POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Machine Learning for Booking Control

JUSTIN DUMOUCHELLE
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Mathématiques appliquées

Août 2021

Machine Learning for Booking Control

DEDICATION

To my parents.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Andrea Lodi and Emma Frejinger, for their direction and support throughout the duration of my degree and this thesis. Without them, I would have not had the opportunity to work on an exciting application at the intersection of machine learning and operations research. I greatly appreciate the flexibility with remote meetings and quick replies to emails which has made much of this work possible during the pandemic.

Within this context of my thesis project, I would like to thank my colleagues Gabriele and Feng for providing an excellent set of resources which helped me when starting this project, Luca Accorsi and Daniele Vigo who provided access to their vehicle routing code FILO, and Francesca Guerriero for sharing details on the implementation of their approach.

From the chair, I would like to thank Khalid for the technical support and Mehdi for scheduling my interviews and helping me get set up at the chair. I am also grateful for the opportunity to work on Ecole with Antoine, Didier and Maxime which has been a great learning opportunity and collaboration. In addition, I would like to thank all of my colleagues at the chair for the countless great memories and helpful discussions.

I am grateful for my parents who have been supportive and encouraging throughout my entire education, without whom this would not be possible. In addition, I am thankful for my partner, Josie, who has always been there for the past 5 years and was supportive of my decision to move to Montréal.

RÉSUMÉ

La gestion des revenus est une approche analytique que les entreprises utilisent pour maximiser leurs profits. Cette thèse se concentre sur le problème du contrôle de l'acceptation et de rejet de réservations : Étant donné une capacité limitée, accepter une demande de réservation ou la rejeter afin de réserver de la capacité pour de futures réservations avec des revenus potentiellement plus élevés. Ce problème peut être formulé comme un programme dynamique stochastique à horizon fini, où l'acceptation d'une demande entraîne un profit et, à la fin de la période de réservation, le coût d'exécution des réservations acceptées est encouru. Le coût de l'exécution des demandes est appelé le problème de la prise de décision opérationnelle et, dans de nombreuses applications, il nécessite la résolution de problèmes non triviaux de programmation en nombres entiers mixtes.

Ce travail propose d'entraîner un modèle prédictif pour approximer la valeur de la solution du problème de prise de décision opérationelle par l'apprentissage supervisé. Les prédictions peuvent ensuite être exploitées par de la programmation dynamique approximative générale et de l'apprentissage par renforcement pour résoudre le problème du contrôle des réservations. Cette méthodologie est ensuite évaluée sur deux problèmes de contrôle de réservation issus de la littérature. Le premier problème est un problème de logistique de distribution, pour lequel cette méthodologie produit des politiques qui permettent d'obtenir des bénéfices significativement plus élevés avec un temps de calcul réduit par rapport aux solutions de base. La deuxième application est une application de fret aérien, où la performance de cette approche se situe juste en dessous de l'état de l'art. Bien que les résultats obtenus dans cette thèse ne soient pas complètement compétitifs par rapport à l'état de l'art, plusieurs directions pour les travaux futurs existent qui pourraient, nous l'espérons, conduire à une amélioration supplémentaire. En outre, la méthodologie présentée dans cette thèse est générale, c'est-à-dire qu'elle peut facilement être étendue à de nombreux problèmes à horizon fini et, pour cette raison, elle apporte une contribution notable au contrôle des réservations.

ABSTRACT

Revenue management is an analytic approach which companies utilize to maximize profit. This thesis focuses on the problem of controlling booking accept/reject decisions: Given a limited capacity, accept a booking request or reject it to reserve capacity for future bookings with potentially higher revenue. This problem can be formulated as a finite-horizon stochastic dynamic program, where accepting a request results in a profit and at the end of the booking period the cost of fulfilling the accepted bookings is incurred. The cost of fulfilling requests is referred to as the operational decision-making problem, and in many applications requires the solution of non-trivial mixed-integer programming problems.

This work proposes to train a predictor to approximate the solution value of the operational decision-making problem through supervised learning. The predictions can then be leveraged within general-purpose approximate dynamic programming and reinforcement learning to solve the booking control problem. This methodology is then evaluated on two booking control problems from the literature. The first problem is a distributional logistics problem, where this methodology produces policies that result in significantly higher profit at a reduced computing time when compared to baselines. The second application is an airline cargo application, where this approach falls just short of state-of-the-art baselines. Although the results achieved in this thesis are not competitive with the state-of-the-art, there are several directions for future work that can hopefully lead to further improvement. Furthermore, the methodology presented in this thesis is general, i.e., can easily be extended to many end-of-horizon problems and for this reason, it provides a valuable contribution to booking control.

TABLE OF CONTENTS

DEDICATION . . . iii

ACKNOWLEDGEMENTS . . . iv

RÉSUMÉ . . . v

ABSTRACT . . . vi

TABLE OF CONTENTS . . . vii

LIST OF TABLES . . . x

LIST OF FIGURES . . . xi

LIST OF SYMBOLS AND ACRONYMS . . . xii

LIST OF APPENDICES . . . xiii

CHAPTER 1 INTRODUCTION . . . 1
- 1.1 Motivation . . . 1
- 1.2 Problem Description . . . 1
- 1.3 Research Objectives . . . 2
- 1.4 Contributions . . . 3
- 1.5 Outline . . . 3

CHAPTER 2 LITERATURE REVIEW . . . 4
- 2.1 Dynamic Programming and Reinforcement Learning . . . 4
  - 2.1.1 Markov Decision Processes . . . 4
  - 2.1.2 Dynamic Programming . . . 6
  - 2.1.3 Reinforcement Learning . . . 10
  - 2.1.4 T-stage Optimal Control . . . 15
- 2.2 Mathematical Optimization . . . 15
  - 2.2.1 Linear Programming . . . 16
  - 2.2.2 Mixed Integer Linear Programming . . . 17
- 2.3 Supervised Learning . . . 18
  - 2.3.1 Formulation . . . 19

2.3.2 Loss Functions 20
2.3.3 Algorithms 20
2.4 Booking Control 23
2.4.1 Problem Formulation 23
2.4.2 Algorithms 24
2.5 Related Work 26

CHAPTER 3 METHODOLOGY & PROBLEM FORMULATIONS 29
3.1 Methodology 29
3.1.1 Approximate MDP Formulation 29
3.1.2 Supervised Learning 30
3.2 A Distributional Logistics Problem 30
3.2.1 Problem Description 31
3.2.2 Prediction Task 33
3.3 An Airline Cargo Problem 33
3.3.1 Problem Formulation 34
3.3.2 Prediction Task 35

