Titre: Title:	A Data-driven Approach to Revenue Management Problem with Behavioral Considerations
Auteur: Author:	Neda Etebarialamdari
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Etebarialamdari, N. (2019). A Data-driven Approach to Revenue Management Problem with Behavioral Considerations [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4161/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A Data-driven Approach to Revenue Management Problem with Behavioral Considerations

NEDA ETEBARIALAMDARI
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Décembre 2019

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

A Data-driven Approach to Revenue Management Problem with Behavioral Considerations

présentée par **Neda ETEBARIALAMDARI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Louis-Martin ROUSSEAU, président
Miguel F. ANJOS, membre et directeur de recherche
Gilles SAVARD, membre et codirecteur de recherche
Doina PRECUP, membre et codirecteur de recherche
Morad HOSSEINALIFAM, membre
Masoumeh KAZEMI ZANJANI, membre externe

DEDICATION

To my father Ali

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors, Gilles Savard, Doina Precup and Miguel F. Anjos for their constant support, for making me feel welcome in the community, and for their constructive feedback.

It has been a great honor to work under the supervision of Professor Gille Savard for the past few years. His wisdom and valuable advices gave me the opportunity to benefit and learn from every single moment of my PhD journey in both personal and scientific levels.

I was also very lucky to have Professor Doina Precup as my supervisor. Her deep knowledge combined with her humble and understanding manner gave me the chance to learn and study in a peaceful, knowledgeable, and meaningful environment.

Working with Professor Miguel F. Anjos provided me with the opportunity to be in a professional team in which we learned, enjoyed and helped each other to grow.

My sincere thanks also goes to Dr. Fabien Cirinei who provided me the opportunity to join their team at ExPretio Technologies, and for his continuous support and guidance during my two years internship with the revenue management and data science team. I also wish to express my appreciation to all my dear colleagues over there.

I would like to say a big thank you to all my friends and fellow labmates at Montreal Institute for Learning Algorithms (MILA), Groupe d'Études et de Recherche en Analyse des Décisions (GERAD), and McGill Reasoning and Learning Lab (RLLab).

Being part of MILA has been a great opportunity to have numerous fruitful discussions and scientific collaborations with curious minds who never ceased to amaze me by how far a meaningful teamwork can extend and how big of a difference we can make when we have a mutual goal and overlapping interests.

The members of GERAD have contributed immensely to my personal and professional time at Polytechnique Montreal. The group has been a source of friendships as well as good advice and collaboration. I would like to thank every single member of GERAD, particularly, the huge optimization community, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last few years.

I started my PhD journey at RLLab where I got the chance to meet and work with numerous enthusiastic researchers. For me, this was a dream come true and I have nothing but appreciation and gratitude to the past and current members of RLLab with whom I had the chance to spend great amount of valuable time. Without their precious support it would not

be possible to conduct this research at this level of joy and satisfaction.

Finally, a deep thank you to all people in my personal life, particularly my family, who always help me focus on things that truly matter and embrace life.

RÉSUMÉ

Cette thèse porte sur l'étude de diverses méthodes avancées et orientées données pour résoudre la problématique de gestion de revenu plus souvent désigné par "Revenue Management" (RM). Nous nous intéressons à deux sous-problèmes du RM que sont la prévision de demande et le contrôle d'inventaire, poursuivons avec une revue de la littérature suivi d'une synthèse des contributions de la thèse. Nous commençons par une introduction générale sur la métodologie usuelle pour traiter la prédiction de la demande et les politiques optimales de contrôle d'inventaire. Nous présentons dans les trois chapitres suivants, nos travaux sur ces problématiques, chacun correspondant à un article soumis dans une revue internationale. Finalement nous concluons par des remarques sur le travail actuel et une discussion sur les possibles futurs travaux. Nous présentons maintenant brièvement les trois articles.

Dans le premier article, nous nous intéressons à la prévision de la demande pour une importante compagnie ferroviaire. Pour cela, nous explorons diverses approches de prétraitement, apprentissage machine et sélection de caractéristiques des données. Comme cette prévision est utilisée pour différents objectifs, nous travaillons sur deux niveaux d'agrégation différents. La solution devant être industrialisable, nous mettons l'emphase sur la rapidité, la simplicité et la robustesse. Nous combinons alors des méthodes de l'état de l'art avec des techniques innovantes de construction des caractéristiques des données pour arriver à des résultats prometteurs. Bien que nous traitons la prévision de demande pour le domaine ferroviaire, nos résultats s'appliquent également aux autres domaines de transport et à l'hôtellerie.

Dans le second article, nous considérons le problème du contrôle d'inventaire du RM sous comportement d'achat pour le domaine aérien avec une méthode d'apprentissage par renforcement du type "Deep Q-Network" (DQN). Par rapport aux approches traditionnelles en RM, DQN ne dépend pas d'une prévision de la demande pour retourner de bonnes décisions de contrôle. Il fonctionne en utilisant des données historiques et/ou une interaction directe avec les clients. Nous nous concentrons essentiellement sur l'aspect comportemental de notre modèle. Nous entraînons et évaluons notre solution avec des données synthétiques puis la comparons avec des méthodes tradionelles de RM sur des instances aériennes fournies par la littérature.

Dans le troisième article, nous abordons des instances de taille plus importante pour des problèmes de RM que l'on retrouve en pratique. Nous proposons un algorithme "Action Generation" (AGen) à intégrer au DQN pour étendre son utilisation à des problèmes de plus grande taille de RM sans trop augmenter le coût de calcul. La motivation derrière

cette approche vient d'une analyse des offres optimales à travers l'horizon de réservation qui montre qu'elles sont souvent les mêmes, nous les appelons alors "offres efficaces". À partir de cette information nous pouvons considérablement réduire le temps de calcul dans les cas pratiques. AGen est un algorithme heuristique de type glouton qui mimique la génération de colonnes dans le but de générer ces "offres efficaces". La combinaison de DQN et AGen donne des résultats prometteurs sur les problèmes de plus grandes tailles.

ABSTRACT

This dissertation presents a systematic study of various data-driven advanced methodologies employed to solve a Revenue Management (RM) problem. We address two main modules within an RM system; namely, demand forecasting and inventory control. We start with a general introduction into the thesis and then proceed to overall methodology used to both predict customer demand and analyze the capacity control policies. The methodologies are explained in detail in the three following chapters each of which corresponds to an article already submitted to an international journal. Finally, we conclude with final remarks and discussions of implications for further work. Following is a brief explanation of each article.

In the first article, we study a demand forecasting problem to be addressed for a major railway company. To do so, we explore various preprocessing, machine learning and feature engineering techniques. Moreover, the demand is estimated in two different aggregation levels of data in order to serve different purposes. To comply with the industry-specific requirements, the emphasis of our solution method is on speed, simplicity, and robustness. In this study, the use of state-of-the-art machine learning methods along with innovative feature construction techniques led to high quality results. Although railway industry is the representative of our problem, the studied demand forecasting approaches can easily be extended to other transportation industries or hospitality businesses.

In the second article, we address a choice-based seat inventory control problem in airline industry using a deep reinforcement learning method named Deep Q-Network (DQN). In contrast to traditional RM techniques, DQN does not rely on predicted demand to make informed capacity control decisions. It operates using historical data and/or real-time interaction with customers. In this study, we mainly focus on the choice-based characteristic of our model. We train and evaluate our solution method with synthetic data and compare the final performance to those of well-known RM methods using common flight examples provided in the literature.

In the third article, we tackle large-scale practical RM problems. We propose an "Action Generation" (AGen) algorithm to be integrated into DQN and extend its application to larger RM problems without incurring enormous computational costs. The analysis of the optimal offersets offered to customers throughout the booking horizon shows that only particular offersets (*i.e.,* actions), which we call them "effective sets", are repeatedly used. Thus, if we manage to develop a method to generate such actions, we will be able to substantially reduce the processing time in practical cases. AGen is a greedy heuristic algorithm that mimics

the column generation algorithm [1] with the aim of generating “effective sets”. The AGen embedded DQN yields promising results in large-size network problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Revenue Management (RM) is the application of disciplined analytical tactics to predict customer behavior and optimize both product availability and price with the aim of maximizing the revenue [2]. Practically, an RM problem is composed of the following subproblems: demand forecasting, overbooking policy determination, capacity allocation (*i.e.,* seat inventory control) and pricing which collectively make up the revenue optimization policy [3]. This thesis focuses on solving demand forecasting and seat inventory control problems.

Demand forecasting lies at the heart of any traditional revenue management system. Since determining the most efficient capacity allocation and pricing strategies rely on the predicted demand, any RM system is anticipated to precisely predict upcoming demands. An accurate demand prediction will lead to higher revenue whereas imprecise predictions will cause suboptimal revenue and loss of opportunities. Forecasting future demand is considered a complex problem due to the uncertainties caused by firm's decisions and external factors [4].

In the first phase of this study, we present a machine learning framework for demand forecasting. Throughout the years, many methods have been used to provide precise demand predictions. Machine Learning (ML) approaches are one of the most recent attempts to address demand forecasting tasks. ML algorithms are mathematical tools with the primary goal of learning how to efficiently generalize based on past experiences. ML makes data-driven predictions or decisions by analyzing structural patterns of data and making inferences.

We forecast the future number of bookings for a major European railroad company using advanced ML methods. In order to do so, we benefited from both supervised and unsupervised learning methods. The effects of different contributing parameters on the performance of our solution method are explored and various heuristic feature engineering techniques are developed, accordingly. Our main contributions are as follows:

- applying various state-of-the-art machine learning methods to an industrial railway demand forecasting problem in order to discover hidden patterns in historical data, make inferences based on them and use them to achieve precise and reliable predications.
- developing new heuristic feature engineering techniques to improve the performance of ML algorithms, including shallow and deep features. Shallow features are designed to capture shallow characteristics of data while deep ones target more complex underlying characteristics.

- analyzing the effects of precise clustering and its integration into data on the demand forecasting performance which led to significant accuracy improvement

As the next step, we explore seat inventory allocation problem. In an RM system, the decision maker is required to address the question: Which subset of products the firm should offer to the customers at any given time in order to maximize the total revenue of the firm? To integrate the customers' choice behavior into this process, a category of models named, choice-based models, was developed.

In these models, the subset of products offered by the firm at each time step is regarded as choices offered to the customers. Accordingly, each customer's choice behavior categorizes her in a specific predefined segment. During this sequential decision making process, the firm has to trade off between selling low profit products when resources are ample and securing enough high fare products for the price-insensitive customers.

More particularly, we consider a specific type of RM problem in which perishable products are sold to diverse categories of customers with different booking behaviors throughout a finite time period called booking horizon. In so doing, the goal is to maximize the total revenue, assuming that capacities are fixed [3].

To address choice-based seat inventory control problem, we present a Deep Reinforcement Learning (DRL) approach to dynamically control inventories with the aim of maximizing the revenue. Reinforcement learning is the science of studying sequential decision making processes in which a goal-oriented agent interacts with an uncertain environment. At each time step, the agent receives a feedback from the environment in the form of a reward based on which it makes a more informed decision in the future. The primary goal in RL is to maximize the accumulated long term rewards [5]. In our problem settings, the firm is the decision making agent and customers play the role of the environment.

One of the well-known DRL algorithms is Deep Q-Network (DQN) [6]. Besides being practical and easy to implement, this method is suitable for addressing a choice-based seat inventory control problem due to the following reasons:

- DQN can naturally capture customer choice behavior because of dynamic interaction between the agent and the environment at each time step. This means that the agent is capable of making informative decisions based on historical data and/or real-time interaction with its environment without relying on forecast data.
- it is designed to handle large state space problems using its integrated function approximation unit which generalizes the previous experiences to unseen states.

We aim to contribute to the application of RL techniques in solving practical RM problems. To do so, we choose airline industry as the representative of our problem and use synthetic data to simulate customer choice behavior. The results are compared to those of common techniques provided in RM literature. The main contributions of this research are:

- addressing a choice-based seat inventory control problem using an advanced DRL method, named DQN
- training a real-time goal-oriented agent to offer the most profitable set of products at each time step of the booking horizon with the objective of maximizing the firm's revenue

We extend the application of DQN in seat inventory control problem to large-size and more practical cases. Although DQN demonstrates a promising performance when applied to small examples, it becomes intractable in real-size RM problems because of high computational costs emerging as a result of large discrete action space.

At each time step of the booking horizon, the firm offers a set of products (*i.e.,* offersets) to customers, which we consider as an action. A slight increase in the number of products leads to an exponential growth in the number of potential actions. It is extremely expensive to consider all possible actions at each iteration of DQN algorithm.

In practice, however, we observe that from among all possible actions which the firm could offer only a few ones, which we call "effective sets", were eventually offered. As the objective is to develop an algorithm to generate such sets, we introduce a greedy heuristic algorithm, which we name "Action Generation" (AGen). Once integrated into DQN, AGen iteratively generates "effective sets" resulting in significant reduction in the computational cost of DQN without compromising the high quality of the solution.

In summary, as our main contribution in solving large-size RM problems, we develop an "Action Generation" algorithm to generate a set of most profitable offersets to be made available to customers throughout the booking horizon. This algorithm solves large-size choice-based seat inventory allocation problems when integrated into DQN with promising results.

Next chapter provides the organization of this thesis.

CHAPTER 2 CRITICAL LITERATURE REVIEW

Revenue Management (RM) is a combination of methods which can be used as a decision aid tool to help firms to sell the right inventory unit to the right type of customer, at the right time, and for the right price [7]. In other words, according to Talluri and Van Ryzin [3], RM consists of methods required to manage the firm's interactions with the market with the objective to maximize revenue.

Development of modern RM started right after the Air Deregulation Act in 1978 in the United States. This act allowed airlines to set and modify their prices and operate routes without being required to get approval from the U.S. Civil Aviation Board. As a result, ticket pricing, scheduling, and inventory control strategies were liberalized and thus, airlines started to develop surviving tactics in the new competitive environment [3].

Kimes et al. [7] suggest that RM techniques are applicable to a problem if six general conditions are satisfied: 1) fixed capacity, 2) customer heterogeneity, 3) perishable inventory, 4) product sold in advance, 5) uncertain and fluctuating demand, and 6) low marginal costs. As explained by Helve [8], following is a brief description of each case.

- Fixed capacity: RM techniques can be used to address a problem if the capacity is fixed, at least for a short-term. That is, considering the application of RM in airline as an example, if the type of airplane is fixed for a specific flight, then the capacity is fixed. Even if the seats are sold out in this flight, considering the specific pre-assigned airplane, it is not reasonable to change the airplane because of high costs associated with it.

- Customer heterogeneity: One of the techniques implemented in RM is dividing customers into different categories based on their preferences. These preferences closely relate to customer's price and time sensitivities. More specifically, customer segmentation is mainly functioning based on customer's Willingness To Pay (WTP) and their purchase behavior over time. For instance, the assumption is that price-sensitive passengers book their flights far before the departure date.

- Perishable inventory: In many service industries, such as airlines and entertainment show businesses, the inventory is perishable as opposed to physical products-based manufacturers. That is, if a seat in an airplane or a theater is not sold before the departure date or show time, the inventory perishes. Thus, the perished inventory can not be used to fulfill future demand. Apparently, if there is a possibility to store

the inventory, RM techniques are not required anymore. Due to the importance of inventory management, seat inventory allocation is a crucial part of any RM approach to optimize inventory usage and hence maximize the obtained revenue.

- Products sold in advance: For an RM method to be applicable, a reservation system is required using which a firm can sell inventories in finite time horizon before the actual use (*e.g.,* departure date in airline). The firm applies different purchase restrictions throughout this period considering customer segmentations and their preferences. Traditionally, RM tactics are designed to sell part of capacity to price-sensitive customers while securing seats for time-sensitive customers which usually have higher WTP.

- Uncertain and fluctuating demand: Demand forecasting plays an important role in traditional RM. A precise prediction helps to increase revenue by improving inventory allocation, however, this is a challenging task because of various uncertainties involved in the process. The more the fluctuations and uncertainties, the more difficult the demand management becomes. Thus, RM techniques aim to address some level of uncertainty in demand forecasting.

- Low marginal costs: To make an RM system more efficient, the marginal cost of providing extra unit of available capacity should be reasonably low. As an example, referring to the first condition where fixed capacity in airline was discussed, when the firm selects the type of airplane for a particular flight, it practically fixes both the capacity and costs associated with it since the total cost does not depend on how many passengers are on a flight [3].

We focus on an RM problem in which perishable resources with fixed capacity are sold through different products to heterogeneous customers during a reservation period called booking horizon. Airline and railway industries are two vivid examples in which RM could be of great importance and we consider them as representative of our problem.

In literature, RM systems consist of two main components: a demand forecasting unit which feeds required estimated demand data to an optimization unit and the optimization module responsible for providing the best price and allocation scenario. Having a fixed capacity, inventory control is considered to be a critical part of an optimization unit for maximizing revenue. The remainder of this chapter is divided into two main categories: demand forecasting and inventory control.

2.1 Demand Forecasting

Demand forecasting can be addressed in three different levels: Macro-level, Micro-level, and choice modeling. Macro-level is a top-level aggregated forecast which contains high-level general information, such as industry level demand prediction. Micro-level, on the other hand, is a disaggregated and more detail-oriented estimation of demand, for example, forecasts of customer demand for a specific class in a particular flight/train. Alternatively, choice modeling is interested in forecasting each customer's socioeconomic behavior. For example, forecasting individual's choice between transportation modes [8].

In order to perform any kind of quantitative forecast, some general conditions should be satisfied: having access to historical data, being able to present this data in numerical format, and finally, the requirement that this data should be representative of future customer behavior to some extent (*i.e.,* the assumption of continuity) [9]. Meeting the required criteria, there are various contributing parameters which make accurate quantitative forecast a challenging task such as [4]:

- fluctuations and variations in demand caused by dependencies on time of day, day of week and week number in a year or existing of special events or time/price sensitivity of customers.
- level of sensitivity of customers to price changes: The firm's decision to modify ticket prices over booking horizon directly affects the level of demand and the amount of fluctuation in demand depends on the sensitivity of each customer's sensitivity to price.
- restrictions imposed by the firm: The firm can redirect customers from low fare tickets to higher fares by closing the bookings for the former and opening them for the latter. Whether the customer will buy a high fare ticket or will leave without any purchase depends on the customer's choice preferences.

Throughout the years, demand forecasting and its uncertainties have been addressed in numerous studies. Here, we mainly focus on micro-level forecast to review the required literature for the first section of this thesis.

As described by Weatherford et al. [10], demand forecasting models can be categorized in three main groups:

- historical booking models such as moving average, exponential smoothing and ARIMA,

- advanced booking models such as additive and multiplicative pick up and other time-series models, and
- combined models such as regression.

Historical booking models only consider the final number of passengers for a specific departure time as the required historical data to construct their model, while advanced booking models only include the build-up of reservations over time for a particular departure time. The combined models, on the other hand, take both final booking information and build-up reservation data into account [10].

In a railway RM system, where the aim is to forecast the future number of bookings for each departure time, the main focus has been on the following models: moving averages, exponential smoothing, pickup, and regression models [4]. A complete description of each model could be found in a comprehensive book about forecasting models published by Makridakis et al. [9].

Recently, an advanced category of regression models called Machine Learning (ML) models has been explored. ML methods are mathematical tools which aim to learn to generalize from experience. They extract underlying patterns of available data to maximize the prediction accuracy of future instances [11].

ML methods has been applied to different categories of demand forecasting such as hotel reservation, car rental and transportation. For example, Sanz-Garcia et al. [12] developed a hybrid method to predict hotel room reservations with the main focus on the impacts of last-minute reservations. Another interesting study with the primary emphasize on using ML to forecast booking cancellations, was published by da Conceicao Antonio [13] as his PhD thesis.

In a comprehensive review paper, Shadi Sharif et al. [14] analyzed and categorized various statistical and ML techniques used for demand forecasting in revenue management. Overall, ML methods show promising results when applied to demand forecasting problems. This is an ongoing research topic which we address in this thesis.

2.2 Inventory Control

Inventory control problem in the airline industry addresses the question: Which subset of products the firm should offer to the customers at any given time in order to maximize the total revenue of the firm [3]? This is a remarkably sophisticated problem since numerous contributing factors are involved in the process such as stochastic demand for air travel,

fluctuations in fare prices, multiple-leg passenger itineraries, and diversion of passengers to other fare classes or flights. Although no such model is developed yet which can include all these parameters, various simplified and heuristic approaches have been proposed with noticeable success in practice [15].

The first theoretical model to solve a capacity control problem proposed by Littlewood in 1972 [16]. The proposed model holds various restrictive assumptions such as sequential booking classes, i.e. booking classes do not interconnect with each other, low-before-high price booking arrival, statistical independence of demand between booking classes, no batch bookings, i.e. only one booking at a time, etc. The model was applied to a simple example with single leg and two fare classes. According to Littlewood's method, if the expected revenue of selling the same seat at the higher fare exceeds the certain revenue of selling another low fare seat, then the low fare class should be closed [8].

Belobaba [17] extended Littlewood's inventory control model from a single-leg with two fare classes to multiple fare classes in a nested reservation system. Nested reservation system is a booking method in which the firm makes all the seats of a lower fare class available for the higher fare class but not the reverse. The proposed method, named Expected Marginal Seat Revenue(EMSR-a), is a heuristic probabilistic RM model which is developed based on historical demand data and average fares to specify the booking limits of each fare class. EMSR-b [18] is a revised version of this model in which Belobaba addressed the problem of pooling or statistical averaging which was lacking in EMSR-a. The EMSR-b model solves this issue by aggregating over demand instead of protection levels [3].

