symposium, 2021. Sept. 2021 (cited on page 53).

[49] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, et al. "Reinforcement learning with unsupervised auxiliary tasks". In: *arXiv:1611.05397* (2016) (cited on pages 55, 117).

[50] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. "Distributional reinforcement learning with quantile regression". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (cited on page 55).

[51] Giovanni Gatti Pinheiro, Nicolas Bondoux, and Alix Lhéritier. *Towards a distributional reinforcement learning approach to revenue management*. SophI.A Summit. Nov. 2021 (cited on page 55).

[52] Bent Hansen. *Report of the Uppsala meeting*. Aug. 1954 (cited on page 63).

[53] Edward R. Hawkins. "Methods of estimating demand". In: *Journal of Marketing* 21.4 (1957), pp. 428–438 (cited on page 63).

[54] Miguel Sousa Lobo and Stephen Boyd. "Pricing and learning with uncertain demand". In: *INFORMS Revenue Management Conference*. Citeseer. 2003 (cited on pages 63, 64).

[55] Meenal Chhabra and Sanmay Das. "Learning the demand curve in posted-price digital goods auctions". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. 2011, pp. 63–70 (cited on page 63).

[56] H. Dharma Kwon, Steven A. Lippman, and Christopher S. Tang. "Optimal markdown pricing strategy with demand learning". In: *Probability in the Engineering and Informational Sciences* 26.1 (2012), pp. 77–104 (cited on page 63).

[57] Omar Besbes and Assaf Zeevi. "On the minimax complexity of pricing in a changing environment". In: *Operations research* 59.1 (2011), pp. 66–79 (cited on page 63).

[58] N. Bora Keskin and Assaf Zeevi. "Dynamic pricing with an unknown demand model: Asymptotically optimal semi-myopic policies". In: *Operations Research* 62.5 (2014), pp. 1142–1167 (cited on pages 63, 64).

[59] Ningyuan Chen and Guillermo Gallego. "Nonparametric pricing analytics with customer covariates". In: *Operations Research* 69.3 (2021), pp. 974–984 (cited on page 63).

[60] Masanao Aoki. "On a dual control approach to the pricing policies of a trading specialist". In: *IFIP Technical Conference on Optimization Techniques*. Springer. 1973, pp. 272–282 (cited on pages 63, 65).

[61] Chee-Yee Chong and David Cheng. "Multistage pricing under uncertain demand". In: *Annals of Economic and Social Measurement, Volume 4, number 2*. NBER, 1975, pp. 311–323 (cited on page 63).

[62] Michael Rothschild. "A two-armed bandit theory of market pricing". In: *Journal of Economic Theory* 9.2 (1974), pp. 185–202 (cited on pages 63, 64).

[63] Omar Besbes and Assaf Zeevi. “Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms”. In: *Operations Research* 57.6 (2009), pp. 1407–1420 (cited on page 64).

[64] Arnoud V. den Boer and Bert Zwart. “Dynamic pricing and learning with finite inventories”. In: *Operations research* 63.4 (2015), pp. 965–978 (cited on page 64).

[65] Kris Johnson Ferreira, David Simchi-Levi, and He Wang. “Online network revenue management using thompson sampling”. In: *Operations research* 66.6 (2018), pp. 1586–1602 (cited on page 64).

[66] Arnoud V. den Boer and Bert Zwart. “Simultaneously learning and optimizing using controlled variance pricing”. In: *Management science* 60.3 (2014), pp. 770–783 (cited on pages 64, 65, 100).

[67] Dina Elreedy, Amir F Atiya, and Samir I Shaheen. “Novel pricing strategies for revenue maximization and demand learning using an exploration-exploitation framework”. In: *Soft Computing* 25.17 (2021), pp. 11711–11733 (cited on pages 64–66, 100, 102).

[68] Ching-Lai Hwang and Abu Syed Md. Masud. “Multiple objective decision making—methods and applications: A state-of-the-art survey”. In: 164 (2012) (cited on page 64).

[69] Yossi Aviv and Amit Pazgal. “Dynamic pricing of short life-cycle products through active learning”. In: *Olin School Business, Washington Univ., St. Louis, MO* (2005) (cited on page 64).

[70] Eric Cope. “Bayesian strategies for dynamic pricing in e-commerce”. In: *Naval Research Logistics (NRL)* 54.3 (2007), pp. 265–281 (cited on page 64).

[71] Cathy H Xia and Parijat Dube. “Dynamic pricing in e-services under demand uncertainty”. In: *Production and Operations Management* 16.6 (2007), pp. 701–712 (cited on page 64).

[72] William R. Thompson. “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples”. In: *Biometrika* 25.3/4 (1933), pp. 285–294 (cited on page 64).

[73] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multiarmed bandit problem”. In: *Machine learning* 47.2 (2002), pp. 235–256 (cited on page 64).

[74] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. “Curiosity-driven exploration by self-supervised prediction”. In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787 (cited on pages 64, 117).

[75] Giovanni Gatti Pinheiro, Michael Defoin-Platel, and Jean-Charles Régin. “Optimizing revenue maximization and demand learning in airline revenue management”. In: *arXiv:2203.11065* (2022) (cited on pages 66, 90).

[76] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, et al. “Successor features for transfer in reinforcement learning”. In: *Advances in neural information processing systems* 30 (2017) (cited on page 71).