CHAPTER 4 EXPERIMENTS 36
4.1 Application I - Distributional Logistics 36
4.1.1 Sets of Instances 36
4.1.2 Supervised Learning Results 36
4.1.3 Baselines 37
4.1.4 Control Algorithms 39
4.1.5 Reinforcement Learning Training 39
4.1.6 Control Results 39
4.2 Application II - Airline Cargo 43
4.2.1 Instance 43
4.2.2 Supervised Learning Results 44
4.2.3 Baselines 44
4.2.4 Control Algorithms 45
4.2.5 Reinforcement Learning Training 45
4.2.6 Control Results 45

CHAPTER 5 CONCLUSION AND RECOMMENDATIONS 49

REFERENCES 50

APPENDICES 55

LIST OF TABLES

Table 4.1 Distributional Logistics: Supervised Learning Results. 38
Table 4.2 Distributional Logistics: Supervised learning times (seconds). . 38
Table 4.3 Distributional Logistics: SARSA neural network hyperparameters. 40
Table 4.4 Distributional Logistics: Profits and computing times (seconds). 42
Table 4.5 Airline Cargo: Supervised Learning Results. 44
Table 4.6 Airline Cargo: Profits and computing times (seconds). 47
Table 4.7 Airline Cargo: Number of best solutions (out of 2000 instances). 48
Table A.1 Airline Cargo: Expected weights and volumes. 57
Table A.2 Airline Cargo: Profit per chargeable weight probabiltiies. . . . 58
Table A.3 Airline Cargo: Class request probabilities. 59

LIST OF FIGURES

Figure 4.1 Distributional Logistics: SARSA training curves for 1 run. . . 40
Figure 4.2 Distributional Logistics: Optimality gaps to best known solutions. 43
Figure 4.3 Airline Cargo: SARSA training curves. 46
Figure 4.4 Airline Cargo: Optimality gaps to best known solutions. . . . 47
Figure 4.5 Airline Cargo: Differences between DLP and SARSA-5. 48

LIST OF SYMBOLS AND ACRONYMS

MDP	Markov decision process
DP	dynamic programming
ADP	approximate dynamic programming
TD	temporal difference
MC	Monte Carlo
SARSA	State-Action-Reward-State-Action
MCTS	Monte Carlo tree search
RM	revenue management
VRP	vehicle routing problem
LP	linear program
MIP	mixed-integer program
MILP	mixed-integer linear program
ML	machine learning
RL	reinforcement learning
SVM	support-vector machine
MSE	mean squared error
MAE	mean absolute error
UCT	upper confidence bounds applied to trees
DLP	deterministic linear programming
RLP	randomized linear programming
SGD	stochastic gradient descent
ReLU	rectified linear unit
TSP	traveling salesman problem

LIST OF APPENDICES

Appendix A Sets of Instances 55

CHAPTER 1 INTRODUCTION

1.1 Motivation

Every retailer is faced with decisions regarding what products to offer, how to set prices, and how to manage inventory. These decisions can vastly impact the profit and ultimately dictate the success of the retailer. For this reason, the importance of making strategic decisions is crucial. In recent years, the demand for e-commerce and global parcel delivery has increased remarkably. For example, in 2019 there were 103 billion parcels shipped, which generated a revenue of \$351 billion [1], making logistics and parcel transport a key area of research for decision-making.

Revenue management (RM) has played an important role by providing a framework for making these decisions analytically. The first application of RM was developed for use in the airline passenger transport [2], but since then has expanded to a wide variety of sectors such as airline cargo management, car rentals, and time-constrained home delivery [3]. Typically, RM problems fall within three categories: (1) Structural decisions, which focuses on what services to offer and how to bundle services together. (2) Price decisions, which relates to the pricing of products or services over time. (3) Quantity decisions, which focuses on decisions made around accepting or rejecting an offer and how to allocate resources under limited capacity. The focus of this thesis is with respect to the last category, quantity decisions. Such decisions arise in a variety of applications such as airline cargo and distributional logistics.

1.2 Problem Description

In capacity control problems, a set of independent requests is received over a fixed time horizon, often referred to as the booking period. At the time of each request, a decision to either accept or reject the request needs to be made. Accepting a request results in a profit, however it decreases the capacity for future requests. In some applications, there is also a cost associated with the fulfillment of accepted requests. Rejecting the request will simply reserve the capacity for future requests.

The goal of capacity control is to decide if the profit from accepting a request outweighs the loss in capacity that can be used to fulfill future requests. This problem can be viewed as a stochastic sequential decision-making problem. The problem is stochastic as the future requests are unknown and typically sampled from a distribution. The sequential aspect of the problem arises as the decision of accepting or rejecting a request will ultimately affect the remaining decisions due to the loss in capacity.

Capacity control problems can be formulated as a Markov decision process (MDP). However, in many cases, general-purpose dynamic programming (DP) and reinforcement learning (RL) algorithms cannot be applied directly due to the unique nature of the problem. Specifically, the key component that distinguishes capacity control problems from other control problems is the need to determine how to fulfill the set of accepted requests at the end of the booking period. This fulfillment of requests is referred to as the *operational decision-making problem* and typically requires solving a discrete optimization problem in order to obtain a cost-minimizing solution. For this reason, any algorithm that seeks to determine a profiting-maximizing control policy must take the operational decision-making problem into account as it dictates the value at the final state in the MDP.

The difficulty of the operational problem can vary depending on the application. In applications such as airline seat allocation, the operational problem is trivial, since each seat can be assigned to a passenger and overbooking costs relate only to the number of excess passengers, which allows general DP and RL algorithms to be applied directly since simulation can be done efficiently. In other applications, solving the operational problem is not trivial and in some cases even intractable. Examples of more challenging operational problems can be observed through the vector packing problem arising in airline cargo management [4] or the vehicle routing problem (VRP) arising in distributional logistics [5].

In RM, control is often obtained through booking limit and bid-price policies [6]. Booking limit policies define thresholds for each request type and simply do not accept if the threshold is met. Bid-price policies define thresholds based on the expected revenue from the current request, the period, and the remaining capacity. Mathematical programming formulations are often used as the method for obtaining booking limit and bid-price policies.