Despite their advantages in practice, these methods were designed for a single-leg problem and became intractable in case of flight networks. Later, various approximate methods were proposed to this shortcoming such as virtual nesting, bid price models and mathematical programming. Virtual nesting [19] was developed based on decomposing the network problem into a set of single-resource problems. Bid price control [20], alternatively, performs based on the marginal value of each unit of inventory. Until around 2004, majority of proposed methods were designed according to independent demand paradigm. That is, they consider booking demand as an independent request unaffected by both the inventory control applied by the firm and any market conditions such as prices offered by competitors.

Considering customer choice behavior versus independent demand models were an evolutionary discovery in revenue management, which significantly improved the firms' revenue. However, it has always been challenging to predict the precise customer's purchase habits. The choice-based RM mainly deals with the question: Which subset of products the firm should make available to the customers at any given time in order to maximize the total

revenue of the firm while taking each customer's choice behavior into account [3]? Recently, Strauss et al. [21] published a comprehensive review paper on various customer choice behavior models.

In the literature, there are two categories to address customer choice modeling; parametric and non-parametric. Parametric models are based on random utility theory. In this case, we assume that customers assign a certain utility to each product, and make their decisions based on the choice that maximizes their utility. On the other hand, we can consider non-parametric, rank-based models. Where, we assume every consumer ranks all products and the non-purchase choice in a specific order, and chooses the highest-ranking available option [21].

Non-parametric design of the problem has been explored in various studies [22], [23], [24]. One of the recent studies by Hosseinalifam et al. [25] proposes a flexible mathematical programming framework to address dynamic resource allocation in the airline revenue management problem. The authors extended the proposed non-parametric model with embedding practical and technical constraints [26]. The proposed column generation-based heuristic approaches performed well in terms of both quality and processing time assessed against those of alternative approaches.

Within the category of parametric methods, Talluri et al. [27] were the first to propose the application of Multinomial Logit (MNL) choice model in RM. MNL is a standard approach for determining the probability of purchase in product line problems [28]. They used Expectation-Maximization (EM) approach to estimate the homogeneous arrival rate of the customers and the utility parameters from the historical data. Different customers may have different preferences, thus, an extension of MNL called finite-mixture logit considers multiple customer segments, each assumed to follow a segment-specific MNL model [21].

The problem of choice-based inventory allocation could be formulated as a Markov Decision Processes (MDP) [5] and solved using the following Dynamic Programming (DP) formulation [29]:

$$\begin{aligned} V_t(x) &= \max_S\{\sum_{j\in S}\lambda P_j(S)(r_j + V_{t+1}(x-A_j)) + (\lambda P_0(S) + 1 - \lambda)V_{t+1}(x)\} \qquad (2.1)\\ &= \max_S\{\sum_{j\in S}\lambda P_j(S)(r_j - (V_{t+1}(x) - V_{t+1}(x-A_j)))\} + V_{t+1}(x) \end{aligned}$$

where, $V_t(0) = 0$ for $t = 1, 2, ..., T$, and $V_{T+1}(x) = 0\ \forall x \geq 0$.

It is not possible to solve most of the large-size MDPs with classical DP due to the curse of dimensionality. A common approach in optimization domain is to approximate DP with Linear Programming (LP).

Gallego et al. [30] proposed Choice-based Deterministic Linear Programming (CDLP) framework to solve large-size choice-based RM problems with reasonable computational complexity. Despite the model's various advantages, they did not consider any kind of customer segmentation in their solution approach. Later, Liu et al. [31] and Bront et al. [32] extended the application of CDLP to the case with disjoint and overlapping segments of customers, respectively.

Another category of approximate methods which can be used to find an optimal capacity control policy in inventory control problem is Reinforcement Learning(RL). According to Sutton et al. [5], "Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning." In RM problem settings, customers are the environment and the airline firm is the learner (also called agent) who makes sequential inventory control decisions.

Recent advancements in the field of artificial intelligence lead to development of new models, called Deep Reinforcement Learning (DRL), some of which show human level performance. Deep Q-Network (DQN) [6] is a practical well-known example of DRL methods.

Traditional RM techniques mainly depend on the accuracy of estimated demand, which may be difficult to accurately estimate or may not even be available (*e.g.,* new markets), to maximize the revenue. DQN, however, can address this issue by relying only on the available historical data and/or real-time interaction with the customers instead of using demand prediction results. Moreover, DQN uses function approximation to approximate the values of unseen states and thus, it is naturally designed to address large state space problems.

In a most recent study, Shihab et al. [33] used DRL to solve a single-leg seat inventory assignment problem. Although they achieved promising results, their solution method suffers from lack of considering customer choice behavior as well as more complex structures like hub and spoke network.

CHAPTER 3 SYNTHESIS OF THE WORK AS A WHOLE

Encompassing both demand forecasting and seat inventory control problems, this thesis is organized as follows.

In Chapter 1, the thesis begins with a general introduction about revenue management and two of the main modules in any RM system; namely, demand forecasting and inventory control. We provide a brief explanation of each module and the framework of solution methods that we use to address them. Finally, the main contributions for each case are presented.

Chapter 2 contains a critical literature review of both modules as well as the approaches used to tackle them throughout the years, in a chronological order. We start with a brief history of revenue management and its applications. Then, the remainder of this chapter is divided to two subsections; one for each module in which the historical advancements in solving each problem are explained.

Chapter 3 describes the organization of this thesis. The next 3 chapters present the detailed explanation of our methodologies used for predicting customer demand and finding capacity control policy each of which corresponds to an article already submitted to an international journal. Below is a brief explanation of each article.

Chapter 4 presents the first article in which we study an industrial demand forecasting task for a major European railway company. Various preprocessing, state-of-the-art machine learning, and heuristic feature engineering techniques are developed and/or explored in order to estimate the number of future bookings in two different aggregation levels.

In Chapter 5, the second article is presented where we solve a choice-based seat inventory allocation problem in airline industry using a deep reinforcement learning method named Deep Q-Network (DQN). We train a goal-oriented agent using synthetic data to interact with customers with the objective of finding the optimal policy while taking the customer's choice behavior into account.

Chapter 6 presents the third article in which we extend solving a choice-based seat inventory allocation problem to large-size cases. In order to do so, we propose an "Action Generation" (AGen) algorithm to be integrated into DQN to reduce the computational complexity of large-scale RM problems through generating the "effective offersets" while conserving high performance quality.

Finally, in Chapter 7 and 8, we present general discussion and final conclusion, respectively.

CHAPTER 4 ARTICLE 1 : APPLICATION OF MACHINE LEARNING TECHNIQUES IN RAILWAY DEMAND FORECASTING

Chapter Information : An article based on this chapter is submitted to International Journal of Revenue Management for publication. Authors: N. Etebari Alamdari, MF. Anjos, and G. Savard.

In this paper, we address a demand forecasting problem in railway industry using advanced machine learning approaches and innovative heuristic feature engineering techniques.

ABSTRACT

Demand forecasting lies at the heart of any revenue management system. It aims to estimate the quantity of a product or service that will be purchased in the future. In this paper, we perform railway demand forecasting for a major European railroad company by taking various contributing parameters into account. Using state-of-the-art machine learning methods and various heuristic feature construction techniques, remarkable results with high forecast accuracy and reasonable computational complexity are achieved. To have multipurpose results, the current problem is explored in two different aggregation levels. Although this paper is focused on demand forecasting in railway industry, the studied methodologies can easily be extended to other transportation or hospitality businesses.

Key words: Revenue Management, Demand Forecasting, Feature Engineering, Machine Learning

4.1 Introduction

Revenue Management (RM) is the application of various analytical tactics and mathematical approaches with the aim of predicting customer behavior at the micro-market level while optimizing price and availability of products [34]. RM tasks are shaped by various components including customer segmentation, demand forecasting, pricing techniques and inventory control management.

Demand forecasting plays a vital role in any traditional revenue management system. All the models aiming to answer the question "How to determine the most efficient capacity allocation and pricing decisions?" rely on the predicted demand as the main building block of an RM system. As emphasized by McGill et al. [35], all RM decisions are made based on different forecasts, particularly, customer demand which provides input data for the capacity and pricing optimization module. Forecasting future demand is a complicated task due to uncertainties caused by the firm's decisions and external factors [4].

Companies can improve the quality of their pricing and capacity control systems by increasing the accuracy of their predicted demand. Over the years, a fundamental collection of forecasting methods has been developed and new improvements have continued to evolve. Some of these forecasting methods are based on solid mathematical and statistical foundations while some others are largely heuristic in nature. In terms of forecasting methods, since a large number of forecasts have to be made during a limited time period; thus, fast, accurate and simple methods are preferred in RM [3].

One of the early works on statistical demand forecasting in airline industry using time series data was done by Sen [36]. Since then, there have been numerous "time series analysis"-based studies with the aim of improving the forecast accuracy and achieving more stable and generalizable models [37], [38] and [39], [40]. A well-known and extensively explored time-series analysis method is AutoRegressive Integrated Moving Average (ARIMA) [41], which is a generalization of AutoRegressive Moving Average (ARMA) to non-stationary data.

Over the years, various booking models have been explored, models such as pickup, advanced pickup and booking profile which are based on registered bookings over time, and can be of the additive or multiplicative type [42]. More detailed information on these models could be found in the literature [43], [44], [45]. Simple and weighted averages are also among the popular demand forecasting methods which were outperformed by pickup models [46]. Cleophas et al. [47] summarized recent developments in demand forecasting for airline revenue management.

One of the recent categories of models addressing demand prediction in RM is Machine

Learning (ML). ML methods [11] are mathematical tools with the core objective of learning to generalize from experience. They mainly rely on the underlying patterns and characteristics of historical data in order to minimize prediction errors of unseen data. In general, ML algorithms are classified into two main categories: supervised learning and unsupervised learning. The goal of supervised learning is to infer a functional mapping according to a set of input-output training examples. Unsupervised learning, on the other hand, discovers patterns and structures hidden in data without having access to labeled output.

Classical statistics-based methods, such as time series, may struggle to cope with high-dimensional data sets and sometimes fail to respond accurately to sudden changes. Machine learning methods, however, are more flexible when dealing with sudden changes in the format of data, missing information, and high-dimensional data sets [42].

Demand forecasting, as a regression prediction problem, has been also studied extensively with the help of various ML techniques. For instance, Ziekow et al. [48] used ML methods to evaluate the use of disaggregated smart home sensor data for household-level demand forecasting. ML methods are also used for urban water demand forecasting in situations with limited data availability. These methods were tested using three years of daily water demand and meteorological data for the city of Calgary in Alberta, Canada [49]. A thorough review paper on the application of machine learning models to commercial building electricity load forecasting was published by Yildiz et al. [50].

Booking demand forecast is also one of the crucial decision-making challenges in service industries which is extensively studied through ML techniques. In an interesting study, Sanz-Garcia et al. [12] developed a hybrid method to estimate hotel room reservations that explores the effects of last-minute reservations. A very recent study on hotel reservation management has been published by da Conceicao Antonio [13] as his PhD thesis, which emphasizes on using ML to predict booking cancellations. With a focus on service companies, Shadi Sharif et al. [14] analyzed and categorized various statistical and ML techniques used for demand forecasting in revenue management.

In this paper, we forecast the future number of bookings for a major railroad company by taking various contributing factors into account. In order to do so, we use different ML approaches along with heuristic feature engineering techniques.

Forecasting is a complex task, however, it can be broken down into simpler steps. We perform our forecasting task in two aggregation levels. These levels are created based on the zonal data and used to demonstrate the overall performance of the prediction models. At each level, the forecast is considered good if it is accurate, plausible, simple, quick and flexible.

Overall, this research intends to contribute to the application of ML techniques in RM. More specifically, it addresses the problem of demand forecasting in revenue management by proposing:

- new heuristic feature engineering techniques including shallow and deep features,
- exploring the importance of accurate clustering and its integration into data, and
- implementing state-of-the-art machine learning methods in order to discover complex hidden patterns of data and improve the accuracy of predications.

The remainder of this paper is organized as follows: Section 4.2 provides problem description in course of which general definitions and problem settings are explained in detail. In Section 4.3, we discuss the details of various types of preprocessing, machine learning and feature engineering techniques which will be used in this paper. In Section 4.4, the numerical results of applying such methods in both aggregation levels of data are demonstrated and analyzed. Finally, the concluding comments are outlined in Section 4.5.

4.2 Problem Definition

We start with introducing some technical terms and general definitions which will be used throughout this paper. Afterwards, we will go through the details of demand forecasting problem in the context of railway industry.

- Market: an origin-destination pair between which the passengers wish to travel
- Itinerary: a specific sequence of legs on which passengers travel from their origin to their ultimate destination
- DBD: number of days before departure
- Booking horizon: time horizon in which bookings are open
- Booking period: booking horizon between each two subsequent DBDs
- Time-range: a predefined time horizon during the departure day which is an aggregation of departure times
- Demand: expected demand of a product in a market which depends on itinerary, time range and period

- Fare Class: different prices for the same itinerary, usually distinguished from one another by the set of restrictions that firms impose
- Product: an itinerary and fare class combination

To have a reliable multipurpose forecast, we treat the data in two different aggregation levels. One of the main reasons for forecasting the potential demand in different levels is to meet the railroad company's specific needs. For instance, for overall planning of all trains, it is sufficient to have a less detailed estimated values. However, it is necessary to perform a more comprehensive forecast for inventory control and pricing purposes.

In this problem setting, DBDs are defined as of 120 days before departure date. The period between DBD119 and DBD-1 (*i.e.,* departure date) is divided into 20 booking periods. Note that booking periods are not necessarily of the same length. In the beginning of the horizon, booking periods consist of several days; however, they become shorter as we get closer to the departure date and get as short as one day within the last few days before the departure day.

Level I

This level provides an overall view of the data. In this top-level, the historical booking information are aggregated by booking periods. We would like to forecast the total number of bookings for all trains departing on a specific departure date and within a certain time range. For example, the illustrated area in Figure 4.1 presents the total number of bookings that we aim to forecast for a given departure date in the time range of 7:00 am - 9:00 am.

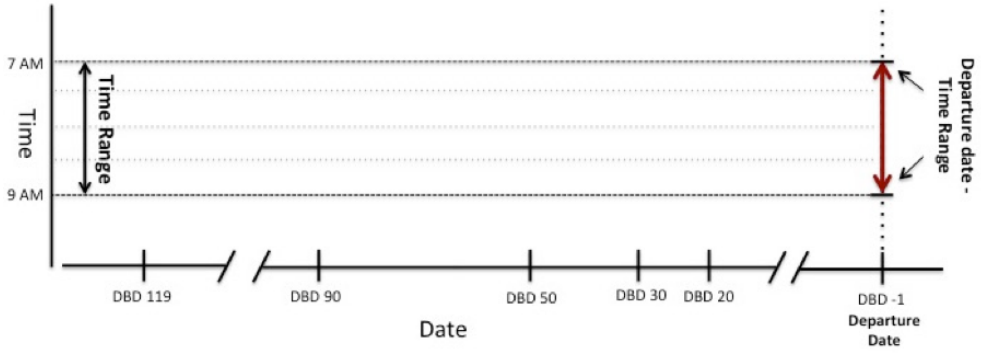

Figure 4.1 Demand forecasting in Level I

Level II

In level II, we add the dimension of booking period to level I data. Consequently, in this level, the prediction models aim to compute the total number of bookings within each booking period for all trains leaving in a specific time range of a certain departure date. Figure 4.2 shows the total number of bookings for a given departure date in the time range of 7:00 am - 9:00 am that was particularly booked in the booking period between two consequent DBDs of 90 and 50.

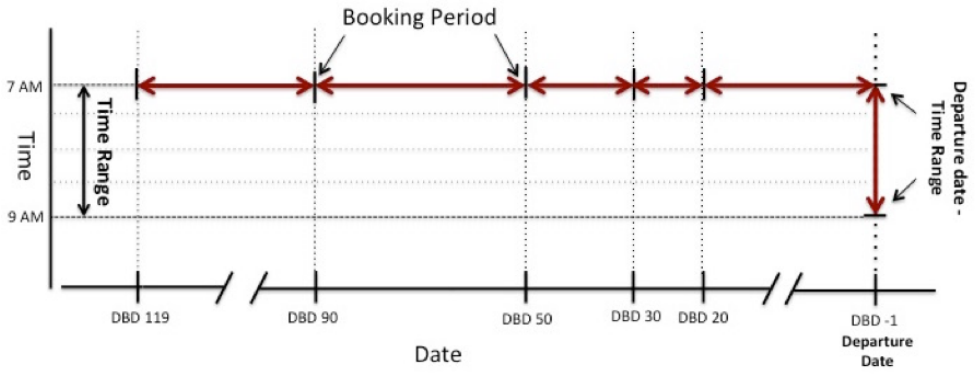

Figure 4.2 Demand forecasting in Level II

4.3 Solution Methods

In this section, we explain the details of preprocessing steps, model selection process and feature engineering techniques used for railway demand forecasting. Our dataset consists of two years of historical booking data (*i.e.,* 2013 and 2014) collected from travels between two major European cities, which is provided to us by a major European railroad company. The objective is to predict the potential demand of the future bookings based on the historical data in two different aggregation levels.

4.3.1 Preprocessing

Data preprocessing is a process to transform raw data into a format that is usable as an informative input to predictive models. Industrial datasets usually require various steps of preprocessing such as data cleaning (*e.g.,* missing values imputation and noisy data smoothing), data transformation (*e.g.,* normalization and aggregation), data reduction, and data discretization.

We start data preprocessing with verifying data type and data representation consistency. Our initial raw dataset consists of a few attributes including departure date and time, booking periods and potential period demand with no missing values.

We extract useful information from the departure date feature and construct new attributes; namely, month, week number, week day and date value. The first three provide us with intuitive and valuable knowledge regarding departure dates. Moreover, the date value is a numerical representation of the departure date that shows the number of seconds since 1970. This method of representation helps us to preserve intervals and keep the order of events.

As the next step, we perform data discretization, a method to reduce the number of values of a continuous feature by dividing it into predefined number of intervals. In this step, we split each departure day into six time ranges based on the popularity of departure times (*e.g.,* 19:00-23:59 is one of the time ranges).

Many machine learning models require numerical values as their input, and this translates into the necessity of transforming categorical features into numerical ones. Depending on the nature of the categorical data, we have various options to do so such as one hot encoding and integer number assignment.

One hot encoding is one of the most well-known encoding schemes used to transform a single categorical variable into its corresponding binary variables. Each binary variable takes "1" when its associated category is present, and "0" otherwise [51].

For example, categorical features such as month and weekday could be integrated into the data using one hot encoding technique. However, it may result in dimension augmentation. This should not be problematic since we have a large enough number of samples to avoid overfitting. The categorical booking periods attribute; however, is different as there is an order in it which allows for assignment of integer numbers.

As an initial clean format of data available to us, we can perform an outliers detection algorithm now. Outliers are extreme perturbations in the data caused by occasional unpredictable events. Outlier detection is considered an extremely important step since outliers can impose remarkable noise on the mean and variance of the entire dataset and distort the real pattern of the data. In this study, we use modified Z-score to detect and then remove extreme outliers.

Z-score or standard score [52] discovers by centering and rescaling of data, and then, detecting the points that are far from the mean. When using mean and standard deviation themselves are directly affected by outliers, this method is not robust enough. In modified z-score, although the intuition is the same, we use the median and Median Absolute Deviation (MAD)

to measure central tendency and dispersion, respectively. Thus, modified z-score turns out to be a more robust method in terms of detecting outliers [53].

Upon completion of this step, the data is ready for further analysis.

4.3.2 Model Selection

We start this section with a brief review of ML models that will be extensively used in this paper. Afterwards, we explain the model selection process for level I and level II data. Finally, we describe the feature engineering process for level II data.

Model Description

Many tasks in machine learning can be expressed as a classification or a regression problem. Regression estimates the conditional expectation of a dependent variable given the independent ones whereas classification predicts categorical class labels.

The simplest regression model is linear regression which is capable of capturing linear relationships between predictors and target, but we mainly deal with more complex and nonlinear tasks such as demand forecasting in the real world.

In general, tree-based models and Neural Networks (NNs) are two main categories of models used for demand forecasting in the literature. Both of them are supervised learning methods used for regression and classification purposes. The intuition behind NNs is to extract linear combinations of inputs as derived features, and then, to model the target as a non-linear function of those features [54].

A Decision Tree (DT), as a building block of any tree-based method, is a decision making tool that uses a tree-like model to estimate the value of a target variable by learning simple decision rules deduced from data attributes. DTs are the foundation of very powerful predictive models such as tree-based bagging and boosting ensemble models [11].

Bagging (bootstrap aggregating) is an ensemble averaging meta-algorithm that improves the prediction accuracy by reducing variance [55]. In decision tree-based bagging, each bootstrapped sample is used as a training set to grow a decision tree, and the result is the average over the predictions of all trees. Random forests, also known as random decision forests, is a practical ensemble method designed based on the bagging idea [56].

Boosting methods are built sequentially over weak regressors in order to reduce the bias [57]. The final meta-algorithm is a linear weighted combination of the base estimators with a reduced generalization error. One of the most well-known boosting models is Gradient

Boosting Trees (GBT) which is a generalization of boosting to arbitrary differentiable loss functions using decision trees as base estimators [58].

Model Selection

In this section, we describe the model selection process in the level I and the level II data.