[77] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. “Universal value function approximators”. In: *International conference on machine learning* (2015), pp. 1312–1320 (cited on page 71).

[78] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134 (cited on page 72).

[79] Thomas Degris, Martha White, and Richard S. Sutton. "Off-policy actor-critic". In: *arXiv:1205.4839* (2012) (cited on page 74).

[80] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv:1912.06680* (2019) (cited on page 75).

[81] Özgür Şimşek, Simón Algorta, and Amit Kothiyal. "Why most decisions are easy in tetris – and perhaps in other sequential decision problems, as well". In: *International Conference on Machine Learning* (2016), pp. 1757–1765 (cited on page 75).

[82] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999) (cited on page 76).

[83] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256 (cited on page 78).

[84] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv:1506.02438* (2015) (cited on pages 79, 86).

[85] Noe Casas. "Deep deterministic policy gradient for urban traffic light control". In: *arXiv:1703.09035* (2017) (cited on page 80).

[86] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870 (cited on page 80).

[87] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, et al. "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames". In: *arXiv:1911.00357* (2019) (cited on page 80).

[88] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, et al. "Distributed prioritized experience replay". In: *arXiv:1803.00933* (2018) (cited on page 80).

[89] Yi Wan, Abhishek Naik, and Richard S. Sutton. "Learning and planning in average-reward markov decision processes". In: *International Conference on Machine Learning* (2021), pp. 10653–10662 (cited on page 82).

[90] Shangtong Zhang, Yi Wan, Richard S. Sutton, and Shimon Whiteson. "Average-reward off-policy policy evaluation with function approximation". In: *arXiv:2101.02808* (2021) (cited on page 82).

[91] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cited on page 84).

[92] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv:1508.04025* (2015) (cited on page 84).

[93] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495 (cited on page 84).

[94] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014) (cited on page 84).

[95] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, et al. "Proximal policy optimization algorithms". In: *arXiv:1707.06347* (2017) (cited on page 86).

[96] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, et al. "RLlib: Abstractions for distributed reinforcement learning". In: (2018), pp. 3053–3062 (cited on page 86).

[97] Giovanni Gatti Pinheiro, Michael Defoin-Platel, and Jean-Charles Regin. *Talos*. Version 0.1.0. Oct. 2022. URL: https://github.com/GiovanniGatti/talos (cited on page 86).

[98] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. "Learning without state-estimation in partially observable Markovian decision processes". In: *Machine Learning Proceedings* (1994), pp. 284–292 (cited on pages 100, 105).

[99] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. "Reward is enough". In: *Artificial Intelligence* (2021), p. 103535 (cited on pages 101, 120).

[100] Herbert Jaeger. "A short introduction to observable operator models of stochastic processes". In: 1 (1998), pp. 38–43 (cited on page 104).

[101] Michael Thon. "Spectral learning of dequential systems". PhD thesis. Jacobs University Bremen, 2018 (cited on page 104).

[102] George E. Monahan. "State of the art – a survey of partially observable Markov decision processes: theory, models, and algorithms". In: *Management science* 28.1 (1982), pp. 1–16 (cited on page 104).

[103] Giovanni Gatti Pinheiro, Thomas Fiig, Michael D. Wittman, Michael Defoin-Platel, et al. "Demand change detection in airline revenue management". In: *Journal of Revenue and Pricing Management* (2022). (accepted, see Appendix) (cited on page 106).

[104] Juan José Miranda Bront, Isabel Méndez-Diaz, and Gustavo Vulcano. "A column generation algorithm for choice-based network revenue management". In: *Operations research* 57.3 (2009), pp. 769–784 (cited on page 109).

[105] Laurie A. Garrow. *Discrete choice modelling and air travel demand: theory and applications*. Routledge, 2016 (cited on page 110).

[106] Guillermo Gallego, Richard Ratliff, and Sergey Shebalov. "A general attraction model and sales-based linear program for network revenue management under customer choice". In: *Operations Research* 63.1 (2015), pp. 212–232 (cited on page 111).

[107] Richard S. Sutton. *The bitter lesson*. Mar. 2019. URL: http://incompleteideas.net/IncIdeas/BitterLesson.html (cited on page 114).

[108] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359 (cited on page 114).

[109] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, et al. "Stabilizing transformers for reinforcement learning". In: *International Conference on Machine Learning* (2020), pp. 7487–7498 (cited on page 115).

[110] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In: *International Conference on Machine Learning* (2017), pp. 449–458 (cited on page 117).

[111] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, et al. "Feedback control for cassie with deep reinforcement learning". In: *International Conference on Intelligent Robots and Systems* (2018), pp. 1241–1246 (cited on page 119).

[112] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, et al. "Solving rubik's cube with a robot hand". In: *arXiv:1910.07113* (2019) (cited on page 119).

[113] Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, et al. "Autonomous navigation of stratospheric balloons using reinforcement learning". In: *Nature* 588.7836 (2020), pp. 77–82 (cited on page 119).

Special Terms

A
AI Artificial Intelligence. 4
ANN Artificial Neural Network. 47

C
CEP Certainty Equivalent Pricing. 63

D
DP Dynamic Programming. 31

E
EWL Earning while Learning. 7, 61

M
MDP Markov Decision Process. 20
MSE Mean Squared Error. 89

P
POMDP Partially Observable Markov Decision Process. 104

R
RL Reinforcement Learning. 8, 35
RM Revenue Management. 1
RMS Revenue Management System. 2