1.3 Research Objectives

The objective of this work is to provide a general approximation to the capacity control problems that is agnostic to the operational problem, allowing for the use of general-purpose DP and RL algorithms. This approximation is obtained by replacing the formulation of the operational decision-making problem with a prediction function defined by offline supervised learning. This approach is then evaluated on two applications from the literature along with their respective baselines.

1.4 Contributions

In this thesis, the following contributions are made: (1) A methodology is proposed that *(i)* formulates an approximate MDP by replacing the formulation of the operational decision-making problem with a prediction function defined by offline supervised learning and *(ii)* uses approximate DP and RL techniques to obtain approximate control policies. This methodology can be generalized to a wide variety of booking control problems with different operational decision-making components. (2) An evaluation of the proposed methodology is done within the context of a distribution logistics problem and an airline cargo management problem from the literature. In the distributional logistics problem, the policies obtained by the proposed methodology show increased profit at a reduced evaluation time. In the airline cargo management problem, this approach provides policies which lead to slightly lower profit.

1.5 Outline

The remainder of the thesis is organized as follows. Chapter 2 introduces the essential mathematical concepts and reviews relevant literature. In particular DP, RL, mathematical optimization, machine learning, and capacity control RM are formally introduced. Chapter 3 presents the methodology with respect to a general capacity control problem and introduces two capacity control problems from the literature along with the details required to formulate the problems as supervised learning. Chapter 4 presents results from computational experiments with respect to both applications. Finally, Chapter 5 concludes.

CHAPTER 2 LITERATURE REVIEW

This chapter provides an introduction to the necessary mathematical concepts and a discussion of related work. Section 2.1 discusses MDPs, DP, and RL, Section 2.2 introduces linear and integer programming, Section 2.3 presents supervised learning, and Section 2.4 formalizes the capacity control problem as an MDP and discusses related work.

2.1 Dynamic Programming and Reinforcement Learning

Sequential decision-making problems arise in a wide variety of applications, such as robotics, combinatorial games, and autonomous driving. In this setting, there exists a decision maker or agent which can interact with an environment by taking actions. For each action, the agent receives a reward or cost, with the goal of either maximizing total reward or minimizing total cost. Taking an action provides the agent with a new state where the process is repeated. In a finite horizon problem, this is repeated until a terminal state is reached, while in an infinite horizon problem the process is indefinite.

Disciplines such as mathematical optimization and RL seek to formally model these decision-making problems and derive systematic approaches for determining good policies. The remainder of this section introduces how to model sequential decision-making tasks as a MDP, exact and approximate DP algorithms, RL algorithms, and N-stage-optimal control problems.

2.1.1 Markov Decision Processes

To mathematically model the sequential decision-making task described above, MDPs are typically used as a framework. In the decision-making process, at each time step t, the agent observes a state s_t, takes an action a_t, and observes a resulting reward r_{t+1} and the next state s_{t+1}. A trajectory from the initial state to terminal state T is then given by

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T. \tag{2.1}$$

In the RL literature, (2.1) is also often referred to as an episode. The components required to describe a MDP are hence the state space, the action space, and the probability transition function. The state space is simply the set of all possible states and is denoted by $\mathcal{S}$. The action space is the set of all actions and is denoted by $\mathcal{A}$. In some settings, the set of actions depends on the state, thus the action space for a state is given by $\mathcal{A}(s)$, for all $s \in \mathcal{S}$. The probability transition function describes the dynamics of the problem by giving the transition probability to the next state and reward given a current state and action. More precisely, the probability transition function is denoted by $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $p(s_{t+1}|s_t, a_t)$ is the probability of observing the reward r_{t+1} and the next state s_{t+1}, given that action a_t was taken at state s_t.

In general, the number of transitions before reaching a terminal state can be infinite. However, within the context of this work, it is assumed that the number of transitions to be finite and the notion of discounting [7] is not discussed.

Policies

A policy is a function that dictates the actions in the environment. In the most general case, a probabilistic function can be used which is denoted by $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, where $\pi(a_t|s_t)$ denotes the probability that action a_t is taken given that states s_t is the current state.

Value Functions

A value function is used to quantify the expected future reward from the current state onward under a policy. The value function is quantified by a recursive equation based on the reward, transitions probabilities, and value function at the next state. The value function with respect to a policy π is denoted by $V^\pi : \mathcal{S} \to \mathbb{R}$ and given by

$$V^\pi(s_t) = \sum_{a_t \in \mathcal{A}(s_t)} \pi(a_t|s_t) \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + V^\pi(s_{t+1})), \tag{2.2}$$

where $R(s_t, s_{t+1}, a_t)$ is the expected reward from the transition and $V^\pi(s_{t+1})$ denotes the value function for the next state.

Another useful representation is the state-action value function, which defines the expected reward from a state-action pair to the end of the horizon. The state-action value function is denoted by $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and is given by

$$Q^\pi(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + \sum_{a_{t+1} \in \mathcal{A}(s_{t+1})} \pi(a_{t+1}|s_{t+1}) Q^\pi(s_{t+1}, a_{t+1})), \tag{2.3}$$

where $Q^{\pi}(s_{t+1}, a_{t+1})$ denotes the state-action value function for the next state-action pair. The value function for a terminal state is given by a real number, while the state-action value function retrieves a real number to the state-action pair that leads to a terminal state. This real value may be deterministic or stochastic depending on the problem. In the above definitions of the value and state-action value functions a discount factor of 1 is assumed. As the problems in this thesis are all finite horizon, the functions are well defined with this discount factor.

2.1.2 Dynamic Programming

This section introduces the set of DP algorithms that are relevant to this thesis. Specifically, value iteration, rollout, and tree search are introduced.

Exact Dynamic Programming

The value functions as described in Equation (2.2) can also be used in order to obtain optimal control. This value function is known as the optimal value function and maximizes the expected reward from the state onward for any action and can even be used to obtain optimal control. In contrast to Equation (2.2), the optimal value function does not require a policy π to be computed. The optimal value function is denoted by V^* and given by

$$V^*(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + V^*(s_{t+1})). \tag{2.4}$$

This recursive equation is known as the Bellman equation and can be solved by using value iteration [8]. Value iteration is done by initializing a value function $V(s_t) \leftarrow 0$, then iteratively applying the update

$$V(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + V(s_{t+1})) \tag{2.5}$$

until convergence. Using the optimal value function, the optimal action, a_t^*, for the state s_t is then given by

$$a_t^* = \arg \max_{a_t \in \mathcal{A}(s_t)} \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + V^*(s_{t+1})). \tag{2.6}$$

Although value iteration can be used to obtain an optimal policy, it is often slow to converge and even intractable in many practical examples due to the curse of dimensionality with respect to the number of states and transitions. The curse of dimensionality was coined by Bellman and refers to the fact that in many problems, the computational requirements grow exponentially with the dimension of the problem. For this reason, there has been a significant amount of research exploring methods for approximating the value function or directly deriving approximate control policies.