Level I

We start model selection with the level I data. Note that the data used in this level are already preprocessed, the outliers have been removed and the basic features such as time range added to ensure improved performance of any ML method we may choose. Having a top level aggregated data, we expect that application of a proper ML method will result in an acceptable performance.

Once we applied various ensemble tree-based and neural networks methods on level I data, we achieved sufficiently good results with both NNs and GBT. Since the acquired performance was considered to be efficient according to the industry's guidelines, we focused on level II as a more detailed and complex level.

Level II

We start model selection of level II with evaluating various regression methods in order to compare their performances and achieve a benchmark for more advanced techniques. On this level, among the initially tested methods, GBT outperforms others.

For this aggregation level, considering the fact that the data are more complex, the common ML regressors do not improve the results more than a certain limit. Thus, various combinations of different models such as regular and weighted mixture of regressors are explored. Among these models, stacking, also known as stacked generalization, provides the most accurate predictions while keeping the processing time in a reasonable range.

Stacking is a meta-learner that consists of multiple model mixtures. The idea is to learn a function that combines the predictions of the individual regressors and feed them as input into the final meta learner. This method was originally introduced in 1992 by David H. Wolpert [59] for a classification task. Algorithm 1 represents the general approach to a regression task.

In summary, each base regressor's prediction is used as a new feature in the meta dataset (*e.g.*, data fed into the meta regressor), then, a meta regressor is applied to the meta data. Figure 4.3 illustrates an schematic view of stacking algorithm, where, $R1, R2, \ldots, R5$ refer to base regressors.

Algorithm 1 Stacking algorithm

1: Train n different regressors $R_1, R_2, \ldots, R_n$ (the base regression models).
2: Obtain predictions of each regressor.
3: Form a new dataset using predicted values: the meta data.
4: Train a separate regressor on the meta data: the meta regressor.

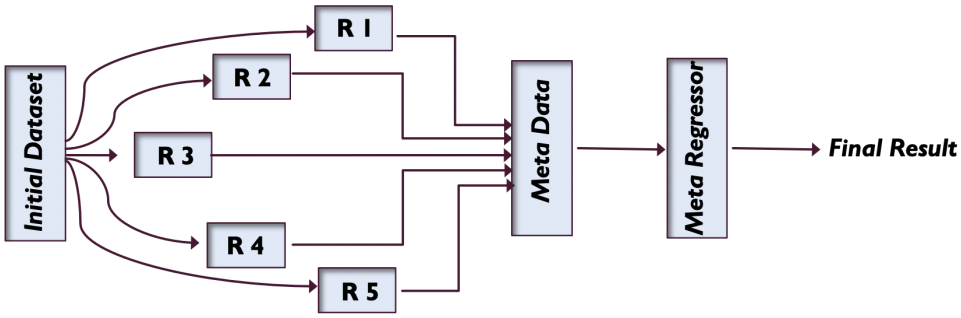

Figure 4.3 An schematic view of stacking algorithm

Although stacking outperforms all other examined regression methods, the high complexity of level II data necessitates further accuracy of forecasting results to ensure that the obtained predictions are a useful source of reliable information for industrial purposes.

4.3.3 Feature Engineering

Feature engineering is the process of using domain knowledge to design attributes that improve the performance of machine learning algorithms. We start with shallow features, which consider shallow characteristics hidden in the data and can be easily extracted from the dataset. Deep features, on the other hand, are the ones that require an algorithmic approach to be constructed, and unlike the shallow ones, they aim to capture deeper characteristics of the data.

Note that we perform feature engineering only on level II data since we have already achieved desirable results on level I thanks to selecting proper ML methods.

Shallow Features

At this step, shallow features are explored. Since ML algorithms are designed to capture hidden characteristics of the data, having more informed attributes will increase the possibility of discovering such characteristics.

By using shallow features, we intend to capture the trends of bookings for each departure date starting as of 120 days before the departure date. We define an observation date as the date on which we are observing a snapshot of the bookings made so far for all the departure dates. An observation date can be any day during the year.

Four new features are also constructed in addition to the observation date attribute. We divide the 120-day booking window prior to each departure date into four 30-day periods and dedicate one attribute to every single one of them. Each attribute indicates the total number of bookings made during its associated month. The observations occur every seven days that means our data are updated on a weekly basis.

For example, if a departure date is July 1st, we will have four new features for the bookings made during March, April, May, and June, separately. If our observation date is some time prior to March 1st, the features will have zero values because the booking has not started for this specific departure date yet.

By moving forwards within the booking window, the booking information will be updated every seven days. For instance, having an observation on April 7th means we have full information regarding total bookings made in March for the departures on July 1st. However, at this point, only one week of data is available for the month of April, and the rest of attributes (*i.e.,* one and two months prior to the departure date) are zero.

These features extracted valuable information from our dataset and improved the accuracy of demand estimation.

Another category of shallow features is external features. We gathered some external data that may affect bookings such as weather abnormalities in the departure and the arrival cities (*e.g.,* snow storm, extreme heat warning, flooding, etc.) along with max, min, and mean temperatures on the departure day. Although weather forecast might not be accurate long ahead of the departure date, it is still an easily accessible data and can be updated anytime needed.

Deep Features

We realized that having proper clusterings of data, as an added feature, would reduce the forecast error significantly. To examine this assumption, K-means clustering is applied to the dataset while having access to the target variable, and the cluster labels are added as a new feature to the whole dataset. This results in achieving the lowest bound of error and validates our initial assumption.

K-means [60] is a simple unsupervised learning method that clusters unlabeled given points into K-predefined number of clusters using an expectation-maximization algorithm. Starting

with randomly defined K centroids, the data points are assigned to the closest centroids so that each group creates a cluster. At the next step, the model recalculates K new centroids and assigns the closest points to them, resulting in a new set of K clusters. This loop runs until centroids do not move anymore.

In order to find the optimal number of clusters, the K-means algorithm is pipelined to a supervised regression method (Any appropriate regression method is applicable at this step, including random forests.) for evaluating the errors of each value of K. The "elbow method" is used to find the optimal K.

To take advantage of this finding in regular settings, where the target variable is not provided, the following steps are taken. First, using K-means method, the train set is clustered into k predefined clusters and the obtained labels are considered as a new target variable for the train set. Afterwards, test data are classified into the same cluster labels. Finally, the cluster numbers are added to the original test and train sets and we proceed with the regression problem. Note that in step 4, the accuracy of classification task is as important as that of the final regression problem and it can be performed using various ML techniques such as stacking.

Algorithm 2 displays the steps of generating a clustering-based feature.

Algorithm 2 Clustering-based Feature Construction

1: Cluster train data into K predefined clusters using K-Means algorithm.
2: Modify train set by removing the demand feature from the dataset.
3: Consider cluster labels as the new target variable in the train set.
4: As a classification task, classify test data into the same K cluster labels.
5: Use the predicted labels as a new feature in both original train and test sets.
6: Apply a regression model (*e.g.,* stacking) to the dataset with the new added feature.

In the next section, we provide numerical results associated with the selected models and added features.

4.4 Numerical Results

We start this section with a general analysis of data. Afterwards, we explain the choice of error evaluation metrics. As the next step, we report the results of various demand forecasting techniques used to predict the number of bookings in different aggregation levels of data: level I and level II.

The computational operations have been carried out on a 2.9 GHz 5-core computer with

16 GB of RAM and the codes are written in Python 3.5. Moreover, we use 2013 dataset for training and validation, and 20% of the data from the first quarter of 2014 dataset is considered to be the test set.

In this study, we focus on tree-based methods and neural networks as two common categories of regression methods to tackle the demand forecasting problem. The ensemble tree-based methods we explore are either bagging methods such as Random Forests (RF) [56] and Extremely Randomized Tress (ERT) [61] or boosting methods such as Gradient Boosting Trees (GBT) [58] and AdaBoost [62].

Evaluation Metrics

Calculating demand forecast accuracy is a process of determining the accuracy of predicted demand compared to actual customer demand. In this paper, we use both Weighted Average Percentage Error (WAPE) and Root Mean Square Error (RMSE) metrics to measure the accuracy of our demand forecasting methods.

WAPE is the quotient of the sum of the absolute deviations divided by the total actual demand. The equation is as follows:

$$WAPE = \frac{\sum_{i=1}^{n} |F_i - A_i|}{\sum_{i=1}^{n} A_i} \times 100 \tag{4.1}$$

where, A and F represent actual and predicted values of demand, respectively. In general, WAPE is easy to understand and interpret because it measures the error in the percentage format. Moreover, since the denominator is a sum over all actual values, WAPE is capable of handling small or zero actual demands. This is an important feature in our case since in some circumstances we have actual demand of zero.

RMSE, on the other hand, is the standard deviation of the residuals.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (F_i - A_i)^2} \tag{4.2}$$

Here again, A and F denote actual and predicted values of demand, respectively. As the above equation indicates, since the residuals are squared before they are averaged, the RMSE gives a moderately high weight to large errors. Consequently, when large errors are particularly undesirable (*e.g.,* in demand forecasting tasks), RMSE could be the evaluation metric of the choice.

Data Analysis

As data visualization is a critical tool for data analysis, we provide some useful insights into the structure and patterns of our dataset.

Figure 4.4 shows the actual cumulative demand for each time range as we start from the initial booking period of DBD119-DBD90 and end on the departure date DBD0-DBD-1. Obviously, bookings increase significantly as we get closer to the departure date.

This figure also illustrates the differences in the customers' booking behaviors in various time ranges. The time range of 16:00 - 19:00 is the most popular one and 09:00 - 12:30 is the second most demanded option. That is, throughout the year, most on-demand travels happen either in the evening or right before noon.

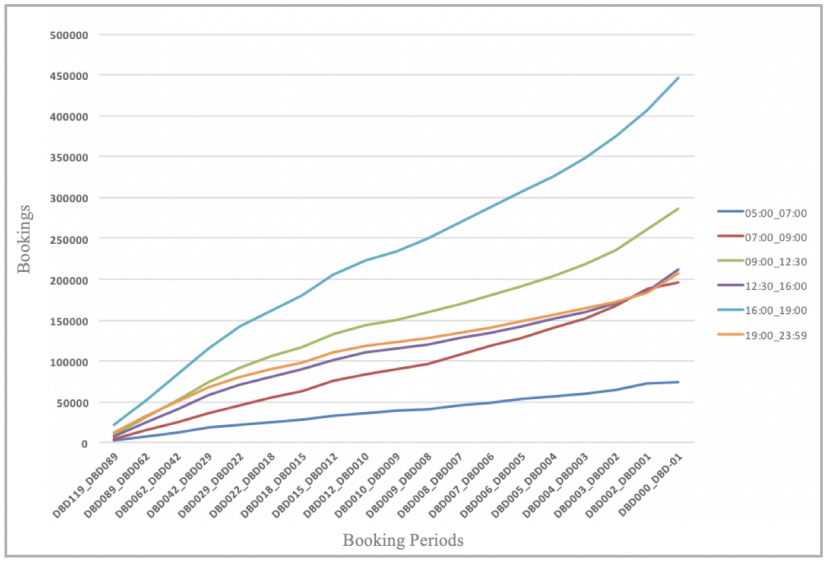

Figure 4.4 Total actual cumulative bookings by booking period and time-range

The trend in the number of bookings based on booking periods and time ranges is illustrated in Figure 4.5. Note that only the data of one month is displayed for the sake of clarity and simplified representation. This graph shows a meaningful difference between the number of bookings occurred in early morning compared to the rest of the day.

During the illustrated month, in almost all time ranges, the bookings follow a similar pattern: starting low in booking demand, reaching the first peak within the initial couple of booking periods, breaking the trend and hitting the minimum about a week before the departure date, and finally raising and reaching the highest number of bookings on the exact departure date.

The picks of this pattern can be explained by the price-sensitive and time-sensitive customers' behavior: price-sensitive customers tend to purchase their ticket much earlier in the booking horizon and time-sensitive customers (*i.e.,* business travelers) book closer to the departure date.

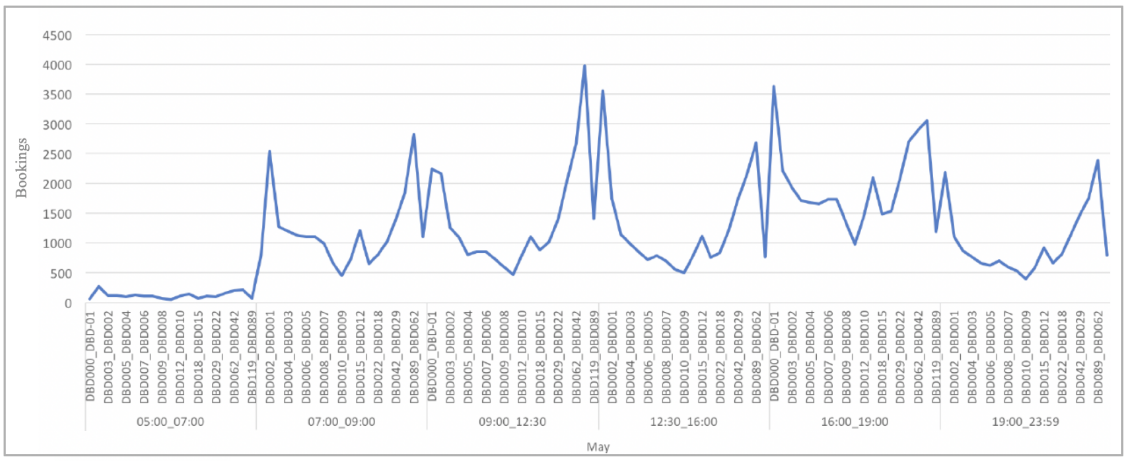

Figure 4.5 Total actual bookings trend by booking period and time-range (May).

Results of Level I Data

After preprocessing step and data analysis, we explore various ML methods on level I data including various ensemble tree-based methods and NNs. Among them, GBT and NNs outperform others. Although GBT achieved only slightly more accurate results than NNs did, we pick GBT as the best performing model.

Industry-wise, there should be a trade-off between the time and efforts dedicated to tuning a model on the one hand, and robustness and accuracy of the results on the other hand. Having almost the same level of computational complexity and accuracy, the NN requires very precise tuning to provide the same results as GBT does. Overall, GBT is more robust and generalizable for the purpose of this problem.

The experiments show that in our case study, ERT is competitive with NNs but not as accurate as GBT. Table 4.1 provides the results of applying all learning methods to level I data. As illustrated, with a WAPE test result of 10.21%, GBT outperforms other models. Note that the processing time for all applied methods of level I is around one minute. Thus, processing time is not a contributing factor in model selection at this step.

Considering various factors, including acceptable accuracy, computational complexity and robustness, GBT satisfies the company's requirements and we explore the more complex

Table 4.1 Results of learning methods on level I data

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train
NNs	10.78 %	7.14 %	96.78	72.10
AdaBoost	12.18 %	8.30 %	119.26	91.08
ERT	11.65 %	8.13 %	105.76	86.21
RF	13.46 %	9.89 %	141.70	100.31
GBT	10.21 %	7.12 %	91.03	69.85

data of level II in the next section.

Results of Level II Data

On this level, we evaluate the performance of the same ensemble tree-based and NNs methods as in level I data. Here again, we observe that GBT outperforms other regressors. However, the achieved results are much less accurate with a higher percentage of the errors due to finer level of aggregation and increased level of data complexity. We use the results of this step as a benchmark to compare with those of the future steps.

Table 4.2 provides results of applying all learning methods to level II data after initial preprocessing and outlier removing. As demonstrated, at this level, the boosting methods (i.e., AdaBoost and GBT) are very competitive in terms of accuracy.

Table 4.2 Results of learning methods on level II data

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train	Processing Time (min.)
NNs	37.54 %	35.86 %	19.31	18.10	2.40
AdaBoost	36.49 %	36.31 %	18.87	18.61	3.54
ERT	38.88 %	37.04 %	20.52	19.11	2.15
RF	40.11 %	38.76 %	22.37	20.19	2.04
GBT - BENCHMARK	35.90 %	35.08 %	18.83	18.51	3.34

Having the benchmark and best performing regressor at this step, we can explore the effects of outlier removal that we used as the last step of preprocessing section. That is, we apply GBT on fully preprocessed data except for the outliers removal, and then, we compare the results to the ones achieved after detecting and removing the outliers. Table 4.3 shows performance

improvement of GBT as a result of outliers removal.

Table 4.3 GBT result improvement due to outliers removal

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train
GBT - Before outliers removal	51.03 %	50.49 %	27.65	27.28
GBT - After outliers removal	35.9 %	35.08 %	18.83	18.51

At the next step of the model selection process, we examine the effects of combining various regressors. The motivation behind this idea is that methods like averaging helps with decreasing the prediction error by reducing variance. To do so, we start with computing predicted demand using selected regressors. At the next step, we average over the predicted values. As an example, Table 4.4 shows the error reduction using averaging over three regressors: RF, GBT, and ERT.

Table 4.4 GBT result improvement due to averaging over predictions of regressors

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train	Processing Time (min.)
GBT - BENCHMARK	35.9 %	35.08 %	18.83	18.51	3.34
Mixture of Regressors	29.01 %	28.56 %	15.61	15.32	7.58

Weighted averaging is another technique used to reduce errors; however, the slight accuracy improvement depends on precise weight assignment. The preference is to find a more robust method of a noticeably higher performance.

According to Table 4.5, stacking results in a remarkably better performance by decreasing the WAPE test result to 23.76%. Although it requires longer processing time, it is still much lower than the industrial computational complexity limits.

The main challenge in improving the results of stacking algorithm lies in the choice of base estimators and the meta regressor. This issue is still referred to as “black art” in the literature. Although there have been some attempts to automate this process or to define a criterion for selection process, no satisfying method has been developed so far [63].

In this study, we use GBT, ERT, Linear regression, RF and KNN as base regressors and RF as the meta regressor. Note that KNN stands for K-Nearest Neighbors [64] and is a simple yet powerful supervised learning method for both classification and regression tasks. In a regression problem, given an example to predict, KNN performs based on finding K most

Table 4.5 Performance improvement as a result of applying stacking to level II data

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train	Processing Time (min.)
GBT - BENCHMARK	35.9 %	35.08 %	18.83	18.51	3.34
Mixture of Regressors	29.01 %	28.56 %	15.61	15.32	7.58
Stacking	23.76 %	23.99 %	13.01	12.98	11.31

similar examples from the training data, called nearest neighbors, and estimates its value as an aggregation of the target values associated with its nearest neighbors.

In stacking, each one of the implemented regressors contributes to capturing underlying characteristics of data using different methodologies. Thus, having various categories of regressors increases the overall performance of stacking method. For example, we include ensemble tree-based (both boosting and bagging), linear regression and K-nearest neighbors methods in the base regressors.

We explored other various innovative methods in order to combine different regressors such as weighted stacking and double stacking. In weighted stacking, the predictions of each base regressor (*i.e.,* features of the meta data) are given normalized weights. To assign weights automatically, we use the feature importance characteristic of RF. When applied to data, RF is capable of assigning importance values to each feature within the dataset. In double stacking, we consider meta data as an initial dataset for another stacking method. In both cases, we decided to ignore the negligible improvements to keep the method simple for industrial purposes and avoid increasing the processing time.

Table 4.6 demonstrates the results of applying shallow features to our dataset. Although the accuracy increase using shallow features is not as high as when stacking is used, this is a single step of the feature engineering process and we expect that the overall combination of shallow and deep features lead to satisfying final results.

As explained in Section 4.3.3, accurate clustering has a significant role in improving the performance of demand forecasting. To validate the clustering effect experimentally, we clustered our dataset into $K = 10$ groups using K-means clustering method while having access to the actual target variable.

Note that this experiment is performed separately and exceptionally in order to validate the assumption about the importance of clustering-based feature, and it was not integrated into any parts of our actual methods. As a result of this experiment, the forecast test error

Table 4.6 Performance improvement as a result of applying shallow features to level II data

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train	Processing Time (min.)
GBT - BENCHMARK	35.9 %	35.08 %	18.83	18.51	3.34
Mixture of Regressors	29.01 %	28.56 %	15.61	15.32	7.58
Stacking	23.76 %	23.99 %	13.01	12.98	11.31
Shallow Features	21.24 %	20.18 %	12.25	11.64	24.30

Table 4.7 Accuracy improvement as a result of adding optimal clustering feature

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train
Stacking	35.9 %	35.08 %	18.83	18.51
Stacking with optimal clustering	9.65 %	9.66 %	4.76	4.74

dramatically drops to below 10% WAPE, which validates our initial assumption. The comparison of stacking results before and after adding optimal clustering feature is demonstrated in Table 4.7.

We constructed the deep feature based on this finding. The forecast accuracy improvement using deep feature is demonstrated in Table 4.8. Using clustering-based feature, we successfully reduced the WAPE test and train results to 18.78% and 16.92%, respectively.

Table 4.8 Performance improvement as a result of applying deep feature to level II data

Algorithm	WAPE Test %	WAPE Train %	RMSE Test	RMSE Train	Processing Time (min.)
GBT - BENCHMARK	35.9 %	35.08 %	18.83	18.51	3.34
Mixture of Regressors	29.01 %	28.56 %	15.61	15.32	7.58
Stacking	23.76 %	23.99 %	13.01	12.98	11.31
Shallow Features	21.24 %	20.18 %	12.25	11.64	24.30
Deep Feature	18.78 %	16.92 %	11.31	10.62	36.15

In summary, using various preprocessing, machine learning and feature engineering techniques, we reduced WAPE test result from the benchmark of 35.9% to 18.78% which is a remarkable result for this aggregation level. Meanwhile, we succeeded to keep the processing

time within the acceptable range according to the transportation company's time constraints and limitations.