Rollout

Rollout is an approach which can be used in order to directly estimate which action to take at a given state. The idea behind rollout is to simply determine which action is expected to give a higher reward from the current state, assuming a so called base policy π is followed. More specifically, to determine which action to take, one can simply look at each action and simulate an equal number of trajectories which start by taking that action, then choose the action with the largest total reward obtained by simulation. Pseudo-code of the rollout scheme is presented in Algorithm 1.

Algorithm 1: Rollout

```
Input: $N$ - number of episodes to simulate before taking action
       $\pi$ - base policy for simulation
1  procedure Rollout($N$, $\pi$):
2      Observe an initial state $s_0$
3      $t \leftarrow 0$
4      while $s_t$ is not terminal do
5          $r_{a_t} \leftarrow 0$, for $a_t \in \mathcal{A}(s_t)$
6          for $a_t \in \mathcal{A}(s_t)$ do
7              for $i = 1, 2, \ldots, N$ do
8                  Take action $a_t$, observe $r_{t+1}$, $s_{t+1}$
9                  Use $\pi$ to sample a trajectory $a_{t+1}, r_{t+2}, s_{t+2}, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
10                 $r_{a_t} \leftarrow r_{a_t} + \sum_{\ell=t+1}^{T} r_\ell$`
11         $a_t \leftarrow \arg\max_{a_t \in \mathcal{A}(s_t)} \{r_{a_t}\}$
12         Take action $a_t$, observe $r_{t+1}$, $s_{t+1}$
13         $t \leftarrow t + 1$
```

Although simple and easy to implement, there are limitations of this approach. The main one being the fact that all actions are explored equally, regardless of information collected from the first few simulations. For example, consider the case where it is obvious that taking an action a_t is suboptimal after just a few simulations, then exploring the action further is not ideal as this time could instead be used to explore states that are likely to lead to higher rewards.

Monte Carlo Tree Search

Tree-based search methods have become a popular extension to rollout as they overcome the equal exploration of all actions and loss of information as described in the previous section. The general idea is to combine strategic action selection and back up rewards observed from child nodes. Tree-based approaches generally consist of routines to traverse, expand, and backup nodes in the tree. "Traversing" controls how the tree is explored. "Expanding" determines when children are added to the tree. "Backing up" updates the values of nodes in the tree, typically with respect to a traversed path.

Tree search algorithms have been successful in many applications, however, in the case of highly stochastic settings with large state spaces, sample-based search techniques are particularly effective as they are not burdened by the curse of dimensionality. In terms of sample-based search algorithms, Monte Carlo tree search (MCTS) [9] has become one of the most widely used algorithms, due to the success in challenging problems such as Go [10].

In MCTS, there is a node for each state s_t in the tree. At the node corresponding to the state s_t, $N(s_t)$ represent the number of visits to the node, $Q(s_t, a_t)$ is the state-action value estimate, and $N(s_t, a_t)$ is the number of visits to the state-action pair, for each $a_t \in \mathcal{A}(s_t)$. Expansion is done by adding a new child when it is visited during simulation. To back up information in the tree, $N(s_t)$ and $N(s_t, a_t)$ are incremented when visited and the state-action value function, $Q(s_t, a_t)$, is updated based on the observed reward. The traversal in MCTS divided into two cases: (1) a tree-policy, which uses the state-action value estimate to traverse node which are in the tree, and (2) a simulation policy, which is deployed for nodes which have not yet been added to the tree. Action selection when traversing nodes in the tree can be defined in various ways, but to balance the trade-off between exploration and exploitation upper confidence bounds applied to trees (UCT) [11] is commonly chosen. UCT selects an action by

$$a_t^* = \arg\max_{a_t \in \mathcal{A}(s_t)} Q(s_t, a_t) + c\sqrt{\frac{2\log(N(s_t))}{N(s_t, a_t)}}, \tag{2.7}$$

where $c \in \mathbb{R}$ is chosen to balance exploitation and exploration. As c approaches 0 the selection will be done only using the state-action value function and is considered to be exploitative. When c approaches ∞, the least explored action will always be chosen regardless of the state-action value function.

Pseudo-code for MCTS with UCT selection is presented in Algorithm 2 where `MCTS` is the main function which determines control over an episode. At each state, the tree is updated through `Rollout`, which simulates N episodes and propagates the rewards received until the end of the horizon. The choice of simulating over N episodes is arbitrary and could easily be replaced with a time budget if desired. `Simulate` controls how the tree is traversed in an episode. For states in the tree, the tree policy `UCTSelect` is followed, while the base policy π if not. Nodes are also added to the tree during simulation through `Expand`, which adds the node and initializes the count and state-action value estimates. `Backup` uses the trajectory obtained through simulation and updates counts and state-action value estimates at each node in the trajectory. `UCTSelect` implements the UCT action selection as described in Equation (2.7).

Algorithm 2: Monte Carlo Tree Search

```
Input: $N$ - number of episodes in each rollout
       $\pi$ - base policy for simulation
       $c$ - exploration constant
procedure MCTS($N$, $\pi$, $c$):
    Observe $s_0$
    $t \leftarrow 0$
    while $s_t$ is not terminal do
        Rollout($s_t$, $t$, $N$, $c$)
        $a_t \leftarrow \arg\max_{a_t \in \mathcal{A}(s_t)} Q(s_t, a_t)$
        Take action $a_t$, observe $s_{t+1}, r_{t+1}$
        $t \leftarrow t + 1$
procedure Rollout($s_t$, $t$, $N$, $c$):
    for $i = 1, \ldots, N$ do
        $\{s_t, a_t, r_{t+1}, s_{t+1}, \ldots, s_{T-1}, a_{T-1}, r_T, s_T\} \leftarrow$ Simulate($s_t$, $t$, $c$)
        Backup($\{s_t, a_t, r_{t+1}, s_{t+1}, \ldots, s_{T-1}, a_{T-1}, r_T, s_T\}$)
```

```
13 procedure Backup($\{s_t, a_t, r_{t+1}, s_{t+1}, \ldots, s_{T-1}, a_{T-1}, r_T, s_T\}$):
14     $r \rightarrow 0$                                   // reward from period $t$ onward
15     for $i = T-1, T-2, \ldots, t$ do
16         $r \leftarrow r + r_{i+1}$
17         $N(s_i) \leftarrow N(s_i) + 1$
18         $N(s_i, a_i) \leftarrow N(s_i) + 1$
19         $Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \frac{r - Q(s_i, a_i)}{N(s_i, a_i)}$
20 procedure Simulate($s_t$, $t$, $c$):
21     $T \leftarrow t$
22     while $s_T$ is not terminal do
23         if $s_T$ in tree then
24             $a_T \leftarrow$ UCTSelect($s_T$, $c$)
25         else
26             Expand ($s_T$)
27             Choose $a_T$ according to $\pi$
28         Take action $a_T$, observe $r_{T+1}, s_{T+1}$
29         $T \leftarrow T + 1$
30     return $\{s_t, a_t, r_{t+1}, s_{t+1}, \ldots, s_{T-1}, a_{T-1}, r_T, s_T\}$
31 procedure Expand($s_t$):
32     $N(s_t) \leftarrow 0$
33     $N(s_t, a_t) \leftarrow 0, \forall a_t \in \mathcal{A}(s_t)$
34     $Q(s_t, a_t) \leftarrow \infty, \forall a_t \in \mathcal{A}(s_t)$
35     Add $s_t$ to tree
36 procedure UCTSelect($s_t$, $c$):
37     return $\arg\max_{a_t \in \mathcal{A}(s_t)} Q(s, a) + c\sqrt{\frac{2 \log N(s_t)}{N(s_t, a_t}}$
```

2.1.3 Reinforcement Learning

In recent years, RL algorithms have become widely adopted for deriving control in sequential decision-making problems. Typically, RL algorithms use simulation to approximate the value function, state-action value function, or a control policy. These approximations are often derived offline in a training phase and following this can be applied directly to obtain efficient real-time control. One advantage of RL algorithms is that they typically do not need explicit knowledge of the MDP dynamics. Generally, RL algorithms can be model-free or model-based, where model-free algorithms learn a control policy and model-based algorithms model information about the underlying MDP such as the transition and reward functions. This section introduces the necessary background and approaches for value function and state-action value function approximation used in the context of this thesis. For further detail on the algorithms presented in this section, the reader is referred to [7].

Monte Carlo Methods

Policy evaluation is a fundamental concept in RL which quantifies the expected reward from a given state under a given policy. Often this is done by simulation of a policy in an environment and many of the ideas from policy evaluation can be extended to derive control. One of the most basic methods for evaluating a policy is Monte Carlo (MC) evaluation. It works by simulating a policy π for a large number of episodes. The value function is then approximated using the average return from the state until the end of the episode. Pseudocode for MC evaluation is given in Algorithm 3, where $N(s)$ stores the number of visits to each state. In turn it which can then be used to update the value function estimate directly, avoiding the need to store an entire list of rewards for each state. This particular version of MC evaluation is known as every-visit MC since the value function is updated every time the state is visited in the trajectory. First-visit MC is an alternative approach that updates the state only on the first visit in the trajectory.

Algorithm 3: Monte Carlo Evaluation

```
Input: $N$ - number of episodes in each rollout
       $\pi$ - policy to evaluate
Output: $V^\pi$ - value function estimate for the policy $\pi$
procedure MC($N$, $\pi$):
    Initialize $V^\pi(s)$ randomly $\forall s \in \mathcal{S}$
    Initialize $N(s) \leftarrow 0$, $\forall s \in \mathcal{S}$        // number of visits to $s$
    for $i = 1, 2, \ldots, N$ do
        Use $\pi$ to generate the episode $s_0, a_0, r_1, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
        $r \leftarrow 0$        // reward from period $t$ onward
        for $t = T-1, T-2, \ldots, 0$ do
            $r \leftarrow r + r_{t+1}$
            $N(s_t) \leftarrow N(s_t) + 1$
            $V^\pi(s_t) \leftarrow V^\pi(s_t) + \frac{r - V^\pi(s_t)}{N(s_t)}$
    return $V^\pi$
```

MC evaluation is a natural approach, but unfortunately requires a large number of episodes to converge. In general, the value function estimation from MC is unbiased but suffers from high variance.

Temporal Differences

Temporal difference (TD) methods have been a widely successful alternative to MC in many tasks. This section presents TD policy evaluation, which is a fundamental concept required for defining the control algorithm presented in the following section. The main idea behind TD is to update the value function in each step, rather than at the end of the episode. This is done through bootstrapping, which uses the observed reward and next state to update the value estimate of the current states. Specifically, for each transition, the value function is updated based on the difference between the observed reward plus value estimate of the next state and the value estimate of the current state. This difference is often referred to as the TD error and is given by

$$\delta_t = r_{t+1} + V^{\pi}(s_{t+1}) - V^{\pi}(s_t). \tag{2.8}$$

Using the TD error, TD evaluation can then be done through simulation. Specifically, a policy π is followed and in each step the value function estimate is updated based on the TD error. The magnitude of the update is controlled by a step-size parameter, $\alpha \in \mathbb{R}$, which determines how much the state-action value estimate will change based on the TD error. The policy is simulated until a sufficient value function is obtained or a computational budget is reached. Pseudo-code for TD evaluation of a policy π is given in Algorithm 4. The estimations from temporal differences tend to have low variance, but are biased, contrasting that of MC.