4.5 Conclusion

Demand forecasting is a crucial part of any traditional revenue management system. In this research, we addressed the problem of demand forecasting in the context of railway industry.

To gain multipurpose forecast information, we tackled this problem using two different aggregation levels of data: level I and level II, with the former being the top level that provided an overall view of the data and the latter a more complex level because of having an additional dimension of booking period. Dealing with lower degree of complexity in level I, we achieved our desired results by performing proper preprocessing steps and applying ensemble tree-based methods.

In level II, however, outstanding results were obtained by combining a wide variety of preprocessing, machine learning and feature engineering techniques. We not only used various state-of-the-art machine learning methods, but also developed two different types of heuristic features; namely, shallow features and deep features. The former aims to discover shallow characteristics of data, while the latter is dedicated to extract more complex information.

We realized that having proper and accurate data clusters as features could significantly reduce the forecast error. The deep attribute is mainly constructed based on this discovery. We successfully reduced the forecasting test WAPE result of level II data from the benchmark of 35.9% to 18.78%, while keeping the processing time and overall model performance in a reasonable range.

CHAPTER 5 ARTICLE 2 : DEEP REINFORCEMENT LEARNING APPROACH TO CUSTOMER CHOICE-BASED SEAT INVENTORY CONTROL PROBLEM

Chapter Information : An article based on this chapter is submitted to Computational Management Science for publication. Authors: N. Etebari Alamdari, and G. Savard.
In this paper, we train a goal-oriented agent to offer the most profitable set of products at each time step of the booking horizon with the objective of maximizing firm's revenue in a choice-based seat inventory control problem.

ABSTRACT

Revenue Management (RM) is the application of disciplined analytical tactics that maximizes revenue growth through forecasting each individual customer's choice behavior dynamically [2]. Choice-based RM mainly deals with the question that "Which subset of products the firm should make available to the customers at any given time in order to maximize the total revenue of the firm while considering the customers' choice behavior into account?". In this paper, we explore seat inventory control problem in airline industry. We have a sequential decision making problem which we formulate as a Markov Decision Process (MDP) and find a promising solution for using Deep Reinforcement Learning (DRL) rather than relying on exact theoretical methods or traditional Operations Research (OR) models. We achieved well-performing results and considerably high expected revenue using a DRL method, named Deep Q-Network (DQN). The main difference between this study and similar ones carried out previously in which DRL was selected as the solution method, are the steps taken towards integrating more real-world issues into the proposed approach; finding best set of products to offer to customers at any given time instead of a binary accept-reject action for a given demand and considering larger size flight network examples.

Key words: Deep Reinforcement Learning, Revenue Management, Seat Inventory Control, Customer Choice Behavior

5.1 Introduction

Originally, RM has its roots in airline industry. The development of RM started in 1970s right after deregulation of airline markets in the United States. As a consequence of this act, a new phase began in airlines industry in which the competition among companies forced them to develop surviving tactics [27]. RM is defined as a combination of various analytical and mathematical approaches to address these problems with the ultimate aim of maximizing the firm's revenue.

One of the active research areas in RM is the problem of seat inventory allocation to stochastic demand. Although this problem, particularly in airline industry, has been extensively studied over the past few decades, there are multiple factors that have made this topic more challenging in the recent years. For example, two important role playing parameters are the growth of low-fare airlines and customers' increasing use of Internet search engines to find minimum available fare options. These factors result in more competition which requires better inventory control and pricing policies [65].

The initial theoretical model for capacity control problem was proposed by Littlewood in 1972 [16]. According to this method, if the expected revenue of selling the same seat at the higher fare exceeds the certain revenue of selling another low fare seat, then the low fare class should be closed [8]. Throughout the years, various approaches have been proposed to upgrade Littlewood's method by relaxing some of its constraints and improving the final results.

Expected Marginal Seat Revenue (EMSR) is a heuristic probabilistic revenue maximization model introduced by Belobaba in 1987 [17], which uses historical demand data and average fares to determine the booking limits of each fare class. The revised version of this model, named EMSR-b, performs based on aggregation of demand and weighted average revenues [18].

Until around 2004, the majority of proposed methods was based on the idea of independent demands, in which, demand of any product was assumed to be independent of the availability of other products. More clearly, a customer would either purchase the offered product or buy nothing. However, due to the increased competition caused by low-fair carriers and online travel agencies, this assumption results in unrealistic predictions [21]. The choice-based RM, on the other hand, mainly deals with the question that "Which subset of products the firm should offer to the consumers at any given time in order to maximize the total revenue of the firm while considering the customers' choice behavior into account?". This is the question we address in this research.

This problem is formulated as a Markov Decision Process (MDP) and solved by Dynamic Programming (DP) as an exact method. However, as DP suffers from the curse of dimensionality when applied to real-size problems, approximation techniques are required to overcome the drawback. Within this framework, Talluri et al. [27] published one of the early studies on choice-based RM. Where, they provided a complete characterization of an optimal policy under a general discrete choice model of customer behavior.

Gallego et al. [30] developed a customer choice-based linear programming model for network revenue management, which was culminated in an interesting study by Van Ryzin et al. [31] in which they prove that when capacity and demand are scaled up proportionately, revenue obtained under choice-based deterministic linear programming converges to the optimal revenue under the exact formulation. An extension on this study was provided by Bront et al. [32], where, they assume customers belong to overlapping segments.

Hosseinalifam et al. [25] developed a nonparametric model for choice-based revenue optimization along with corresponding algorithmic framework to solve practical large-scale problems. More recently, a new approximation model named product-closing is proposed by Barbier et al. [66] that is specifically designed for nonparametric demand with promising results. There are several comprehensive review papers on various customer choice behavior models including [67], [68] and [21]. Moreover, in a recent publication, Azadeh et al. [69] studied the impacts of customer behavior models on revenue management systems.

In this paper, we explore the application of Reinforcement Learning (RL) in choice-based RM. RL, as described by Sutton et al. [5], is learning how to map situations to actions in order to maximize the total reward signal. The key feature is that a goal-directed agent interacts with an uncertain environment. In our problem settings, customers are our environment and the airline firm is the goal-oriented agent who makes inventory allocation decisions. Various applications of RL algorithms in RM which are studied before 2007 are published in [70].

Q-learning, as a well-known RL algorithm, has been explored in pricing and some other RM problems: Raju et al. [71] applied it on an electronic monopolistic retail market to find the optimal prices and Chinthalapati et al. [72] also studied Q-learning in an electronic retail market; however, they considered different set of assumptions. Other related examples include [73], [74] and [75].

Recently, a new category of algorithms, named Deep Reinforcement Learning (DRL) has been developed by integrating deep learning into RL. One of the practical examples of DRL is called Deep Q-Network (DQN), which is a combination of deep learning and Q-learning [6]. In the inventory control problem settings, DQN relies on the available historical data as well as real-time interaction with customers to find the optimal policy. Moreover, DQN uses

function approximation to generalize the state-action values to unseen states for problems with a large state space.

In a recent study, Shihab et al. [33] successfully addressed seat inventory allocation problem using DRL by taking into account both booking cancellations and overbooking. Their action space, however, consists of binary yes-no response to a given demand without considering customers choice behaviors and the results are achieved using only a single origin-destination example.

In this paper, we focus on choice behaviors and practical network flight examples. We use simulation to consider the stochastic nature of demand in both training and evaluation of our approach. In so doing, we follow the customer choice behavior explained by Bront et al. [32]. We examined our method on two practical small network and parallel flight examples rather than on a single origin-destination case provided in the literature for similar approaches. We obtained promising performance using DQN in comparison with the results of classical exact method, which is considered as the optimal benchmark.

The main contributions of this paper are:

- solving a choice-based seat inventory control problem using DQN.
- training an agent as a real-time decision maker to offer a set of products at each time step of the booking horizon in order to optimize the total revenue. That is, we take into account the customers choice behavior rather than considering a binary accept-reject scenario for a given request. The agent learns through direct interaction with its environment.

The remainder of this paper is organized as follows: Section 5.2 provides problem description, where, general definitions and problem settings are explained in detail. In Section 5.3, we describe how DQN works in our problem settings. Section 5.4 provides numerical results of applying DQN to two different examples: parallel flights and small network. Finally, concluding comments and possibilities for further research are outlined in Section 5.5.

5.2 Problem Description

We start this section with introducing general definitions (5.2.1) and explaining the customer choice-based seat inventory control problem's model set up and notation (5.2.2). The section will be continued by providing some theoretical background on the building blocks of our approach including Markov Decision Processes (5.2.3), Reinforcement Learning and Q-learning (5.2.4) and finally, Neural Networks (5.2.5).

As a general framework in choice-based seat inventory control problem, we assume maximum one customer shows up with a certain probability at each time step during the booking horizon. Depending on the selected choice behavior, the customer either purchases a product from the set of offered products by the firm or leaves without any purchase. The objective is to find the most profitable set of products to offer to the customers at each time step such that the overall revenue is maximized.

5.2.1 General Definitions

Note that although we consider airline industry the representation of our problem, the explored methodologies can easily be extended to other transportation or hospitality businesses. We will refer to the following definitions throughout this paper:

- Market: an origin-destination pair between which the passengers wish to travel,
- Itinerary: a specific sequence of legs on which passengers travel from their origin to their ultimate destination,
- Fare class: different prices for the same itinerary, usually distinguished from one another by the set of restrictions imposed by the firms,
- Product: an itinerary and fare class combination,
- Consideration set: a subset of products that customer considers as probable choices,
- Offerset: a subset of products that firm makes available to the customers at any given time, and
- Resource: each flight leg with its corresponding capacity.

5.2.2 Customer Choice-based Seat Inventory Control

To address the seat inventory control problem, we follow the notation and model set up provided by Bront et al. [32]. Time $t \in T = \{1, 2, \ldots, |T|\}$ runs forward in discrete increments. Here, we assume the problem to be of a discrete finite time horizon. The set of products and resources are specified by $j \in J = \{1, 2, \ldots, |J|\}$ and $i \in I = \{1, 2, \ldots, |I|\}$, respectively. The revenue vector and capacity vector are $r = (r_1, r_2, \ldots, r_{|J|})$ and $c = (c_1, c_2, \ldots, c_{|I|})$, respectively.

An incidence matrix A of dimension $|I| \times |J|$ with binary elements of α_{ij}, which are defined below, specifies the use of resources by products.

$$\alpha_{ij} = \begin{cases} 1, & \text{if resource } i \text{ is used by product } j, \\ 0, & \text{otherwise.} \end{cases}$$

The state of the network is given by a vector $x = (x_1, x_2, \ldots, x_{|I|})$. By selling one unit of product j, the state of the network updates to $x - A_j$ and the obtained revenue would be r_j. We define customers' segments $l \in L = \{1, 2, \ldots, |L|\}$ based on customers' time, price and itinerary preferences. Each customer belongs to a segment l, and each segment has a consideration set Γ_l, which contains the preferred set of products by segment-l customers. Within each time step t, where the maximum number of customers showing up and that of products to purchase is one, the firm decides which subset $S \in J$ of products to offer to customers. Having $|J|$ products, there are $2^{|J|} - 1$ possibilities of non-empty subsets.

If λ denotes the arrival rate at time t and p_l shows the probability that an arriving customer belongs to segment l, then, the customer arrival process is described by $\lambda_l = \lambda p_l$. Moreover, $P_j(S)$ specifies the probability that an arriving customer chooses product $j \in S$, so, the no-purchase probability would be $P_0(S) = 1 - \sum_{j \in S} P_j(S)$.

We use Multinomial Logit (MNL) model to demonstrate the choice behavior of arriving customers. MNL model is a category of discrete choice models which denotes the probability of choosing an option in the presence of other alternatives. In this paper, we follow the customer choice behavior as explained by Bront et al. [32].

The probability that an arriving customer chooses product $j \in S$ is calculated as $P_j(S) = \sum_{l=1}^{L} p_l P_{lj}(S)$. Where, $P_{lj}(S)$ specifies the probability that an arriving customer from segment l purchases product $j \in S$. Based on MNL assumptions, this probability is computed as follows:

$$P_{lj}(S) = \frac{v_{lj}}{v_{l0} + \sum_{h \in \Gamma_l \cap S} v_{lh}} \tag{5.1}$$

where, v_{lj} is the value given to product j by a segment-l customer. The decision that firm should make at any time t during the booking horizon is to choose the best offerset S to offer to the customers so it can gain the optimal revenue.

To address seat inventory control problem we formulate it as an MDP.

5.2.3 Markov Decision Processes

The Markov property states that "the future is independent of the past given the present." If the next observation and reward depend only on the current observation and action, then we are dealing with a Markov Decision-making Process (MDP) [76].

$$P(x_{t+1}, r_{t+1}|x_1, a_1, r_1, \ldots, x_t, a_t, r_t) = P(x_{t+1}, r_{t+1}|x_t, a_t) \tag{5.2}$$

An MDP is a tuple (X, A, P, R, γ), where, X is the state space, A is a finite set of actions, P is the state transition probability function that denotes the distribution over the next state x', R is the reward function that specifies the expected reward for a given state transition, and finally, $\gamma \in [0, 1]$ is a discount factor [77].

We define a deterministic policy π as a map from a state to an action $\pi : x \rightarrow a$. An MDP is "solved" when we find the optimal policy. There are numerous methods to solve an MDP, including classical Dynamic Programming (DP) which results in an exact solution. However, most of the real-size MDPs are not solvable with DP due to the curse of dimensionality. Value-based approximate methods are one of the solutions to tackle this issue.

Value-based solution methods rely on two main concepts to solve an MDP, value function $V^{\pi}(x)$ and action-value function, also known as Q-function $Q^{\pi}(x, a)$. The value function is the expected return from state x under policy π.

$$V_t^{\pi}(x) = \mathbb{E}_{\pi}[\sum_{k=0}^{|T|-t} \gamma^k r_{t+k+1}|x_t = x] \tag{5.3}$$

Q-function is the expected return assuming the agent is in state x and performs action a and then follows policy π until the end of the episode (*i.e.,* booking horizon).

$$Q_t^{\pi}(x, a) = \mathbb{E}_{\pi}[\sum_{k=0}^{|T|-t} \gamma^k r_{t+k+1}|x_t = x, a_t = a] \tag{5.4}$$

Thus, the optimal action-value function $Q^*(x, a)$ can be achieved as follows:

$$Q^*(x, a) = \max_{\pi} Q^{\pi}(x, a) \tag{5.5}$$

Moreover, $Q^*(x, a)$ satisfies the following Bellman Equation:

$$Q^*(x, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(x', a')|x, a] \tag{5.6}$$

Note that the goal is to find an optimal policy $\pi^*(x, a)$. Based on the above Bellman equation, if the optimal value at state x' of the next time step is known for all values of actions a' , then the optimal policy would be selecting an action a' that maximizes the value of $r + \gamma \max_{a'} Q^*(x', a')$ [77].

5.2.4 Reinforcement Learning and Q-Learning

Reinforcement learning is the study of sequential decision-making processes in natural and artificial systems. According to Sutton et al. [5], the decision-making object is referred to as an agent and everything outside the agent is called its environment. At time step t, the agent in state $x_t \in X$ performs an action $a_t \in A$ based on its behavior policy π. Afterwards, the environment provides a feedback signal as a reward $r_{t+1} \in R$. This sequence of actions, observations and rewards creates the agent's experience. The objective of reinforcement learning is to increase the future reward of the agent given its past experience [77].

Q-Learning [78] is a well-known incremental RL algorithm through which agents can learn how to act optimally in a controlled Markovian domain. At each step, the quality of each possible action of each single state improves successively until convergence to an optimal state-action value. Watkins et al. [78] proved the convergency of Q-Learning to the optimal Q-value with probability 1 under the assumption that all actions would be repeatedly sampled in all states and the action-values are discrete.

In the simplest form, Q-Learning starts with creating a Q-matrix (table lookup) for all states and their corresponding actions and initializes all the Q-values with arbitrary fixed values. The Q-matrix shows the agent's accumulated knowledge of its environment. At each iteration, the matrix is updated based on the weighted average of the old Q-values and the new information received through interacting with the environment. The process ends when agent reaches a terminal state. This basic approach, however, is totally impractical, since the action-value function is estimated separately for each pair, without any generalization.

This issue can be addressed by integrating function approximation in Q-learning. State-action value function approximation speeds up the learning process by generalizing previous experiences to similar unseen state-actions. Table lookup is a particular case of linear function

approximation; however, we want to identify nonlinearities.

We need an approximator which receives a state x and returns approximate Q-values for all possible actions from state x. In other words, we want to find a parameter vector θ such that the error between the approximate Q-value $Q(x, a, \theta)$ (*i.e.,* the predicted value), and the target Q-value is minimized. One of the widely used differentiable function approximation candidates is neural network method.

5.2.5 Neural Networks

A Neural Network (NN) [79] is a computational system which predicts an output variable as a function of the inputs. It consists of an input layer, hidden layer(s), and an output layer. All the layers are made of neurons which are considered connection points in a NN. Although it could vary, a neuron usually calculates the weighted average of its input, and integrates nonlinearity by passing this sum through a nonlinear function, such as sigmoid, ReLU and Tanh, and finally returns a single output. This nonlinear function is called an activation function and it allows NN to address complex systems.

A fully connected feed-forward NN is a neural network in which connections between the units do not form a directed cycle and there is a complete connection between the adjacent layers. A common approach to train a fully connected feed-forward NN is backpropagation which computes the gradient of the loss function with respect to the weights of the network [80].

A NN with multiple hidden layers is called a Deep Neural Network (DNN). DNN is a powerful learning method that can model complex nonlinear relationships.

If we integrate DNN into Q-learning algorithm as a function approximator, instead of a table lookup, we will achieve an advanced RL algorithm called Deep Q-Network (DQN) which is perfectly capable of addressing problems with large state spaces. In the next section, we provide the steps of DQN algorithm and its application as our problem solution.

5.3 Solution Methods

This section comprehensively explains the steps towards applying DQN, as the core of our solution method, to a choice-based seat inventory control problem.

5.3.1 DQN Algorithm

Algorithm 3 demonstrates the details of DQN process [6], which is an iterative algorithm with the objective of finding the optimal policy. At each time step, the agent interacts with the

stochastic environment and receives a sequence of observations, actions and rewards. These information are stored in a temporary memory and a random batch of them are fed to the decision making unit as the training data. By the end of the process, the agent is trained to make the most profitable decision at each time step.

Algorithm 3 Deep Q-Network

procedure
  Initialize replay memory D to capacity N
  Initialize action-value function Q with random weights θ
  Initialize target action-value function $\hat{Q}$ with weights $\theta^{'} = \theta$
  for episode = 1, ..., M **do**
    for t = 1, ..., T **do** ▷ T is the terminal state.
      With probability ϵ select a random action a_t
      Otherwise, select $a_t = \text{argmax}_a\, Q(x_t, a; \theta)$
      Execute action a_t and observe reward r_t and next state x_{t+1}
      Store trajectory (x_t, a_t, r_t, x_{t+1}) in D
    Sample random mini-batch of trajectories (x_j, a_j, r_j, x_{j+1}) from D
    Set

$$y_j = \begin{cases} r_j & \text{for terminal } x_{j+1}, \\ r_j + \gamma \max_{a'} \hat{Q}(x_{j+1}, a'; \theta^{'}) & \text{otherwise.} \end{cases}$$

    perform gradient descent on $(y_j - Q(x_j, a_j; \theta))^2$ w.r.t to θ
    Every ω steps reset $\theta^{'} = \theta$
  Return computed final policy

At each time step, the agent decides either to exploit the best possible action based on the available Q-values or to explore a new action randomly. This exploitation-exploration trade-off poses a fundamental dilemma in RL.

A strategy based only on exploitation would probably miss out on achieving more rewarding policies. Exploring, however, can gather more information and discover new better-rewarding state-actions although it may not be possible to explore all state-action pairs in large state and/or action spaces. Overall, the best long-term strategy may involve short-term sacrifices.

To keep such a balance between exploration and exploitation, DQN relies on ϵ-greedy algorithm. That is, at time step t, it either chooses to exploit the approximated best action $a_t = \text{argmax}_a\, Q(x_t, a; \theta)$ with probability $1 - \epsilon$ or selects a random action with probability ϵ.

Moreover, a technique named experience replay is used to gather data for training DNNs. At each iteration, a trajectory of MDP (*i.e.,* (x_t, a_t, r_t, x_{t+1})) is stored in a replay memory D. Only a mini-batch of the stored experiences are randomly sampled to feed into DNNs

as observations. Experience replay, thus, prevents DNN from being trained by temporally correlated data and results in stability of DQN [6].

Besides, in order to optimize the performance of DQN, two different DNNs are used: a policy network and a target network. The target network is responsible for providing an unbiased estimation of the target value $r_j + \gamma \max_{a'} \hat{Q}(x_{j+1}, a'; \theta')$ which will be used to train the policy network. Here, $\gamma \in [0, 1]$ is the discount factor which determines the importance of the expected future rewards compared to that of the immediate ones. The target and policy networks are synchronized after each certain number of steps ω which causes a coupling between the two networks [81].

In summary, DQN starts to interact with its environment according to ϵ-greedy algorithm. That is, the agent either makes the best decision based on the information received from the policy network or acts randomly. The trajectories are stored in the replay memory and random samples of them (*i.e.,* mini-batch) are fed into DNNs as training data.