SARSA

State-Action-Reward-State-Action (SARSA) [12] was introduced as a method to use the notion of TD error to directly derive a control policy. SARSA is an on-policy algorithm, which combines a policy derived from the state-action value function estimate and randomization in order to ensure exploration. In this thesis, the exploration-exploitation trade-off is handled with an ϵ-greedy approach that greedily selects the action which maximizes the state-action value function with probability $1 - \epsilon$ and takes a random action with probability ϵ. SARSA follows a similar implementation to that of TD, with the only difference being that ϵ-greedy

Algorithm 4: Temporal Difference Evaluation

```
Input: $N$ - number of episodes
       $\alpha$ - the step size
       $\pi$ - policy to evaluate
Output: $V^\pi$ - value function estimate for the policy $\pi$
1  procedure TD($N$, $\alpha$, $\pi$):
2      Initialize $V^\pi(s)$ randomly $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
3      $V^\pi(s) \leftarrow 0$, for all terminal states
4      for $i = 1, 2, \ldots, N$ do
5          Observe $s_0$
6          $t \leftarrow 0$
7          while $s_t$ is not terminal do
8              Choose $a_t$ according to $\pi$
9              Take action $a_t$, observe $r_{t+1}$, $s_{t+1}$
10             $V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(r_{t+1} + V^\pi(s_{t+1}) - V^\pi(s_t))$
11             $t \leftarrow t + 1$
12     return $V^\pi$
```

policy is used and estimates the state-action value function, rather than just the value function. Pseudocode of SARSA and ϵ-greedy action selection is provided in Algorithm 5.

The state-action value function obtained from SARSA can then be used to obtain control at a state s_t by the action which maximizes Q, i.e.,

$$a_t = \arg \max_{a_t \in \mathcal{A}(s_t)} Q(s_t, a_t). \tag{2.9}$$

Function Approximation

The value function approximations up to this point still suffer from one major limitation as they are tabular, meaning that information for each state needs to be stored and similar states will not share information. As many practical sequential decision-making problems have an intractable number of states, tabular methods are unusable due to the potential to reach never-before-seen states and the huge memory requirement to store the known states.

Function approximation can be used in order to overcome this issue. This is done by parameterizing the value function, which can be done through more traditional techniques such as linear approximation [7], or more modern techniques like neural networks that will be presented in Section 2.3.3.

Algorithm 5: SARSA

```
Input: $N$ - number of training episodes
       $\alpha$ - the step size
       $\epsilon$ - rate to take random action
Output: $Q^\pi$ - state-action value function estimate
1  procedure SARSA($N$, $\alpha$, $\epsilon$):
2      Initialize $Q(s, a)$ randomly $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
3      $Q(s, \cdot) \leftarrow 0$, for all terminal states
4      for $i = 1, 2, \ldots, N$ do
5          Observe $s_0$
6          $a_0 \leftarrow$ $\epsilon$-greedy$(Q, s_0, \epsilon)$
7          $t \leftarrow 0$
8          while $s_t$ is not terminal do
9              Take action $a_t$
10             Observe $r_{t+1}$, $s_{t+1}$
11             $a_{t+1} \leftarrow$ $\epsilon$-greedy$(Q, s_{t+1}, \epsilon)$
12             $Q(s_t, a_t) \leftarrow \alpha(r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
13             $t \leftarrow t + 1$
14     return $Q$
15 procedure $\epsilon$-greedy($Q$, $s$, $\epsilon$):
16     Sample $p \sim U(0, 1)$
17     if $p < \epsilon$ then
18         return $a \in \mathcal{A}(s)$ at random
19     return $\arg\max_{a \in \mathcal{A}(s)} \{Q(s, a)\}$
```

2.1.4 T-stage Optimal Control

In this thesis, the sequential decision-making problems are all T-stage optimal control problems, meaning a terminal state is reached after exactly T decisions are made. The same set of algorithms can be applied, although some discrepancies in notation are present. The most notable of which is with respect to the value function which is commonly represented with respect to time. This type of value function is denoted by $V_t^\pi : \mathcal{S} \to \mathbb{R}$ and given by

$$V_t^\pi(s_t) = \sum_{a_t \in \mathcal{A}(s_t)} \pi(a_t|s_t) \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)(R(s_t, s_{t+1}, a_t) + V_{t+1}^\pi(s_{t+1})), t = 1, \ldots, T, \quad (2.10)$$

where $V_{T+1}^\pi(s_{T+1})$ is defined by a real number, which can be deterministic or the expected value of some distribution.

In general, the approaches described in the previous sections can be applied directly or modified slightly to accommodate the time period. If the MDP can be represented as an ordered graph, the optimal value function can be computed with backward induction by evaluating $V_T(s), V_{T-1}(s), \ldots, V_1(s)$ for all states $s \in \mathcal{S}$. Rollout can be applied directly. For MCTS, the time period can be added to the state information at each node. For functional approximation either a set of parameters for each period can be used or alternatively the period can be incorporated into the state information. For a more complete discussion on T-stage optimal control the reader is referred to [13].

2.2 Mathematical Optimization

Mathematical programming is an approach that is used to mathematically model and find solutions to decision-making problems. In capacity control problems, mathematical programming is crucial to efficiently determine solutions to the end-of-horizon operational problem. There are also direct applications of mathematical programming to deriving approximate value functions. This section introduces the fundamental concepts of both linear and integer programming used throughout the remainder of this thesis.

A mathematical program consists of variables, constraints, and an objective function. The goal of mathematical programming is to determine an assignment of values to the variables which minimizes or maximizes the objective function, while not violating any of the constraints. In general, objective functions and constraints can be arbitrary functions and depending on the type of function, special algorithms are required to solve the mathematical program.

2.2.1 Linear Programming

A mathematical program is considered to be a linear program (LP) if both the constraints and the objective function are linear and the decison variables are continuous. Despite their simplicity, LPs have been an effective approach at modeling real world problems. A LP can be expressed in canonical form as

$$\begin{aligned} \max \quad & \mathbf{c}^\intercal \mathbf{x} && (2.11) \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} && (2.12) \\ & \mathbf{x} \geq \mathbf{0} && (2.13) \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the set of variables, $\mathbf{c} \in \mathbb{R}^n$ is the vector of coefficients of the linear objective function, and $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ define the coefficients of the constraint set. To characterize a solution to the original LP as optimal, duality is used. The dual of the above linear program can be expressed as

$$\begin{aligned} \min \quad & \mathbf{b}^\intercal \mathbf{y} && (2.14) \\ \text{subject to} \quad & A^\intercal \mathbf{y} \geq \mathbf{c} && (2.15) \\ & \mathbf{y} \geq \mathbf{0} && (2.16) \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^m_+$. A solution to the original LP, $\mathbf{x}^*$, is optimal if there exists a solution to the dual linear program $\mathbf{y}^*$ such that $\mathbf{c}^\intercal\mathbf{x}^* = \mathbf{b}^\intercal\mathbf{y}^*$. This property is known as strong duality.