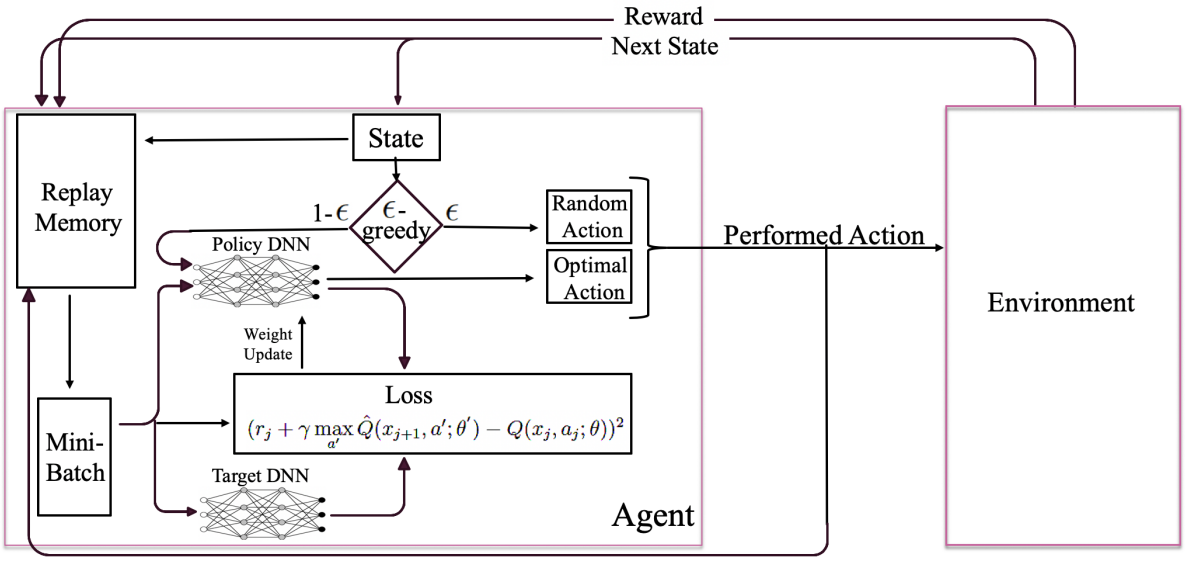

Figure 5.1 Deep Q-Network Algorithm Flowchart

Throughout the process, the policy network adjusts its weights to optimize the predicted Q-values according to target Q-values provided by the target network. After each ω (hyper-parameter) steps we update the weights of the target network based on the policy network.

This process repeats for as many episodes as required to provide us with the best performing policy and its corresponding total reward. Figure 5.1 illustrates the details of DQN algorithm.

5.3.2 Application Process

In this section, we describe how the agent learns to make decisions at each time step of the booking period using simulation data.

A state x is defined as a vector specifying the quantity of remaining capacity in current resources with one dimension for each resource. An action a is a vector specifying a set of offered products. The action space A includes $2^{|J|}$ possible combinations of products to offer, including no offer, while the reward is the immediate revenue gained by selling a product.

At each iteration of DQN algorithm, the agent observes a state x_t and makes an action a_t. Subsequently, the environment transitions to a new state x_{t+1} and sends a feedback (reward) r_t to the agent. In other words, at each time step, the firm offers a set of products and the arrived customer either buys one of these products or leaves without any purchase. The customer's decision is used as a feedback for the firm to make more informed decisions in the future.

In customer choice-based inventory control problem, the environment is a stochastic one; that is, the agent's action does not uniquely determine the outcome. For the same action (offerset), the environment (customer) may choose a different product each time, resulting in a different next state and reward. To capture the stochasticity of the environment, we create a simulator based on the customer choice behavior modeling paradigm.

Each episode of DQN starts from the initial booking period and ends when it meets the stopping criteria (i.*e.* either products are all sold or we reach the departure date). In the training process, to strike a balance between exploration and exploitation, we rely on ϵ-greedy search algorithm. Moreover, we integrate replay memory, target network and policy network in training DQN to stabilize our model as explained in Section 5.3.1.

5.4 Numerical Results

In this section, we assess the performance of DQN on both classic "Parallel Flights" and "Small Network" examples. We compare DQN performance with that of various control

policies including exact and simulated methods. The Upper Bounds (UB) are obtained using Choice-based Deterministic Linear Programming (CDLP) exact method. The expected revenues, used for comparison purposes, are the result of simulating the control policies which are obtained by CDLP and bid price approaches.

CDLP is one of the most well-known methods in network RM that addresses the question that "Which alternatives should the firm make available to the customers in order to profitably influence their choices?" [30]. The details of CDLP algorithm are provided in [31].

Moreover, bid price control policy is another prominent approach in the airline industry to operate inventory control management. This method mainly performs based on computing the marginal value of every single unit of capacity and using it to determine if we should accept or reject an arriving customer [82].

The computational results have been carried out on a 2.21 GHz 7-core computer with 16 GB of RAM. The codes are written in Python 3.6.4 and the environment FICO Xpress-Mosel 7.2.1 has been used to solve CDLP model.

5.4.1 Parallel Flights Example

For this example, we rely on the generalization of original parallel flights example [83] provided by Bront et al [32]. As illustrated in Figure 6.4, we have a network of three parallel flights; namely, leg1, leg2, and leg3, which correspond to morning, afternoon and evening flights, respectively. Two fare classes are available ; High (H) and Low(L). Consequently, there are six products in total and $2^6 = 64$ possible actions (offersets).

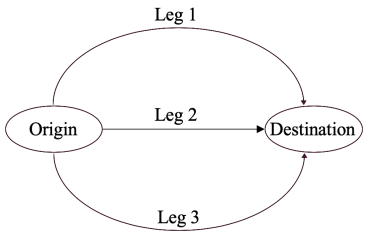

Figure 5.2 Parallel Flights Example.

With the initial capacities being $c = (30, 40, 50)$, the booking starts 300 time steps before departure date ($T = 300$). In the meantime, the customer arrival rate is considered to be 0.5 which leads to an average of 150 customers per episode, assuming a maximum of one show up per time step. The customers are described by their time and price preferences; this leads to four different segments, each with its own specific consideration set. Tables 6.2 and

6.3 show product description and customers segmentation, respectively. This information is used to simulate the environment (customer's choice behavior) by Monte Carlo simulation.

Table 5.1 Product description for the parallel flights example

Product	Leg	Class	Fare
1	1	L	400
2	1	H	800
3	2	L	500
4	2	H	1000
5	3	L	300
6	3	H	600

Table 5.2 Customer segmentation for the parallel flights example

Segment	Consideration set	Pref. vector	λ_l	Description
1	{2,4,6}	(5,10,1)	0.1	Price insensitive, afternoon preference
2	{1,3,5}	(5,1,10)	0.15	Price sensitive, evening preference
3	{1,2,3,4,5,6}	(10,8,6,4,3,1)	0.2	Early preference, price sensitive
4	{1,2,3,4,5,6}	(8,10,4,6,1,3)	0.05	Price insensitive, early preference

The stream of demand arrivals starts at time period $t = 0$, and we determine which set S to offer based on the chosen inventory management method. With the given offerset S, the customer buys one unit of product $j \in S$ with probability $\lambda P_j(S)$, where λ is the arrival rate. That is, the customers arrive during the booking horizon according to a Poisson process with the rate of λ. There is a maximum of one arrival per each period and in the case of an arrival, she belongs to segment l with the probability of p_l.

Finally, the customer's choice is made among products in her consideration set according to their utility vector compatible with multinomial logit discrete choice model. If the customer chooses to leave without a purchase, we move to the next time period with the same capacity level. Otherwise, one unit of capacity from product j's associated leg i would be subtracted and the process will be repeated all over again with the new offerset until reaching a stop criterion.

Regarding DQN, the implementation is done using Keras [84], which is an open source Python library. Our DNNs have two hidden layers with 21 ReLU-activated neurons each. The input layer receives the remaining capacity of each leg as its states, and this results in three input neurons. The output layer contains 64 neurons, one for each Q-value of a possible action. In order to consider the effects of capacity size on our seat inventory control approach, we use

parameter α to scale all leg capacities. For example, $\alpha = 1.2$ means the initial capacity of $1.2 \times (30, 50, 40) = (36, 60, 48)$, and $\alpha = 1$ results in the original base case.

The upper bound of revenue is obtained using linear programming approximation. As reported by Van Ryzin et al. [31], this approach converges to the optimal revenue achieved by DP formulation when changes in capacity and demand are proportional. Moreover, we examine our model under two different no-purchase weight vector assumption, $v_0 = (1, 5, 5, 1)$ and $v_0 = (1, 10, 5, 1)$, to show the effects of differences in customers' purchase behavior on the results. Each weight vector has four elements, one for each segment of customers.

Table 5.3 Comparison of the obtained revenue for the parallel flights example with 95% CI

α	v_0	UB	CDLP	CDLP Bid Price	DQN
0.6	(1, 5, 5, 1)	56,884	54,156 ± 234	55,144 ± 128	55,254 ± 241
	(1, 10, 5, 1)	56,848	54,003 ± 225	55,191 ± 152	55,302 ± 513
0.8	(1, 5, 5, 1)	71,936	67,815 ± 237	67,515 ± 518	69,355 ± 521
	(1, 10, 5, 1)	71,794	67,853± 226	66,918 ± 532	67,522 ± 721
1	(1, 5, 5, 1)	79,155	75,700 ± 253	70,034 ± 718	77,323 ± 658
	(1, 10, 5, 1)	76,866	73,753± 257	67,515 ± 708	73,907 ± 851
1.2	(1, 5, 5, 1)	80,371	79,012 ± 320	71,797 ± 1062	79,450 ± 711
	(1, 10, 5, 1)	78,045	76,978 ± 335	69,630 ± 1135	77,014 ± 902

Revenue results for the parallel flights example are illustrated in Table 5.3. We compared our results with the UB, as well as the expected revenue obtained by both CDLP and CDLP bid prices models [32] and [82].

The results demonstrate that DQN outperforms both CDLP and CDLP bid prices policy in almost all cases; i.e., with different α parameters and v_0 coordinate values. Differences are more significant in the highly constrained capacity cases (*i.e.,* $\alpha = 0.6$ and 0.8). In such scenarios, decision making is more challenging as it has a smaller margin of mistake in resources allocation. This is when DQN is more rewarding than other methods and stands closer to the UB. For the original capacity and ample capacity case (*i.e.,* $\alpha = 1.2$), all choice-based methods perform reasonably good; however, DQN still shows higher mean expected revenue.

This mainly happens due to the advantage that DQN has as a real-time decision making method as it makes each decision at each time step based on the real-time feedback received from the environment. Each arrived customer's choice directly affects the next offerset. On the other hand, modification and re-optimization at each time step is impossible when using CDLP-based methods because of computational costs involved. Thus, CDLP-based methods mainly rely on their initial estimations with the possibility of occasional re-optimization.

The results also reveal the effects of changes in customers' willingness to pay on the final revenue. Higher tendency of customers towards no-purchase option (*i.e.,* the cases with higher weights in the preference vector) leads to the expected revenue decrease.

We train DQN using 2000 episodes. The trained model, then, is used to interact with the customers for another 2000 episodes to obtain the simulated results. Moreover, the processing time of training phase is under 30 minutes while simulation step is less time consuming as it takes less than 10 minutes. The size of the replay memory and mini-batch are 2000 and 100, respectively.

Figure 5.3 shows the learning phase of our DQN method for $\alpha = 0.6$. Note that we observe the same learning trend for other values of α. The figure demonstrates the rolling mean value of the expected revenue achieved from the last 50 episodes.

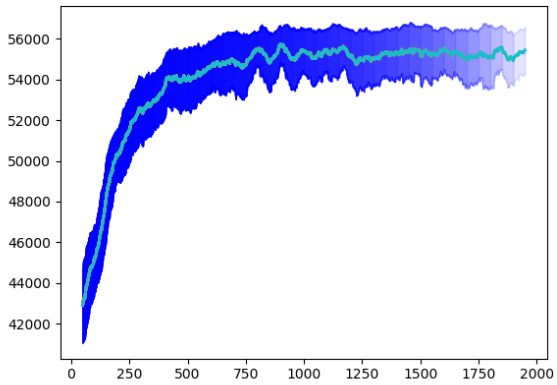

Figure 5.3 Expected revenue increase based on the number of episodes in the training phase of DQN for $\alpha = 0.6$

5.4.2 Small Network Example

In this example [32], illustrated in Figure 5.4, we consider a small airline network consisting of three cities and three legs (flights) with capacities of 100, 50, and 50 for leg 1 to 3, respectively. Moreover, we assume that $T = 375$ time periods.

Each itinerary has low and high fares. The combination of class, fare and origin-destination defines 8 different products hence $2^8 = 256$ different possible offersets (actions), including no product offer. Table 5.4 presents the description of the products. The customers are described by their route and price preferences; this categorizes them in 5 segments, each with its corresponding arrival rate. The details of customer segmentation are provided in Table

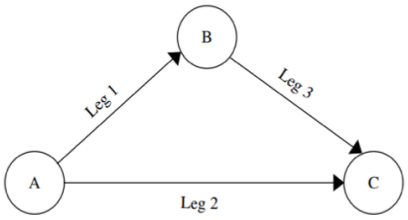

Figure 5.4 Small network example

5.5. The implementation of DQN is done using Keras, and our DNNs consist of two hidden layers with 28 ReLU-activated neurons each. The size of the replay memory and mini-batch are 2000 and 100, respectively.

Table 5.4 Product description for the small network example

Product	Origin-Destination	Class	Fare
1	A → C	H	1,200
2	A → B → C	H	800
3	A → B	H	500
4	B → C	H	500
5	A → C	L	800
6	A → B → C	L	500
7	A → B	L	300
8	B → C	L	300

Table 5.5 Customer segmentation for the small network example

Segment	λ_l	Consideration set	Pref. vector	Description
1	0.15	{1,5}	(5,8)	Price sensitive, Nonstop (A → C)
2	0.15	{1,2}	(10,6)	Price insensitive, (A → C)
3	0.2	{5,6}	(8,5)	Price sensitive, (A → C)
4	0.25	{3,7}	(4,8)	Price sensitive, (A → B)
5	0.25	{4,8}	(6,8)	Price sensitive, (B → C)

Table 5.6 shows the results of applying DQN to small network example to solve the seat inventory control problem. The network example is considered to be more challenging compared to parallel flights case due to availability of shared resources. More clearly, a product can use different resources in network example, and this characteristic makes the process of finding the most profitable offerset more difficult.

In the previous example, we explored the impacts of changes in capacity size on the obtained revenue. Here, we introduce the parameter β which alternates the number of arriving demands. Using which, we assess the performance of our model from another point of view.

Since we have exactly one customer show up per time period, based on the $\lambda = 1$ assumption, parameter β scales the number of time steps (*i.e.,* $\beta = 0.8$ means $375 \times 0.8 = 300$ arriving customers or 300 time steps), too. We also consider various preference vectors in order to explore the impacts of customers' different tendencies to purchase on the final revenue. The model was trained using 2000 episodes with processing time of less than 40 minutes and the simulation results were achieved by running trained DQN for another 2000 episodes within less than 10 minutes.

There is a similar behavior between the obtained results for the network example and that of parallel flights case. DQN outperforms CDLP in all scenarios and the differences are more noticeable in the ample demand case (*i.e.,* $\beta = 1.2$). As for the differences in customers' willingness to pay, the higher the tendency of customers towards no-purchase option, the less the expected revenue.

Table 5.6 Comparison of the obtained revenue for the small network example with 95% CI

β	v_0	UB	CDLP	DQN
0.8	(2, ,5, 2, 2, 2)	114,090	109,520 ± 430	111,216 ± 781
	(5, 5, 5, 4, 3)	106,750	100,760 ± 472	102,432 ± 880
	(6, 8, 6, 6, 7)	101,556	96,264 ± 325	97,871 ± 981
1	(2, ,5, 2, 2, 2)	120,563	114,530 ± 486	115,880 ± 554
	(5, 5, 5, 4, 3)	114,595	109,469 ± 411	112,097 ± 550
	(6, 8, 6, 6, 7)	109,640	104,510 ± 302	106,108 ± 931
1.2	(2, ,5, 2, 2, 2)	130,000	116,497 ± 531	118,983 ± 570
	(5, 5, 5, 4, 3)	122,202	115,103 ± 375	117,452 ± 569
	(6, 8, 6, 6, 7)	118,104	114,077 ± 504	117,050 ± 675

5.5 Conclusion

In this paper, we considered a deep reinforcement learning approach, named deep Q-network, to solve a choice-based seat inventory control problem. We intended to answer the question that "Which offersets should the firm make available at each time step in order to maximize the total revenue achieved during the whole booking period horizon while taking the customer choice behavior into account?"

The problem was formulated as an MDP and solved using DQN. For comparison purposes, UBs were achieved using choice-based deterministic linear programming exact method. More-

over, the simulated results obtained by CDLP and bid price approaches were compared to that of DQN.

The training processes were done through interacting with the environment (customers) and the decisions were made in real-time. For the environment simulation, we relied on the general case where customers belonged to overlapping segments and made their choices based on the multinomial logit model [32].

We mainly focused on learning the customers' behavior through a goal-oriented agent with the objective of maximizing the total revenue. We obtained promising results in both parallel flights and small network examples. Moreover, the consequences of alteration in capacity size, demand stream and customers' willingness to pay on the total revenue were studied.

We realized that DQN works fine when it deals with toy examples; however, in realistic size networks, the problem of substantial computational cost arises. Indeed, as the number of products increases, the number of actions grows exponentially and the problem becomes intractable.

For example, a network of 100 products would have $2^{100} = 1.26e30$ possible actions. It is unreasonably expensive to solve a NN with $1.26e30$ outputs in each iteration. To address this issue, we will introduce an optimization-based heuristic algorithm in our future work. The objective of this method is to solve larger-scale seat inventory control problems with both high performance quality and reasonable processing time.

$2^{100} = 1.26e30$

CHAPTER 6 ARTICLE 3 : DEEP REINFORCEMENT LEARNING BOOSTING VIA ACTION GENERATION TO SOLVE LARGE-SIZE CUSTOMER CHOICE-BASED SEAT INVENTORY CONTROL PROBLEM

Chapter Information : An article based on this chapter is submitted to Journal of Revenue and Pricing Management for publication. Authors: N. Etebari Alamdari, and G. Savard.

In this paper, we address a large-scale network revenue management problem with taking customer choice behavior into account. A heuristic algorithmic framework is proposed to solve the problem efficiently in terms of solution quality and processing time.

ABSTRACT

Nowadays, firms intend to use customer choice-based models instead of an independent demand paradigm to generate more revenue. In this paper, we address choice-based seat inventory control problem with stochastic demand using a deep reinforcement learning technique named Deep Q-Network (DQN). DQN can naturally address large state space problems with its integrated function approximation. However, it becomes intractable in the case of large discrete action space. To address this issue, we propose an Action Generation (AGen) algorithm. AGen is a greedy heuristic algorithm designed to be integrated into DQN to overcome the complexity of the original problem. It aims to greedily generate a set of "effective" actions to replace the original action space. This leads to the main achievement of this study which is to dramatically decrease the complexity of the solution method without negatively affecting its performance in a large-scale choice-based seat inventory allocation problem.

Key words: Deep Reinforcement Learning, Large-scale Revenue Management, Seat Inventory Control, Customer Choice Behavior

6.1 Introduction

Revenue management (RM) is the application of systematic analytical strategies, which maximizes revenue growth by dynamically forecasting each individual customer's choice behavior [2]. RM was initially developed in 1970's right after deregulation of airline markets in the United States. This evolutionary transformation started a new era in airline industry where the close competition among companies forced them to develop survival policies [27].

To make a well-informed and accurate demand management decision, RM models require precise information regarding each customer's choice behavior. Knowing customers' purchase habits gives the firm the power to profitably control the availability of products by redirecting the demand stream towards products with high profit margins.

Nowadays, it is common knowledge that customers make decisions not just based on offered products; rather, they also consider the overall market conditions such as prices offered by competitors. In today's market, the independent demand paradigm, where the customer is unaffected by anything but the availability of her desired product, is no more practical.

Solving choice-based seat inventory control problem involves two main challenges: modeling a customer's choice behavior at a particular moment in time and finding most profitable sets of products to offer at each time step based on the proposed demand model [32]. Both issues have been extensively studied as of 1990's, when the first work on choice behavior in networks was presented by Belobaba et al. [85]. There are several comprehensive review papers on various customer choice behavior models including [67], [68] and [21].

Solving a substantially large RM problem is expensive due to computational expenses of traditional RM or optimization methods. Initially, Gallego et al. [30] addressed this issue by introducing a general framework, named Choice-based Deterministic Linear Programming (CDLP), for efficient computation of the solution. They studied the network RM using flexible products under two circumstances: each product's demand being generated independently and the demand being affected by customer choice model. However, the market demand used in their model suffered from lack of any kind of customer segmentation.

As an extension to the previous study, Van and Liu [31] explored practical applications of the CDLP model. They developed a decomposition heuristic method for using the dual prices of the CDLP in converting the static CDLP solution into a dynamic control policy. Moreover, they demonstrated that as demand and capacity are expanded asymptotically, only a particular set of products are used in the optimal policy. In their market segmentation model, the customers are divided into segments with disjoint consideration sets of products.

Another interesting study is provided by Bront et al. [32], in which, they extend CDLP to

the case of customers with overlapping segments (*i.e.,* consideration sets assigned to different segments have a non-empty intersection). This is the case we consider in this paper for simulating customer behavior. Meanwhile, there are various comprehensive studies that provide a deep insight into the principles for simulations in revenue management such as [86] and [87].

Later on, a nonparametric model for choice-based revenue optimization was developed by Hosseinalifam et al. [25] in which they provided an algorithmic framework to solve practical large-scale problems. More recently, Barbier et al. [66] proposed a new approximation model with promising results, named product-closing, which is specifically designed for nonparametric demand.

In this study, we tackle choice-based seat inventory control problem using Reinforcement Learning (RL) techniques. RL is about learning how to map situations to actions in order to maximize the total reward. That is, a goal-directed agent performs an action in an uncertain environment and receives a feedback which motivates the agent's next move [5]. In the current problem settings, customers are our stochastic environment and the airline firm is the agent who makes seat inventory control decisions.

One of the main goals of RL is to solve complex tasks with high-dimensional nature. In recent years, the advancements in both deep learning and reinforcement learning resulted in development of new models under the category of deep reinforcement learning some of which show human level performance. A well-known example is Deep Q-Network (DQN) [6].

DQN is of great interest to us because of its capability to both handle large state space problems and to be trained with only historical data; that is, it does not require demand prediction. As shown in our recent study [88], DQN demonstrates reasonably good performance when applied to choice-based seat inventory assignment problem. The promising results were achieved in two different scenarios: parallel flights and small network examples. In large-size flight networks, however, DQN becomes intractable because of high computational costs.

Note that in our problem settings, the states are the remaining capacity of each leg at each time step. Moreover, the possible sets to offer at each time step create the action space. As the number of products increases, the action space becomes exponentially large.

Deep Q-Network is a value-based method, which means that we learn a value function which relates a value to each state-action pair. These methods show promising performance when there are limited discrete actions. Deep reinforcement learning is also studied in both continuous and large discrete cases of action spaces. However, the proposed methods mainly fall in the category of policy-based methods (*i.e.,* methods which directly learn approximate

optimal policy) or hybrid methods which merge value-based and policy-based characteristics into a single method called 'actor-critic'.

For example, Lillicrap et al. [89] proposed an actor-critic, model-free algorithm based on the deterministic policy gradient, which can perform in continuous action spaces. They successfully solved more than 20 simulated physics tasks such as the classical cart-pole swing-up. Recently, a comprehensive study titled "Deep Reinforcement Learning in Large Discrete Action Spaces" has been published by Google Deepmind, which tackles the problem using policy gradient methods in an actor-critic framework [90].

Since DQN is a straightforward, easy-to-implement and practical algorithm, we use it to address a large-scale RM problem. We develop a modified version of DQN, specifically designed for RM and capable of perfectly handling both large state and action spaces.

This paper intends to contribute to the application of RL techniques in solving practical RM problems. More specifically, it addresses the problem of large-size choice-based seat inventory allocation by introducing an "Action Generation" (AGen) algorithm. AGen is an iterative heuristic algorithm used to reduce computational complexity emerging as a result of having a large discrete action space. It generates a set of most profitable actions which are chosen through an opportunity cost estimation algorithm. In general, AGen follows the concept of column generation which is an efficient algorithm designed to solve large linear programming formulations [1].

By integrating this algorithm into DQN, we can solve large instances with an exponential number of potential actions using only a restricted number of actions. This results in a tremendous reduction in the computational cost of DQN while maintaining a high performance quality. In this approach, we assume that customers choose their preferred products based on multinomial logit choice behavior modeling paradigm [32]. Finally, we evaluate our solution method on two practical "parallel flights" and "hub and spoke" examples and compare the results to those of various common benchmarks provided in RM literature.

The remainder of this paper is organized as follows: Section 6.2 provides problem description, where, general definitions, required theoretical background, and problem settings are explained in detail. We start Section 6.3 with a brief explanation on how DQN works in our problem settings. The main focus of this section, however, will be on the action generation algorithm. Section 6.4 demonstrates numerical results of applying RDQN to two different examples: parallel flights and hub and spoke. Finally, concluding comments are outlined in Section 6.5.

6.2 Problem Description

In this section, the problem is defined and its theoretical background is explained. First, we describe the model set-up and notation for customer choice-based seat inventory allocation problem. In the rest of the section, theoretical background of Markov Decision Processes, reinforcement learning, and function approximation are elaborated.

6.2.1 Customer Choice-based Seat Inventory Control

Note that throughout this paper, we follow the notation and model set-up provided by Bront et al. [32], which are summarized in Table 6.1 and were extensively explained in our previous study [88].

In our problem settings, a market is an origin-destination pair between which the passengers wish to travel. Within each market, we may have different itineraries which are defined by a specific sequence of legs connecting the origin to the destination. Moreover, fare classes are different prices given for the same itinerary, usually distinguished from one another by the set of restrictions imposed by the firms. A product is the combination of an itinerary and a fare class.

We assume that each arriving customer belongs to a segment and each segment is categorized by its consideration set (*i.e.,* a subset of products that a customer considers possible options to purchase). Finally, an offerset is a subset of products that a firm makes available to the customers at any given time.

Customers choose a particular product based on a Multinomial Logit (MNL) model [91]; their purchase is affected by both their consideration set and the firm's offered set. An MNL model is a class of discrete choice models, which indicates the probability of selecting an option in the presence of other alternatives.

We assume that at each time step t, there is a maximum of one customer show-up and a maximum of one product purchase. Moreover, an arriving customer from segment-l, with consideration set Γ_l, assigns a value (weight) v_{lj} to each product $j \in \Gamma_l$. Consequently, based on MNL assumptions, the probability that a segment-l customer chooses an available product $j \in S$ is defined as

$$P_{lj}(S) = \frac{v_{lj}}{v_{l0} + \sum\limits_{h \in \Gamma_l \cap S} v_{lh}} \tag{6.1}$$

Table 6.1 Notation and definitions of seat inventory control problem

t:	time periods indexing forward; $t \in \{1, 2, \ldots, T\}$
J:	set of products
r_j:	revenue obtained by selling one unit of product $j \in J$
I:	set of resources
c_i:	initial amount of resource $i \in I$
L:	set of customer segments
p_l:	probability that an arrived customer belongs to segment l
λ_l:	arrival rate of a customer from segment $l \in L$; $\lambda_l = \lambda p_l$
λ:	arrival rate of a customer at each time period $\lambda = \sum_{l=1}^{L} \lambda_l$
η_{ij}:	Boolean constant indicating whether resource i is used by product j (*i.e.*, $\eta_{ij} = 1$), or not (*i.e.*, $\eta_{ij} = 0$)
H:	product-resource incidence matrix, or simply, the incidence matrix; $H = [\eta_{ij}] \in \{0, 1\}^{m \times n}$
x:	state of network $x = \{x_1, x_2, \ldots, x_{\lvert I \rvert}\}$; selling one unit of product j results in network state update from x to $x - H_j$
S:	set of offered products by the firm, possibly including the 'null' product; $S \in 2^{\lvert J \rvert}$
Γ_l:	set of products preferred by the segment-l customers
$P_j(S)$:	probability of product $j \in S$ being purchased by an arriving customer if set S is offered
$P_0(S)$:	no-purchase probability given offerset S; $\sum_{j \in S} P_j(S) + P_0(S) = 1$

Clearly, the probability of choosing product $j \in S$ by an arriving customer (*i.e.,* $P_j(S)$) equals

$$P_j(S) = \sum_{l=1}^{L} p_l P_{lj}(S) \tag{6.2}$$

Moreover, $R(S)$ specifies the expected revenue generated from an arriving customer when set S is offered,

$$R(S) = \sum_{j \in S} r_j P_j(S) \tag{6.3}$$

The firm would like to maximize its total revenue by selecting the most profitable set S to offer at each time period t during the booking horizon.

To address seat inventory control problem, we formulate it as an MDP.

6.2.2 Markov Decision Processes

As Puterman [76] describes, a task is a Markov Decision-making Process (MDP) if the next observation and reward depend only on the current observation and action. An MDP consists of a state space X and a finite set of actions A. For each pair of state-action, P is the state transition probability function which denotes the distribution over the next state x',

$$P_a^{xx'} = P(x_{t+1} = x' | x_t = x, a_t = a) \tag{6.4}$$

and R is the reward function that specifies an expected reward given the transition from state x to state x' while performing action a,

$$R_a^{xx'} = \mathbb{E}[r_{t+1} | x_t = x, x_{t+1} = x', a_t = a] \tag{6.5}$$

Thus, an MDP is defined as a tuple (X, A, P, R, γ), where $\gamma \in [0, 1]$ is a discount factor. In a decision-making problem, we would like to know, given each state, which action would lead to the maximum reward in the long run. This map from a state to an action is called a policy $\pi : x \to a$. If we consider the maximum reward as the sum over all expected immediate rewards, the result may not be bounded. Thus, the discount factor γ is considered, which

depreciates rewards at every time step.

We define value function as the expected return from state x under policy π,

$$V_t^{\pi}(x) = \mathbb{E}_{\pi}[\sum_{k=0}^{|T|-t} \gamma^k r_{t+k+1} | x_t = x] \tag{6.6}$$

and action-value function, also called Q-function, as the expected return being in state x and performing action a and then following policy π till reaching the terminal state.

$$Q_t^{\pi}(x, a) = \mathbb{E}_{\pi}[\sum_{k=0}^{|T|-t} \gamma^k r_{t+k+1} | x_t = x, a_t = a] \tag{6.7}$$

The goal is to find an optimal policy π^*, a policy that corresponds to the optimal value function (or optimal action-value function). There are numerous methods to solve an MDP (*i.e.,* to find an optimal policy), including the classical well-known exact method Dynamic Programming (DP). Optimal value function of the seat inventory control problem can be achieved through the following DP formulation,

$$\begin{aligned} V_t(x) &= \max_S \{\sum_{j \in S} \lambda P_j(S)(r_j + V_{t+1}(x - H_j)) + (\lambda P_0(S) + 1 - \lambda) V_{t+1}(x)\} \\ &= \max_S \{\sum_{j \in S} \lambda P_j(S)(r_j - (V_{t+1}(x) - V_{t+1}(x - H_j)))\} + V_{t+1}(x) \end{aligned} \tag{6.8}$$

subject to $V_t(0) = 0$ for $t = 1, 2, ..., T$, and $V_{T+1}(x) = 0 \ \forall x \geq 0$ [31]. Assuming that we have calculated the value function, then, for a given t and x, the optimal offerset would be

$$S^*(t, x) := \operatorname*{argmax}_S \{\sum_{j \in S} \lambda P_j(S)[(r_j - (V_{t+1}(x) - V_{t+1}(x - H_j))]\} \tag{6.9}$$

$V_{t+1}(x) - V_{t+1}(x - H_j)$ specifies the opportunity cost of selling one unit of product j at time step t, having capacity state x [92]. Later on in this study, we develop our heuristic approach based on the concept of opportunity cost.

Unfortunately, due to curse of dimensionality, large-size MDPs (*i.e.,* large state space and/or

action space) cannot be solved using DP. In such cases, it is possible to approximate DP with different approximation methods such as Linear Programming. Here, we will address the seat inventory control problem using deep reinforcement learning.

6.2.3 Reinforcement Learning and Q-Learning

Reinforcement Learning (RL) refers to the study of sequential decision-making processes where learning occurs from the experience generated by an agent interacting with its environment. RL was initially inspired by the training method based on rewarding desired behaviors and/or punishing undesired ones in psychology. RL methods were formalized later using MDP [5].

In RL, learning happens through a sequential process illustrated in Figure 6.1 [5]. At time step t, the agent in state $x_t \in X$ takes an action $a_t \in A$ according to the policy π it follows. As a feedback, the environment provides a reward $r_{t+1} \in R$ and the agent finds itself in a next state $x_{t+1} \in X$. This sequence of actions, observations and rewards is called agent's experience. The fundamental goal in RL is to use the past experience to increase the future reward. To put it differently, the agent seeks to maximize the cumulative reward in the long run [77].

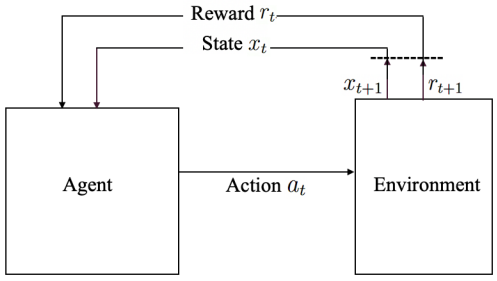

Figure 6.1 Agent-environment interaction in reinforcement learning

One of the well-known RL algorithms is Q-learning in which the learned action-value function incrementally approximates the optimal Q-function. According to Watkins et al. [78], Q-learning converges to optimal Q-value with probability 1 under the assumption that the action-values are discrete and all state-action pairs will be repeatedly sampled.

In its classical form, Q-learning creates a lookup table (also called Q-matrix) to store all the state-action values. The values are initialized to random fix numbers, except for the terminal state which is zero. The Q-values are updated at each iteration of an episode (An episode refers to all the states that occur between the initial and the terminal states.). Thus, the Q-matrix accumulates the agent's knowledge of its environment which gets updated at each

iteration.

In large state and/or action spaces, updates would be extremely expensive due to largeness of Q-matrix. To tackle this issue, function approximation is integrated into Q-learning.

6.2.4 Function Approximation in Q-Learning

Function approximation replaces lookup table in Q-learning by generalizing previous experiences to similar unseen state-action pairs hence speeding up the process. In large and complex problems, such as seat inventory control problem, we need to choose a function approximation which can generalize effectively and capture nonlinearities.

In the context of our problem, function approximation is the mapping from states to values via a parameterized function which uses supervised learning methods to set its parameters (*i.e.,* weights) [93]. That is, the function approximation receives state x as an input and returns approximate Q-values for all possible actions given state x. To do so, it adjusts its weight vector θ to minimize the difference between approximate Q-value $Q(x, a; \theta)$ (*i.e.,* the predicted value) and the target Q-function. This is a regression problem which can be addressed using neural networks.

A Neural Network (NN) is a network of interconnected units, called neurons, which predicts an output variable as a function of its inputs. A NN is composed of an input layer and an output one, as well as some hidden layer(s) in between, all having different numbers of neurons depending on the architecture that the problem may require.

Neurons are semi-linear units, which means each neuron computes a weighted average of its inputs and then applies a nonlinear function to the achieved result. This function is called 'activation function' and it allows NNs to address complex systems. It is usually an S-shaped function (*e.g.,* a sigmoid function); however, sometimes a rectifier nonlinearity is also used. A unit using the rectifier is also called a Rectified Linear Unit (ReLU). The activation functions are parametrized by the connection weights of the network [5].

A neural network with a complete connection between the adjacent layers and no loops within the network is called a fully connected feed-forward NN. Usually a backpropagation method is chosen as the learning algorithm for NNs. As described by Wythoff et al. [80], backpropagation consists of two phases: the forward propagation of the activation and the backward propagation of the error. Like any other supervised learning algorithm, the desired outputs of a NN are compared to the predicted values and the network's weights are adjusted to minimize the difference. More details of backpropagation algorithm can be found in [80].

A NN is called a Deep Neural Network (DNN) if it has more than one hidden layer. DNN is

a powerful learning method for modeling complex data with nonlinear relationships. Figure 6.2 shows a DNN used for function approximation, where it receives a state $x = (x_1, x_2, x_3)$ as its input and returns Q-values for all the possible state-action pairs from state x, assuming there are three actions in total.

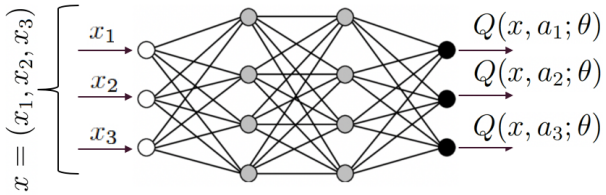

Figure 6.2 Function approximation using a deep neural network

If we use a DNN, rather than a traditional lookup table, for function approximation in Q-learning, we develop a method called Deep Q-Network (DQN) which can handle large state space problems [6].

6.3 Solution Method

We start this section with a brief review of how DQN operates followed by proposing the action generation algorithm. AGen aims to reduce the complexity of DQN in the presence of a large discrete action space.

6.3.1 DQN

Algorithm 4 [6] demonstrates the details of DQN process which is an iterative algorithm used for identifying the optimal policy [88] .

At each iteration, while interacting with the environment, the agent achieves and stores a sequence of observations, actions and rewards. A randomly selected batch of such experiences are then fed into the decision maker as the training data. By the end of the process, the agent is trained to make the most profitable decision at each time step.

In order to improve the stability of DQN in practice, the so-called "experience reply" technique is implemented. At the time step t, a sequence of actions, observations and rewards (x_t, a_t, r_t, x_{t+1}) (i.e, agent's experience) is stored in a replay memory out of which only a mini-batch containing random samples is fed into DNNs as training data hence preventing temporal correlation among training data and thereby assuring enhanced stability of DQN [6].

Algorithm 4 Deep Q-Network

procedure
  Initialize replay memory D to capacity N
  Initialize action-value function Q with random weights θ
  Initialize target action-value function $\hat{Q}$ with weights $\theta^{'} = \theta$
  for episode = 1, ..., M **do**
    for t = 1, ..., T **do** ▷ T is the terminal state.
      With probability ϵ select a random action a_t
      Otherwise, select $a_t = \text{argmax}_a \, Q(x_t, a; \theta)$
      Execute action a_t and observe reward r_t and next state x_{t+1}
      Store trajectory (x_t, a_t, r_t, x_{t+1}) in D
    Sample random mini-batch of trajectories (x_j, a_j, r_j, x_{j+1}) from D
    Set

$$y_j = \begin{cases} r_j & \text{for terminal } x_{j+1}, \\ r_j + \gamma \max_{a'} \hat{Q}(x_{j+1}, a'; \theta^{'}) & \text{otherwise.} \end{cases}$$

    perform gradient descent on $(y_j - Q(x_j, a_j; \theta))^2$ w.r.t to θ
    Every ω steps reset $\theta^{'} = \theta$
  Return achieved final policy

A policy network and a target network are used to ameliorate the performance of DQN; the former requires target values to train and adjust its weights while the latter provides the former with an unbiased estimation of target values. The two networks are coupled following the synchronizations after each certain number of steps [81].

One of the fundamental challenges in RL is the exploration-exploitation trade-off. The objective of exploration is to continually look for new policies, while exploitation aims to tackle the problem using the already well-established solutions. Exploration becomes more vital when the environment is changing because such changeability can cause rewarding solutions depreciate or new solutions to arise over time. Obviously, certain short-term sacrifices are inevitable when formulating the most profitable long-term strategy [94].

In summary, DQN process starts with observation of state x_t. The agent, then, acts ϵ-greedy to strike a balance between exploration and exploitation by taking a random action with probability ϵ or deciding to exploit the best action $a_t = \text{argmax}_a \, Q(x_t, a; \theta)$, with probability $1 - \epsilon$. As we progress, the agent gains more information about the environment; thus, it is a reasonable strategy to increase the probability of exploitation by iteratively reducing the value of ϵ.

In response to the agent's action, the environment returns an immediate reward r_t as the

agent finds itself in the next state x_{t+1}. At each iteration, the acquired experience is stored in the replay memory. Training of DNNs start with feeding them a mini-batch of random samples from the replay memory.

Here, the decision maker is the policy DNN. The weight update of policy network is done using the target value $r_j + \gamma \max_{a'} \hat{Q}(x_{j+1}, a'; \theta')$, where the maximum expected Q-value from the next state is provided by the target network.

Each episode of DQN runs until meeting the stop criteria. The entire process repeats itself for as many episodes as required until eventually the best performing policy and corresponding total reward are achieved [88].

Here are the definitions of the major terms frequently used in our problem settings [88]:

- Stochastic environment: the customers
- Agent: the firm
- State x: a vector which specifies the quantity of remaining capacity in the current resources, with one dimension for each resource,
- Action a: a vector specifying a set of offered products. The action space A includes $2^{|J|}$ possible combinations of products to offer, including an empty action set which represent no products to offer,
- Reward r: the immediate revenue gained by selling a product $j \in S$, and
- Stop criterion: conditions in which either products are all sold or the departure date has arrived (*i.e.,* reaching the end of time periods T).

A simulator of overlapping segments is also created in accordance with the parameters of multinomial choice-based modeling to simulate the stochastic environment of the problem [32]. For the sake of simplicity, we assume that cancellations are not allowed and so overbooking will be redundant.

6.3.2 Action Generation

In a large-size seat inventory assignment problem, there are both a large state space (which is handled with a function approximation technique in DQN) and an exponentially large discrete action space. As an example, when there are 21 products, which is a regular minimum

number of products in airline companies, there will be more than one million possible actions to choose from in each time step.

Analyzing the results of solving a large-size seat inventory control problem using the CDLP method shows that the optimal policy includes only few particular sets of products, which the firm makes available to the customers during the booking horizon.

Therefore, if we can generate these so called "effective" offersets in a way, there will be no need to explore all possible combinations of actions at each iteration. We propose Action Generation (AGen), a greedy heuristic approximation algorithm, to deal with the complexity issue of DQN in this problem settings.

Adopting this approach, we divide the original choice-based seat inventory control problem into a restricted master problem (*i.e.*, DQN with a restricted number of actions, called RDQN) and a subproblem. The reduced master problem (RDQN) holds all the characteristics of the original one, except the action space. The subproblem is designed to generate profitable actions which consist of two phases:

- opportunity cost evaluation
- action generation

Following this algorithm, we progress iteratively until no further "effective" offerset remains to be generated.

The master problem could be considered an evaluation unit that uses DQN to explore the performance of the actions generated by the subproblem. First, we explain the master problem; then, we provide the algorithm for action generation in the subproblem.