A variety of approaches have been proposed for solving LPs, but the simplex algorithm, developed by George Dantzig [14], has been the most widely adopted in practice. If the LP is bounded and feasible, then an optimal solution will exist on one of the extreme points of the polyhedron defined by the constraints. The simplex algorithm works by iterating over the extreme points with increasing objective value until an optimal solution is found. In the worst case, the running time of the simplex algorithm is exponential, however, in practice, it often performs well, especially within the context of the iterative LP solving required by the branch-and-bound algorithm. Other algorithms such as interior point and ellipsoid methods offer alternative approaches with guaranteed polynomial running time [15].

2.2.2 Mixed Integer Linear Programming

Many important problems in practice require integer decision variables. This results in a mixed-integer program (MIP) problem and if the constraints and objective functions remain linear a mixed-integer linear program (MILP). The general formulation for a MILP is given by

$$\begin{aligned} \max \quad & \mathbf{c}^{\intercal}\mathbf{x} && (2.17)\\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} && (2.18)\\ & \mathbf{x} \geq \mathbf{0} && (2.19)\\ & \mathbf{x}_i \in \mathbb{Z} \qquad \forall i \in \mathcal{I} && (2.20) \end{aligned}$$

where $\mathcal{I} \subseteq \{1, \ldots, n\}$ denotes the subset of integer variables. This formulation can also express a binary variable, which is a special case and often used to model binary decisions. Although the formulation is similar to that of a LP, solving a MILP is $\mathcal{NP}$-hard [16], meaning that no polynomial time algorithm is known. Despite this, there has been significant progress in the last several decades dedicated to the development of MILP solvers such as CPLEX [17], Gurobi [18], and SCIP [19], which have been successful in solving many large-scale MILP problems. These solvers are based on the branch-and-bound algorithm [20], executed in conjunction with several other algorithmic components like cutting planes, preprocessing, and primal heuristics.

Branch and Bound

A key concept within the branch-and-bound algorithm is the LP relaxation, which is simply a relaxation of the original MILP obtained by removing integrality constraints of the variables. Branch and bound is a tree-based-search method that is used to explore the search space and find optimal solutions to a MILP problem. In branch and bound, each node consists of an LP relaxation. A node is a leaf if: (i) the LP relaxation is infeasible, (ii) the solution to the LP relaxation, x^*, satisfies the integrality constraints defined by $\mathcal{I}$, or (iii) there is a known MILP feasible solution $\bar{x}$ which satisfies $c^T\bar{x} \geq c^T x^*$ since exploring further will not result in better solutions. If x^* does not satisfy either of the previous three conditions, then a fractional variable $j \in \mathcal{I}$ is chosen and two child nodes are defined by adding either the constraint $x_j \leq \lfloor x_j^* \rfloor$ or the constraint $x_j \geq \lceil x_j^* \rceil$ to the LP relaxation. This process is repeated until every node is a leaf, at which point an optimal solution and proof of optimality are obtained.

Branch and bound alone is often not efficient enough to solve many MILP instances, but it can be improved upon using the concepts highlighted in the next subsections [21].

Cutting Planes

The general idea behind cutting planes is to tighten the LP relaxation by additional constraints without removing any of the integer points in the original MILP. Cutting planes can be added to the root LP or at each node in the tree. One particularly effective approach is to add cutting planes after solving the LP relaxation at each node. This is referred to as the branch-and-cut algorithm.

Preprocessing

Preprocessing is another component which has had a significant impact on the performance of MILP solvers. Preprocessing is done at the beginning of the solving process to simplify the MILP problems. This can be done by removing redundant constraints and variables, tightening bounds, and recognizing special structures within the constraints, which allows for stronger inequalities to be added.

Primal Heuristics

Primal heuristics are designed to find solutions to the original MILP throughout the branch-and-bound process. Finding good quality solutions early in the branch-and-bound process allows for pruning the tree since there will be more nodes that satisfy condition (iii) as described in the branch-and-bound algorithm.

2.3 Supervised Learning

In recent years, machine learning (ML) has gained significant interest due to the success in many challenging applications such as natural language processing and computer vision [22] and Atari games [23]. ML is focused on developing algorithms which learn through experience and observation rather than relying on hard-coded rules. Depending on the nature of the observation, an ML problem can typically be categorized into either supervised learning, unsupervised learning, or RL. Supervised learning is concerned with learning a mapping between an input and an output space through a collection of labelled samples. Unsupervised learning seeks to find structure and provide insights for a collection of data where only input information exists. As discussed in Section 2.1, RL is used to solve sequential decision-making problems.

In this thesis, supervised learning is an important component as it is used to predict the objective function value of MILP problems using features derived from the instance. Supervised learning tasks can typically be characterized as either classification or regression depending on the set of outputs. A classification task occurs when the set of outputs is a discrete, while a regression task is when outputs are continuous. Predicting the objective value of a MILP is a regression task as it is continuous.

The remainder of this section presents a general supervised learning framework, evaluation metrics, and the set of algorithms used within the context of this thesis. For more details on any of the topics discussed in the remainder of this section, the reader is referred to [24].

2.3.1 Formulation

To formally define the task of supervised learning, let $\mathcal{X}$ and $\mathcal{Y}$ be the set of input and outputs, respectively, and $P_{\mathcal{X},\mathcal{Y}}$ be the distribution for which input-output pairs are sampled from. The objective of supervised learning is to choose a function $f : \mathcal{X} \to \mathcal{Y}$ from a class of functions $\mathcal{F}$, which minimizes the expected error between the prediction, $f(x)$, and y for all pairs $(x, y) \sim P_{\mathcal{X},\mathcal{Y}}$. In general, it is not possible to evaluate the error of the expectation of $P_{\mathcal{X},\mathcal{Y}}$ and for this reason machine learning algorithms are often based on the use of samples to determine a mapping between the input and the output. Generally, it is assumed that $\mathcal{X} \subseteq \mathbb{R}^m$ and $\mathcal{Y} \subseteq \mathbb{R}$. The set of training points is given by $\{(x_1, y_1), \dots, (x_n, y_n)\}$. The m-dimensional inputs and the outputs are referred to as the features and the labels, respectively.