In the master problem, we initially define a small restricted set of randomly chosen actions out of the original action space $2^{|J|}$ and call it "restricted set", Ω. We also define Restricted DQN (RDQN) as a DQN with an action space limited to Ω. By solving RDQN, we obtain Q-values associated with given actions at each state of the network. Q-values show how good it is to be at each state and perform a specific action from the limited pool of restricted actions set.

Now, we would like to know adding which set of products to the restricted set is potentially more profitable. To do so, we start with computing approximate opportunity cost of each individual resource ξ_i, $\forall i \in I$, as the first phase of subproblem. In the second phase, using the approximated opportunity costs, we verify whether or not there are any new actions which could increase the expected value function. Figure 6.3 shows a simple flowchart of action generation method.

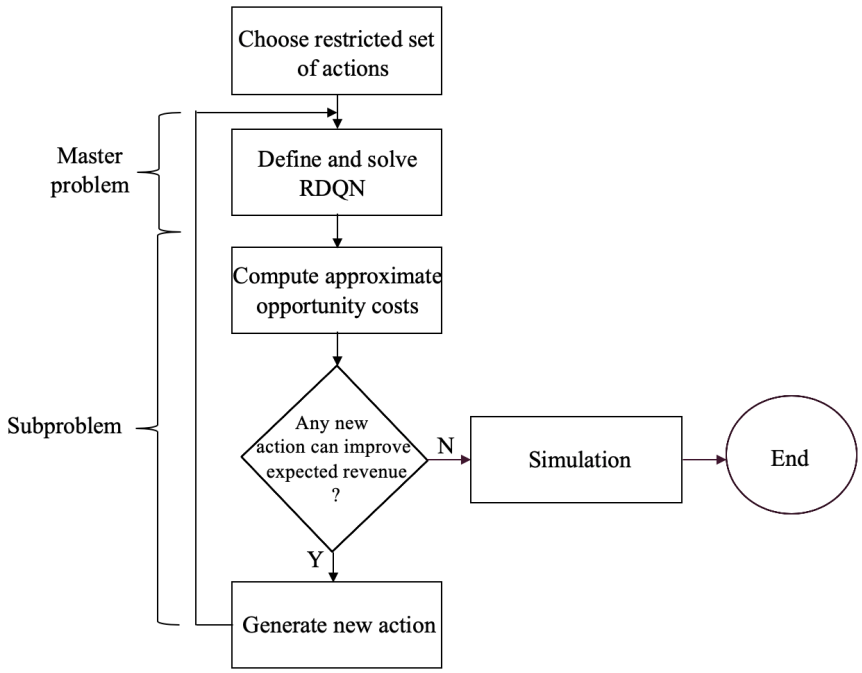

Figure 6.3 Action Generation (AGen) flowchart

Subproblem

Phase 1: Opportunity Cost Assessment

In this phase, we first assess the value of each unit of capacity, and then use the information obtained to evaluate the actual expected net profit of each product. For instance, if selling product j produces a revenue of 100\$, but each seat is worth more than 300\$, then this product is not a good choice to be in the offerset.

Opportunity cost assessment starts with predicting $Q(x, a)$, given the initial state x (*i.e.,* all resources are available) and actions a which belong to the restricted set of randomly chosen actions Ω. Having Q-values, we can compute the approximate value function according to the following equation:

$$\tilde{V}(x) \approx \max_{a} Q(x, a) \qquad \forall a \in \Omega \tag{6.10}$$

Note that although this is just an approximation, it is good enough for the purpose of computing opportunity costs since we are more concerned with the differences between the value functions rather than with their exact amounts.

In this step, we reduce the capacity of each leg $i \in I$ by exactly one unit independently; that is, $x - e_i$, where e_i is the standard basis vector with all elements equal to zero, except the

i-th position with value 1.

For each leg i with new reduced capacity, we calculate the current approximate Q-value $Q(x - e_i, a)$ and obtain $\tilde{V}(x - e_i)$ using Equation 6.10. By subtracting this value from the reference value function, we obtain the approximate opportunity cost ξ_i of resource i through the following equation:

$$\xi_i \approx \tilde{V}(x) - \tilde{V}(x - e_i) \qquad \forall i \in I \tag{6.11}$$

Algorithm 5 describes the details of the capacity value assessment phase.

Algorithm 5 Opportunity Cost Assessment

1: **procedure** THE RESTRICTED SET OF RANDOMLY CHOSEN ACTIONS Ω IS GIVEN.
2: compute $Q(x, a)$ using RDQN. ▷ x is the initial state where all resources are available.
3: compute reference $\tilde{V}(x)$. ▷ according to its definition given in Equation 6.10.
4: **for** each $i \in I$ **do**:
5: decrease capacity of leg-i by 1 unit, $x - e_i$.
6: predict $Q(x - e_i, a)$ using RDQN.
7: compute $\tilde{V}(x - e_i)$.
8: compute $\xi_i \approx \tilde{V}(x) - \tilde{V}(x - e_i)$ ▷ approximate opportunity cost of resource i
9: Return ξ ▷ approximate opportunity cost vector

Phase 2: Action Generation

The pseudo code of the Action (offerset) generation is provided in Algorithm 6, an algorithm inspired by an interesting study that Bront et al. [32] have published.

The general idea behind this algorithm is to iteratively generate a new action on top of the current restricted set Ω. In order to generate each action, we first start with an empty set; then, the algorithm chooses the first product of the new set as the one with the highest expected return while taking the probability of its purchase into account. The process continues iteratively by greedily searching for the new products to be added to the current set.

This step ends when there is no new product found to be added to the current offerset such that the value of the offerset exceeds its current value. It is also of great importance to note that based on MNL properties, the availability of different alternative products in the set will affect the customer choice preferences of the ones already existing in the offerset.

As mentioned in 6.2.1, $P_{lj}(S)$ specifies the probability of purchasing product j by a segment-l customer when set S is offered. Therefore, the new choice probability of the products in

Algorithm 6 Action Generation

procedure Approximate opportunity cost vector ξ and initial set of restricted actions Ω are given.

define Ψ as set of potential new offersets . ▷ Initially, Ψ is empty.

define S as the generated offerset. ▷ Initially, S is empty.

define y as an assignment vector with binary variables, one for each product j.

$$y_j = \begin{cases} 1 & \text{if producst } j \text{ is offered in the generated set } S, \\ 0 & \text{otherwise.} \end{cases}$$

define S' as a set of products j with unassigned y_j values.

define $w_j = r_j - \sum_i \eta_{ij}\xi_i, \forall j \in J$ ▷ w gives the value of each product.

If $w_j \leqslant 0, \forall j \in J \rightarrow y_j = 0$, update S' ▷ these products are not in the offerset.

$j_1^* \leftarrow \underset{j \in S'}{\operatorname{argmax}}\{\sum_{l=1}^{L} \frac{w_j v_{lj}}{v_{lj}+v_{l0}}\}$ ▷ j_1^* is the first product of S.

update $y_{j_1^*} = 1$ and $S = \{j_1^*\}$ and $S' = S' - \{j_1^*\}$

until S is not changed **repeat**

compute

$$j^* = \underset{j \in S'}{\operatorname{argmax}}\{\sum_{l=1}^{L} \lambda_l \sum_{k \in \Gamma_l \cap (S \cup \{j\})} w_k P_{lk}(S \cup \{j\})\}$$

If $\Phi(S \cup \{j^*\}) > \Phi(S)$, then:

$$S := S \cup \{j^*\},\ S' = S' - \{j^*\}$$

$$\Psi := \Psi \cup \{S\}$$

end until

$\Psi = \Psi - \Omega$ ▷ Ψ consists of offersets not considered before.

If $\Psi = \emptyset$: Break; **Else:** $S \leftarrow \{\text{last element in } \Psi\}$. ▷ S is a new offerset with the highest value.

$\Omega = \Omega \cup \{S\}$

offerset S, after adding a new product j, will be computed using the equation below:

$$P_{lk}(S \cup \{j\}) = \frac{v_{lk}}{\sum\limits_{k \in \Gamma_l \cap (S \cup \{j\})} v_{lk} + v_{l0}} \tag{6.12}$$

Moreover, if we define expected net profit of set S, $\Phi(S)$, as below:

$$\Phi(S) = \sum_{l=1}^{L} \lambda_l \sum_{k \in \Gamma_l \cap (S)} w_k P_{lk}(S) \tag{6.13}$$

then, the expected net profit of set S after adding a new product j will be updated according to the following equation:

$$\Phi(S \cup \{j\}) = \sum_{l=1}^{L} \lambda_l \sum_{k \in \Gamma_l \cap (S \cup \{j\})} w_k P_{lk}(S \cup \{j\}) \tag{6.14}$$

As shown in Figure 6.3, the whole process ends when there is no new action with positive value left to be added to the restricted set of already existing actions.

6.4 Numerical Results

In this section, we evaluate the performance of RDQN (*i.e.,* DQN with an integrated action generation algorithm) in both classic "Parallel Flights" and "Hub and Spoke Network" examples, which are both based on the examples given by Bront et al. [32]. The upper bounds are the maximum revenues obtained by solving the exact choice-based deterministic linear programming model. Moreover, the results are compared to those of simulated CDLP assignments and bid price control policy.

CDLP is one of the most recent methodologies used to perform inventory control management in RM while taking customer's choice behavior into consideration. CDLP provides a set of products to offer at each time period such that the firm can maximize its revenue [32]. Moreover, bid price control policy is a widely used heuristic approach mainly adopted by the airline industry. Using the bid price control policy, a marginal value is computed for each resource, and the product is offered if its fare exceeds the sum of the bid prices of its constituent resources [95].

The computational results have been carried out on a 2.21 GHz 7-core computer with 16 GB of RAM. The codes are written in Python 3.6.4 and the environment FICO Xpress-Mosel 7.2.1 has been used to solve CDLP model.

6.4.1 Parallel Flights Example

As illustrated in Figure 6.4, the example in question consists of a network of three parallel legs between the same origin and destination. Each leg corresponds to a different time of a day (morning, afternoon, evening) with initial capacities of 30, 50, and 40, respectively. With two fare classes, High (H) and Low (L), for each leg, there are six products altogether. This results in $2^6 = 64$ actions (possible offersets). Table 6.2 demonstrates the product description [32].

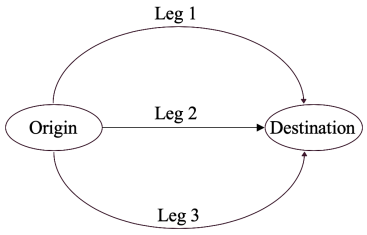

Figure 6.4 Parallel flights example

Table 6.2 Product description for the parallel flights example

Product	Leg	Class	Fare
1	1	L	400
2	1	H	800
3	2	L	500
4	2	H	1000
5	3	L	300
6	3	H	600

Customers are divided into four overlapping segments based on their time preferences and price sensitivities. We assume a maximum of one customer shows up at each time step. Given 300 time steps in the entire booking horizon and a customer arrival rate of 0.5, we will have an average of 150 customers per episode in the parallel flights example. Customer segmentations are shown in Table 6.3. We use Monte Carlo simulation to simulate customer's choice behavior based on the parameters of MNL Model.

Table 6.3 Customer segmentation for the parallel flights example

Segment	Consideration set	Pref. vector	λ_l	Description
1	{2,4,6}	(5,10,1)	0.1	Price insensitive, afternoon preference
2	{1,3,5}	(5,1,10)	0.15	Price sensitive, evening preference
3	{1,2,3,4,5,6}	(10,8,6,4,3,1)	0.2	Early preference, price sensitive
4	{1,2,3,4,5,6}	(8,10,4,6,1,3)	0.05	Price insensitive, early preference

Time starts at $t = 0$ and moves forward in discrete increments until T, the terminal state. At each time period, a customer shows up with probability λ which may belong to any of the segments. Afterwards, she either purchases a product $j \in S$ with probability $\lambda P_j(s)$ or leaves without any purchase. By selling one unit of product j, the current state x would be updated by subtracting one unit of capacity from product j's associated resource (leg) i.

All details of DQN implementation can be found in [88]. RDQN is implemented using Keras [84], which is an open source Python library. For RDQN, we use the same architecture as that of DQN, except for the integration of AGen. Moreover, in this example, we evaluated the impact of capacity modification on the total expected revenue through a scale parameter, α, to adjust all leg capacities. For example, $\alpha = 0.8$ means the initial capacity of $0.8 \times (30, 50, 40) = (24, 40, 32)$, and $\alpha = 1$ results in the original base case.

$\alpha = 0.6$

The process starts with creating a restricted action set Ω. For simplicity, we started with a set of six actions with only one product j at each offerset. Having run RDQN for 3000 episodes, we now have the approximate Q-values required to calculate opportunity costs for each of the three resources. The action generation, then, starts with defining an empty offerset S which will eventually be added to the six existing actions in Ω.

By applying the iterative action generation algorithm as explained in algorithm 6, we obtain the first generated offerset. Now we return to the master problem to solve RDQN using the new "effective" action set with 7 offersets. Then follows the subproblem consisting of the opportunity cost assessment and the iterative action generation. This loop between master problem and subproblem continues until no more "effective" offerset which can be added to the current restricted set to improve the expected revenue exists.

To make it easier to understand, we continue with a simple example of the procedure. Let us consider the case with $\alpha = 0.6$. The exact method results in a total revenue of 56,884 \$ which is the objective function value of the CDLP model (*i.e.,* UB). We observe that in the optimal policy achieved by CDLP, only four actions were taken throughout the whole booking horizon: {6}, {4, 6}, {2, 4, 6}, and {2, 4, 5, 6}. On the other hand, after solving

RDQN, we obtained the final restricted set of nine actions including {2, 4, 6}, {4, 5, 6}, and {1, 2, 3, 4, 5, 6} along with the original Ω consisting of six offersets; one product in each offerset with no repetitions.

To estimate how good our method performs, we solve the exact method using only the nine offersets generated by action generation. The achieved revenue of 56,528 $ is remarkably close to the UB value, showing that we have had a noticeably good performance using RDQN with only nine actions rather than DQN with 64 actions. The reduction in the size of the action set directly affects the computational cost; however, due to smallness of the example, the difference is not outstanding in this case.

Table 6.4 provides a detailed comparison between the results of RDQN and those of alternative approaches considering differences in both capacity and customers' willingness to pay. Once RDQN is trained, the trained model is used to interact with the customers for another 2000 episodes to achieve the reported simulated results. The size of the replay memory and that of the mini-batch are 3000 and 100, respectively. Furthermore, as for the processing time, our observations indicate that the maximum values corresponding to the training time and the simulation processing time were 30 and 10 minutes, respectively.

Table 6.4 Comparison of the obtained revenue for the parallel flights example with 95% CI

α	v_0	UB	CDLP	CDLP Bid Price	DQN	RDQN
0.6	(1, 5, 5, 1)	56,884	54,156 ± 234	55,144 ± 128	55,254 ± 241	55,012 ± 144
	(1, 10, 5, 1)	56,848	54,003 ± 225	55,191 ± 152	55,302 ± 513	55,131 ± 149
0.8	(1, 5, 5, 1)	71,936	67,815 ± 237	67,515 ± 518	69,355 ± 521	69,144 ± 197
	(1 ,10, 5, 1)	71,794	67,853± 226	66,918 ± 532	67,522 ± 721	67,259 ± 203
1	(1, 5, 5, 1)	79,155	75,700 ± 253	70,034 ± 718	77,323 ± 658	76,505 ± 276
	(1, 10, 5, 1)	76,866	73,753± 257	67,515 ± 708	73,907 ± 851	73,770 ± 273
1.2	(1, 5, 5, 1)	80,371	79,012 ± 320	71,797 ± 1062	79,450 ± 711	79,109 ± 298
	(1, 10, 5, 1)	78,045	76,978 ± 335	69,630 ± 1135	77,014 ± 902	76,885 ± 314

In the above table, v_0 is the no-purchase weight vector, with four elements, one for each consideration set per each segment of customers. For example, having two cases of $v_0 = (1, 10, 5, 1)$ and $v_0 = (1, 5, 5, 1)$, implies that the customers of the first case are less willing to pay compared to the second one. In addition, the results of CDLP-based methods and DQN are taken from [82] and [88], receptively.

The numerical results of the RDQN indicate that using RDQN instead of DQN results, not only in keeping the mean expected revenue at almost the same quality range but also in successfully improving the level of confidence intervals with more robust solutions.

This mainly occurs because of having much lower numbers of actions compared to the original

number of actions in DQN; thus, the agent will have fewer opportunities to explore. As it is valid for DQN, here again, we observe a constant good performance of RDQN in almost all scenarios with different capacity size and customer willingness to pay.

Studying the simulated results of CDLP model, we observe that it shows a better performance for ample capacity case (*i.e.,* $\alpha = 1.2$) compared to the highly constrained capacity cases (*i.e.,* $\alpha = 0.6$ and 0.8). In general, decision making in small size problems is more challenging due to the fact that there is a much less margin of mistake in resource allocation process of such cases. This is when the outperforming of DQN and RDQN compared to CDLP is more evident.

On the other hand, CDLP bid price achieves noticeably high mean expected revenue in restricted capacity cases, however, its performance drops dramatically when capacity is increased. Overall, although RDQN shows slightly lower performance quality compared to DQN, it still outperforms CDLP-based methods in majority of scenarios. The main advantage of RDQN comes to play when we have a larger scale problem such as hub and spoke example.

6.4.2 Hub and Spoke Example

The second example consists of a 7-leg hub and spoke network with 22 products. Figure 6.5 and Table 6.5 illustrate the design of the network and contain description of the products, respectively.

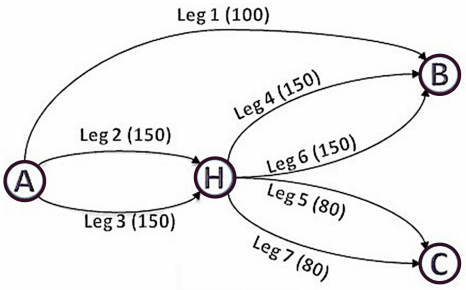

Figure 6.5 Hub and spoke example

Unlike in the simple parallel flights example, here, products may use more than one resource. This characteristic of flight network increases the complexity of the example in question. Moreover, the initial capacity of legs are $(100, 150, 150, 150, 150, 80, 80)$, respectively, and there are 1000 time periods in booking horizon, $T = 1000$. Thus, we are dealing with a larger scale problem compared to the first example.

Table 6.5 Hub and spoke example

Product	Leg	Class	Fare	Product	Leg	Class	Fare
1	1	H	1000	12	1	L	500
2	2	H	400	13	2	L	200
3	3	H	400	14	3	L	200
4	4	H	300	15	4	L	150
5	5	H	300	16	5	L	150
6	6	H	500	17	6	L	250
7	7	H	500	18	7	L	250
8	{2,4}	H	600	19	{2,4}	L	300
9	{3,5}	H	600	20	{3,5}	L	300
10	{2,6}	H	700	21	{2,6}	L	350
11	{3,7}	H	700	22	{3,7}	L	350

Having an arrival rate of $\lambda = 0.91$ results in an average arrival rate of 910 customers per stream of demand. Customers belong to 10 overlapping segments based on their time and price sensitivity as well as on their origin-destination choice. Customer segmentation definitions are presented in Table 6.6.

Table 6.6 Customer segmentation definitions for the hub and spoke network example

Segment	O-D	Consideration set	Pref. vector	λ_l	Description
1	A $\rightarrow$ B	{1,8,9,12,19,20}	(10,8,8,6,4,4)	0.08	Price insensitive, early pref.
2	A $\rightarrow$ B	{1,8,9,12,19,20}	(1,2,2,8,10,10)	0.2	Price sensitive
3	A $\rightarrow$ H	{2,3,13,14}	(10,10,5,5)	0.05	Price insensitive
4	A $\rightarrow$ H	{2,3,13,14}	(2,2,10,10)	0.2	Price sensitive
5	H $\rightarrow$ B	{4,5,15,16}	(10,10,5,5)	0.1	Price insensitive
6	H $\rightarrow$ B	{4,5,15,16}	(2,2,10,8)	0.15	Price sensitive, slight early pref.
7	H $\rightarrow$ C	{6,7,17,18}	(10,8,5,5)	0.02	Price insensitive, slight early pref.
8	H $\rightarrow$ C	{6,7,17,18}	(2,2,10,8)	0.05	Price sensitive
9	A $\rightarrow$ C	{10,11,21,22}	(10,8,5,5)	0.02	Price insensitive, slight early pref.
10	A $\rightarrow$ C	{10,11,21,22}	(2,2,10,10)	0.04	Price sensitive

We start dealing with this example by creating a restricted set of actions Ω with 22 actions, in which, every action consists of a single product $j \in J$. In the first step, we solve the initial master problem using RDQN considering only the restricted action set and performing the method for 3000 episodes.

Afterwards, we calculate the opportunity costs for each of the seven resources based on the algorithm 5. The next step involves applying the iterative action generation algorithm as defined in the algorithm 6. Eventually, the generated offerset will be added to the initial restricted set yielding total 23 offersets in total.

The hub and spoke example, then, will be solved using the new restricted action set with 23 updated offersets. After processing another 3000 episodes and estimating the updated opportunity costs, the next action will be generated and added to the existing set.

This repetitive process of solving the master problem, opportunity cost assessment and action generation will be continued until addition of new actions does not increase the expected potential revenue anymore. Here again, the trained model is used in simulation process for 2000 episodes to obtain the results which are going to be reported.

In the meantime, regarding the processing time of RDQN, our observations denote that training takes between 50 to 170 minutes depending on the number of the generated actions whereas simulation takes less than 20 minutes. The size of the replay memory and mini-batch are 3000 and 200, respectively.

Table 6.7 presents the results of solving the seat inventory control problem in the hub and spoke example using the exact method (UB), simulated CDLP-based methods, which are taken from [82], and RDQN.

Columns in this table are defined in like manner as in the previous example 6.4.1: UB, CDLP, CDLP bid price and RDQN represent the upper bound value, the result of simulating the control policies obtained through CDLP, bid price and RDQN approaches, respectively.

Here again, α is a scale parameter used to examine the effects of having various resource capacities on the performance of our method. v_0 is the no-purchase weight vector which has one coordinate for each segment, making it a vector of 10 elements. However, for the sake of simplicity, it was decided to reduce its length to two. Based on the definitions provided in Figure 6.6, the segments can be classified in five pairs based on their origin-destination. We assume the no-purchase preference weight to repeat for each pair (*e.g.*, $v_0 = (5, 10, 5, 10, 5, 10, 5, 10, 5, 10)$ is represented as $v_0 = (5, 10)$) [32].

Finally, the last column of the table, named "Added Actions", contains the number of actions (offersets) generated using AGen. The total number of actions for each case (*i.e.,* each α

and v_0 combination) consists of the initial 22 actions plus the number of the added actions generated through AGen iterations. For instance, for the case of $\alpha = 0.6$ and $v_0 = (5, 10)$, there are $22 + 24 = 46$ actions in total.

Table 6.7 Comparison of the obtained results for the hub and spoke example with 95% CI.

α	v_0	UB	CDLP	CDLP Bid Price	RDQN	Added Actions
0.6	(1, 5)	215,793	207,709 ± 370	190,384 ± 1,208	210,379 ± 234	19
	(5, 10)	200,515	193,543 ± 372	175,601 ± 906	194,305 ± 266	24
	(10, 20)	170,137	163,628 ± 329	159,685 ± 844	166,787 ± 255	18
0.8	(1, 5)	266,934	260,535 ± 428	224,069 ± 1,281	264,129 ± 283	17
	(5, 10)	223,173	218,231 ± 341	203,810 ± 983	220,144 ± 265	17
	(10, 20)	188,547	184,030 ± 337	175,720 ± 1,096	185,976 ± 298	12
1	(1, 5)	281,967	276,968 ± 434	242,195 ± 986	278,552 ± 290	9
	(5, 10)	235,284	229,704 ± 374	216,860 ± 1,410	231,711 ± 321	11
	(10, 20)	192,038	190,136 ± 404	186,438 ± 1,229	190,670 ± 325	10

According to Table 6.7, RDQN outperforms CDLP as well as the methods based on the bid-price policy in all cases. The differences between CDLP and RDQN are more significant in the highly constrained capacity cases (*i.e.,* $\alpha = 0.6$ and 0.8), yet both methods perform reasonably good on the original capacity case. However, RDQN still shows a mean expected revenue slightly closer to UB. Overall, when the capacity exceeds the demand for different legs, offering almost all products at each time period would be a reasonable strategy as both methods will show similar performances.

On the other hand, a limited capacity makes the choice-based problem more complicated because the margin of error will be more limited, and thus, the obtained revenue will greatly depend on the products offered at each time step. In this example, CDLP bid price shows less promising results overall. Moreover, as expected, when there is a high chance of no-purchase (*i.e.,* $v_0 = (10, 20)$), the revenue decreases in all cases.

6.5 Conclusion

In this paper, we intended to solve a large-size choice-based seat inventory control problem using deep reinforcement learning techniques. Throughout this study, it was assumed that customers belonged to overlapping segments and their choice behavior was simulated using parameters of the Multinomial Logit model. We also realized that by using DQN, as a common well-known DRL method, the problem of extreme computational cost would emerge [88]. In fact, the main issue surfaces as an exponential growth in the number of actions when the number of products increases. To handle this issue, we proposed an action generation

algorithm called AGen.

An AGen is a greedy heuristic algorithm which generates a set of "effective" actions to be used instead of $2^{|J|}$ possible offersets. Using an iterative process, we integrated AGen into DQN and obtained a method named Restricted DQN (RDQN) which consisted of two main steps: a master problem and a subproblem. The master problem could be regarded as an evaluation unit that uses DQN to explore the performance of the actions generated by the subproblem.

RDQN performed remarkably well in practice considering both computational complexity and quality of the results. The algorithm was examined on two examples: the parallel flights and the hub and spoke network. In all scenarios, RDQN produced better results than CDLP and bid price control policy did, specially in the scarce capacity cases.

CHAPTER 7 GENERAL DISCUSSION

In this thesis, we have studied application of novel data-driven approaches to customer choice-based network Revenue Management Systems (RMS). Our main focus was on two main cores of any RMS; namely, forecasting module and seat inventory control module. Specifically, we have studied how state-of-the-art machine learning methodologies can help better manage an RMS, and hence, improve its performance. Our numerical results indicate desirable performance of our algorithms in terms of solution quality and processing time.

In chapter 2, we have developed a combination of various preprocessing, machine learning and feature engineering techniques in order to predict the number of potential arrival customers on different aggregation levels. In doing so, in addition to adopting advanced machine learning approaches, we developed new heuristic methods to extract underlying shallow and deep hidden characteristics of historic observations. Specifically, we demonstrated that to what extent a precise clustering can improve the quality of the forecasting results.

In chapter 3, we addressed a challenging revenue management problem in the absence of forecasting information. To tackle this problem, we explored a deep reinforcement learning methodology which can successfully learn customers' choice behavior and propose inventory management decisions at every period by taking the current availability of the resources into consideration. We gained promising results by applying this algorithm to common RM examples provided in the literature.

In chapter 4, we proposed a solution approach to tackle large practical RM problems. The main difficulty in larger problems arises when the number of products ($|J|$) rises and results in an exponential increase in possible combinations of the products $2^{|J|}$ (called offersets or actions). To address the problem of large discrete action space while inspired by column generation algorithm, we developed a heuristic algorithm called "Action Generation" (AGen). AGen contributes to solving the RM problem by generating most profitable "effective" actions while taking the opportunity costs of resources into consideration. The promising numerical results indicate desirable performance of our proposed methodology.

As a general discussion, during the past few years, we have observed that both firms and customers use more advanced analytical techniques in their decision-making processes. In the presence of increasing competitors and lower-cost carriers, it is essential for the future of the RM systems to develop methodologies which can help them react dynamically in real time. Moreover, the presence of large amounts of data necessitates development of algorithms that can handle the processing of such data in a reasonable time frame.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

In summary, we made the following advancements in this thesis.

First, we started with exploring state-of-the-art machine learning algorithms and developing advanced heuristic feature engineerings techniques in order to estimate the expected number of future bookings for a railway demand forecasting problem in different aggregation levels. In addition, we demonstrated the critical role of precise clustering in substantial performance improvement of forecasting results.

In the next step, we tackled a seat inventory control problem with the assumption of having partial demand information. This situation can happen in new markets or in the presence of new competitors in the existing market. To address this problem, we developed a learner (*i.e.,* an agent) which can learn the market and customer choice behaviors by both observing the past historical bookings and learning from the most recent interactions between the agent and the environment. An agent trained with these data can precisely estimate customer's choice behavior and thus, make informed decisions throughout the booking horizon in order to increase the total revenue.

Solving a choice-based seat inventory control system for the real world practical problems, we face the challenge of having exponentially large number of possible actions when number of products increases. We addressed this drawback by developing a heuristic action generation algorithm to iteratively generate effective actions while taking the computed opportunity costs of products' constituent resources into consideration.

There are several avenues for further development of this research, including:

- extending the proposed forecasting methodology to estimate unconstrained demand
- computing the product replacement probabilities through analyzing historical observations
- developing a Transfer learning (TL) method as an extension to the proposed DQN-based approach for the seat inventory control system
  TL helps transfer and apply the knowledge gained from solving one problem to a different but similar one. Since many RM problems share comparable concepts and structures with main differences in the number of products or legs, this can be regarded as a very valuable extension.

- improving the performance of AGen algorithm by considering the possibility of generating more than one action at every iteration of the algorithm and removing the redundant actions

REFERENCES

[1] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006, vol. 5.

[2] R. G. Cross, *Revenue management: Hard-core tactics for market domination*. Crown Business, 2011.

[3] K. T. Talluri and G. J. Van Ryzin, *The theory and practice of revenue management*. Springer Science & Business Media, 2006, vol. 68.

[4] R. H. Zeni, *Improved forecast accuracy in airline revenue management by unconstraining demand estimates from censored data*. Universal-Publishers, 2001.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] S. E. Kimes, "Yield management: A tool for capacity-considered service firms," *Journal of operations management*, vol. 8, no. 4, pp. 348–363, 1989.

[8] V. Helve *et al.*, "Demand forecasting in a railway revenue management system," 2015.

[9] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting methods and applications*. John wiley & sons, 2008.

[10] L. R. Weatherford and S. E. Kimes, "A comparison of forecasting methods for hotel revenue management," *International journal of forecasting*, vol. 19, no. 3, pp. 401–415, 2003.

[11] G. James *et al.*, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[12] A. Sanz-García *et al.*, "Application of genetic algorithms to optimize a truncated mean k-nearest neighbours regressor for hotel reservation forecasting," in *International Conference on Hybrid Artificial Intelligence Systems*. Springer, 2012, pp. 79–90.

[13] N. António, A. de Almeida, and L. M. Nunes, "Using data science to predict hotel booking cancellations," in *Handbook of research on holistic optimization techniques in the hospitality, tourism, and travel industry*. IGI Global, 2017, pp. 141–167.

[14] S. S. Azadeh, P. Marcotte, and G. Savard, "A taxonomy of demand uncensoring methods in revenue management," *Journal of Revenue and Pricing Management*, vol. 13, no. 6, pp. 440–456, 2014.

[15] G. Van Ryzin and J. McGill, "Revenue management without forecasting or optimization: An adaptive algorithm for determining airline seat protection levels," *Management Science*, vol. 46, no. 6, pp. 760–775, 2000.

[16] K. Littlewood, "Forecasting and control of passenger bookings," *Airline Group International Federation of Operational Research Societies Proceedings, 1972*, vol. 12, pp. 95–117, 1972.

[17] P. Belobaba, "Air travel demand and airline seat inventory management," Ph.D. dissertation, Massachusetts Institute of Technology, 1987.

[18] P. P. Belobaba, "Optimal vs. heuristic methods for nested seat allocation," in *Presentation at ORSA/TIMS Joint National Meeting*, 1992.

[19] B. C. Smith, J. F. Leimkuhler, and R. M. Darrow, "Yield management at american airlines," *interfaces*, vol. 22, no. 1, pp. 8–31, 1992.

[20] R. Simpson, "Using network flow techniques to find shadow prices for market and seat inventory control," *Memorandum M89-1, Flight Transportation Laboratory, Massachusetts Institute of Technology, Cambridge, MA*, 1989.

[21] A. K. Strauss, R. Klein, and C. Steinhardt, "A review of choice-based revenue management: Theory and methods," *European Journal of Operational Research*, vol. 271, no. 2, pp. 375–387, 2018.

[22] G. van Ryzin and G. Vulcano, "An expectation-maximization method to estimate a rank-based choice model of demand," *Operations Research*, vol. 65, no. 2, pp. 396–407, 2017.

[23] I. Sher *et al.*, "Partial identification of heterogeneity in preference orderings over discrete choices," National Bureau of Economic Research, Tech. Rep., 2011.

[24] V. F. Farias, S. Jagabathula, and D. Shah, "A nonparametric approach to modeling choice with limited data," *Management science*, vol. 59, no. 2, pp. 305–322, 2013.

[25] M. Hosseinalifam, P. Marcotte, and G. Savard, "Network capacity control under a nonparametric demand choice model," *Operations Research Letters*, vol. 43, no. 5, pp. 461–466, 2015.

[26] M. Hosseinalifam, G. Savard, and P. Marcotte, "Computing booking limits under a non-parametric demand model: A mathematical programming approach," *Journal of Revenue and Pricing Management*, vol. 15, no. 2, pp. 170–184, 2016.

[27] K. Talluri and G. Van Ryzin, "Revenue management under a general discrete choice model of consumer behavior," *Management Science*, vol. 50, no. 1, pp. 15–33, 2004.

[28] W. Hanson and K. Martin, "Optimizing multinomial logit profit functions," *Management Science*, vol. 42, no. 7, pp. 992–1003, 1996.

[29] D. Zhang and D. Adelman, "An approximate dynamic programming approach to network revenue management with customer choice," *Transportation Science*, vol. 43, no. 3, pp. 381–394, 2009.

[30] G. Gallego *et al.*, "Managing flexible products on a network," 2004.

[31] Q. Liu and G. Van Ryzin, "On the choice-based linear programming model for network revenue management," *Manufacturing & Service Operations Management*, vol. 10, no. 2, pp. 288–310, 2008.

[32] J. J. M. Bront, I. Méndez-Díaz, and G. Vulcano, "A column generation algorithm for choice-based network revenue management," *Operations Research*, vol. 57, no. 3, pp. 769–784, 2009.

[33] S. A. M. Shihab *et al.*, "Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking," 2019.

[34] C. W. Chase Jr, "Revenue management: a review," *The Journal of Business Forecasting*, vol. 18, no. 1, p. 2, 1999.

[35] J. I. McGill and G. J. Van Ryzin, "Revenue management: Research overview and prospects," *Transportation science*, vol. 33, no. 2, pp. 233–256, 1999.

[36] A. Sen, "Examining air travel demand using time series data," *Journal of Transportation Engineering*, vol. 111, no. 2, pp. 155–161, 1985.

[37] R. Devoto, C. Farci, and F. Lilliu, "Analysis and forecast of air transport demand in sardinia's airports as a function of tourism variables," *WIT Transactions on The Built Environment*, vol. 60, 2002.

[38] C. Lim and M. McAleer, "Time series forecasts of international travel demand for australia," *Tourism Management*, vol. 23, no. 4, pp. 389–396, 2002.

[39] J. W. Taylor, "Short-term electricity demand forecasting using double seasonal exponential smoothing," *Journal of the Operational Research Society*, vol. 54, no. 8, pp. 799–805, 2003.

[40] C. García-Ascanio and C. Maté, "Electric power demand forecasting using interval time series: A comparison between var and imlp," *Energy Policy*, vol. 38, no. 2, pp. 715–725, 2010.

[41] G. E. Box *et al.*, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[42] S. Sharif Azadeh, "Demand forecasting in revenue management systems," Ph.D. dissertation, École Polytechnique de Montréal, 2013.

[43] A. Zakhary, N. El Gayar, and A. F. Atiya, "A comparative study of the pickup method and its variations using a simulated hotel reservation data," *ICGST international journal on artificial intelligence and machine learning*, vol. 8, pp. 15–21, 2008.

[44] S. Mishra and V. Viswanathan, "Revenue management with restriction-free pricing," in *Proceedings of the AGIFORS Revenue Management and Distribution Study Group Meeting*, 2003.

[45] T. O. Gorin, "Airline revenue management: Sell-up and forecasting algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[46] S. Ja, B. Rao, and S. Chandler, "Passenger recapture estimation in airline rm," in *AGIFORS 41st Annual Symposium*, 2001.

[47] C. Cleophas, M. Frank, and N. Kliewer, "Recent developments in demand forecasting for airline revenue management," *International Journal of Revenue Management*, vol. 3, no. 3, pp. 252–269, 2009.

[48] H. Ziekow *et al.*, "The potential of smart home sensors in forecasting household electricity demand," in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2013, pp. 229–234.

[49] M. K. Tiwari and J. F. Adamowski, "An ensemble wavelet bootstrap machine learning approach to water demand forecasting: A case study in the city of calgary, canada," *Urban Water Journal*, vol. 14, no. 2, pp. 185–201, 2017.

[50] B. Yildiz, J. I. Bilbao, and A. B. Sproul, "A review and analysis of regression and machine learning models on commercial building electricity load forecasting," *Renewable and Sustainable Energy Reviews*, vol. 73, pp. 1104–1122, 2017.

[51] B. Lantz, *Machine learning with R.* Packt Publishing Ltd, 2013.

[52] R. E. Shiffler, "Maximum z scores and outliers," *The American Statistician*, vol. 42, no. 1, pp. 79–80, 1988.

[53] S. Seo, "A review and comparison of methods for detecting outliers in univariate data sets," Ph.D. dissertation, University of Pittsburgh, 2006.

[54] R. J. Schalkoff, *Artificial neural networks.* McGraw-Hill Higher Education, 1997.

[55] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[56] L. Breman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[57] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Nonlinear estimation and classification.* Springer, 2003, pp. 149–171.

[58] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008.

[59] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[60] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[61] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[62] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[63] M. P. Sesmero, A. I. Ledezma, and A. Sanchis, "Generating ensembles of heterogeneous classifiers using stacked generalization," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 21–34, 2015.

[64] F. Martínez *et al.*, "Time series forecasting with knn in r: the tsfknn package," *The R Journal*, 2019.

[65] R. Ratliff and D. Guenther, "Availability-based pricing for multi-channel distribution," Sep. 7 2006, uS Patent App. 11/072,059.

[66] T. Barbier *et al.*, "Product-closing approximation for nonparametric choice network revenue management," *arXiv preprint arXiv:1805.10537*, 2018.

[67] Z.-J. M. Shen and X. Su, "Customer behavior modeling in revenue management and auctions: A review and new research opportunities," *Production and operations management*, vol. 16, no. 6, pp. 713–728, 2007.

[68] L. R. Weatherford and R. M. Ratliff, "Review of revenue management methods with dependent demands," *Journal of Revenue and Pricing Management*, vol. 9, no. 4, pp. 326–340, 2010.

[69] S. S. Azadeh, M. Hosseinalifam, and G. Savard, "The impact of customer behavior models on revenue management systems," *Computational Management Science*, vol. 12, no. 1, pp. 99–109, 2015.

[70] M. Schwind, *Dynamic pricing and automated resource allocation for complex information services: reinforcement learning and combinatorial auctions*. Springer Science & Business Media, 2007, vol. 589.

[71] C. Raju, Y. Narahari, and K. Ravikumar, "Learning dynamic prices in electronic retail markets with customer segmentation," *Annals of Operations Research*, vol. 143, no. 1, pp. 59–75, 2006.

[72] V. L. R. Chinthalapati, N. Yadati, and R. Karumanchi, "Learning dynamic prices in multiseller electronic retail markets with price sensitive customers, stochastic demands, and inventory replenishments," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 1, pp. 92–106, 2006.

[73] A. GOSAVII, N. Bandla, and T. K. Das, "A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking," *IIE transactions*, vol. 34, no. 9, pp. 729–742, 2002.

[74] Z. Sui, A. Gosavi, and L. Lin, "A reinforcement learning approach for inventory replenishment in vendor-managed inventory systems with consignment inventory," *Engineering Management Journal*, vol. 22, no. 4, pp. 44–53, 2010.

[75] R. Rana and F. S. Oliveira, "Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning," *Omega*, vol. 47, pp. 116–126, 2014.

[76] M. L. Puterman, *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[77] D. Silver, "Reinforcement learning and simulation-based search," 2009.

[78] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[79] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA:, 2015, vol. 25.

[80] B. J. Wythoff, "Backpropagation neural networks: a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 2, pp. 115–155, 1993.

[81] Z. Yang, Y. Xie, and Z. Wang, "A theoretical analysis of deep q-learning," *arXiv preprint arXiv:1901.00137*, 2019.

[82] J. M. Chaneton and G. Vulcano, "Computing bid prices for revenue management under customer choice behavior," *Manufacturing & Service Operations Management*, vol. 13, no. 4, pp. 452–470, 2011.

[83] D. Zhang and W. L. Cooper, "Revenue management for parallel flights with customer-choice behavior," *Operations Research*, vol. 53, no. 3, pp. 415–431, 2005.

[84] F. Chollet *et al.*, "Keras," 2015.

[85] P. Belobaba and C. Hopperstad, "Boeing." MIT simulation study: PODS results update. AGIFORS Reservations, 1999.

[86] M. Frank *et al.*, "Airline revenue management: A simulation of dynamic capacity management," *Journal of Revenue and Pricing Management*, vol. 5, no. 1, pp. 62–71, 2006.

[87] M. Frank, M. Friedemann, and A. Schröder, "Principles for simulations in revenue management," *Journal of Revenue and Pricing Management*, vol. 7, no. 1, pp. 7–16, 2008.

[88] N. Etebarialamdari and G. Savard, "Deep reinforcement learning approach for customer choice-based seat inventory control problem," vol. 216, no. 2, pp. 459–468, 2019.

[89] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[90] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[91] V. K. Borooah, *Logit and probit: Ordered and multinomial models*. Sage, 2002, no. 138.

[92] J. Meissner and A. Strauss, "Network revenue management with inventory-sensitive bid prices and customer choice," *European Journal of Operational Research*, vol. 216, no. 2, pp. 459–468, 2012.

[93] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research*, vol. 7, no. May, pp. 877–917, 2006.

[94] Y. Achbany *et al.*, "Managing the exploration/exploitation trade-off in reinforcement learning," *Technical Paper, Information Systems Research Unit (ISYS), IAG, Université Catholique de Louvain*, 2005.

[95] M. Hosseinalifam, P. Marcotte, and G. Savard, "A new bid price approach to dynamic resource allocation in network revenue management," *European Journal of Operational Research*, vol. 255, no. 1, pp. 142–150, 2016.