The general approach for developing supervised learning algorithms can typically be divided into two stages. The first stage is training, where the function $f \in \mathcal{F}$ is selected based on a set of observed training samples $\mathcal{D}^{train} = \{(x_1, y_1), \dots (x_n, y_n)\}$. The second stage is evaluation, where the function chosen in the training phase is evaluated on a separate set of observed test samples $\mathcal{D}^{test} = \{(x_1, y_1), \dots (x_\ell, y_\ell)\}$. In many cases, it is quite useful to use a set aside subset of the training data as a validation set to understand how the trained model is expected to perform on unseen data. This is typically done to further tune algorithm or model parameters before evaluation on the test data. In practice, the validation set can simply be a subset of the training data or more elaborate techniques such as k-fold cross validation [25] can be used.

Ideally, a trained model would perform well on the entire distribution of input-output pairs, however, this is often not trivial due to underfitting and overfitting. In practice, the train and test set can be used to evaluate the expected generalization of the model. Underfitting occurs when the trained model performs poorly on both the train and test data and can typically be overcome by choosing a model which can express a greater class of functions. Overfitting occurs when the training error is low and the test error is high. This often occurs when the model is able to memorize the training data rather than extract underlying relationships. One approach to overcome overfitting is to choose a less expressive model, while another is to bias the model during training. The latter process is referred to as regularization and is done in a model specific manner.

2.3.2 Loss Functions

A loss function is used to quantify the error between the predictions and the true values. In general, the loss function is a function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The loss from a function f on a sample $(x, y) \sim P_{\mathcal{X},\mathcal{Y}}$ is given by $L(y, f(x))$. The loss over a set of samples, $\mathcal{D}$, is given by

$$\frac{1}{|\mathcal{D}|} \sum_{(x_i,y_i)\in\mathcal{D}} L(f(x_i), y_i). \tag{2.21}$$

The function L can be defined based on the task and can even be problem specific. As regression is the learning task in this thesis, mean squared error (MSE) and mean absolute error (MAE) are used in the evaluation. The definitions for MSE and MAE are given in (2.22) and (2.23) respectively,

$$\text{MSE}(f, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x_i,y_i)\in\mathcal{D}} (f(x_i) - y_i)^2, \tag{2.22}$$

$$\text{MAE}(f, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x_i,y_i)\in\mathcal{D}} |f(x_i) - y_i|, \tag{2.23}$$

where f is the is the trained model and $\mathcal{D}$ is the set of data to evaluate performance on.

2.3.3 Algorithms

For supervised learning, there is often a trade off between the complexity of the class of functions which an algorithm can select from and the generalization to the test set. This section introduces the set of algorithms used in this thesis.

Linear Regression

Linear regression is a simple method which limits the class of functions to be a linear combination of the input features. Let $\hat{x}_i$ be the $(m{+}1)$-dimensional vector given by the concatenation of x_i and 1. This concatenation is done to introduce a intercept term into the linear combination of weights. Let X be the $n \times (m+1)$ dimensional matrix obtained by concatenating $\hat{x}_1, \ldots, \hat{x}_n$ and Y be the n dimensional vector obtained by concatenating $y_1, \ldots, y_n$. Linear regression defines a hyperplane that minimizes the MSE by

$$\min_{\mathbf{w}} ||Y - X \cdot \mathbf{w}||_2^2, \tag{2.24}$$

for $w \in \mathbb{R}^{m+1}$. This can be solved analytically by

$$\mathbf{w}^* = (XX^\intercal)X^\intercal Y, \tag{2.25}$$

however, in practice the optimization problem is often solved with gradient descent methods, such as stochastic gradient descent (SGD), which is described in more detail in the section on neural networks. The prediction on any $x \in \mathcal{X}$ is then given by $f(x) = x^\intercal \cdot \mathbf{w}^*_{0:n} + \mathbf{w}^*_{n+1}$, where $\mathbf{w}_{0:n}$ is the first n entries of $\mathbf{w}^*$ and $\mathbf{w}^*_n$ is the last entry of w^*. The main drawback of linear regression is that it cannot represent a nonlinear relationship between the input and the output, which, in many applications is an important limitation. Due to the limited capacity of linear models, many alternatives have been proposed.

Decision Trees & Random Forests

Decision trees [26] and random forests [27] have become a popular alternative to linear regression due to the simplicity of training and generally good performance. As the name suggests, decision trees are based on the idea of building a tree to estimate the label or regression targets based on the features. At the root node, a feature, $j \in \{1, \ldots, m\}$ is selected and two children are defined based on splitting the feature domain over a threshold z_j. The training data is then split such that the left and right children contain the set of points $\{x_i | x_{ij} <= z_j\}$ and $\{x_i | x_{ij} > z_j\}$, respectively. Here, x_{ji} denotes the j-th feature of the training point x_i. The process is then repeated until a stopping condition such as maximal depth or maximal number of leaf nodes is reached. When choosing a feature j and its threshold z_j, it is ideal to define a splitting such that the labels in each child become more similar. In practice, the similarity is quantified by metrics such as the Gini index [26] or information gain [28].

In machine learning ensemble methods are an approach which combines several trained models in order to achieve better performance. Typically, this is either done through bagging or boosting which combine models in order to improve the variance and bias. Random forests [27] are based on the idea of bagging, which combines several decision trees trained on randomly sampled subsets of the training data. The regression is then defined by the mean of the predictions among all of the trees in the ensemble.

Neural Networks

In recent years, neural networks and deep learning have become successful in a wide variety of applications. Neural networks are known to be universal function approximators [29]. The potential to approximate any function in combination with efficient parallel linear algebra operations has allowed neural networks to take advantage of large scale data.

Neural networks extend the idea of linear regression by combining several matrix multiplications with a non-linear activation function between each multiplication. Mathematically, an ℓ-layer neural network can be expressed as

$$h^{(1)} = \sigma(W^{(0)} \cdot x + b^{(0)}) \tag{2.26}$$

$$h^{(i+1)} = \sigma(W^{(i)} \cdot h^{(i)} + b^{(i)}), i = 1, \dots \ell - 1 \tag{2.27}$$

$$\hat{y} = \sigma(W^{(\ell)} \cdot h^{(\ell)} + b^{(\ell)}) \tag{2.28}$$

where $x \in \mathbb{R}^m$ is the input, $\hat{y} \in \mathbb{R}$ is the prediction, $h^{(i)} \in \mathbb{R}^{d_i}$ is the i-th hidden layer, $W^{(i)} \in \mathbb{R}^{d_i \times d_{i+1}}$ is the matrix of weights from layer i to $(i+1)$, $b^{(i)} \in \mathbb{R}^{d_i}$ is the bias at the i-th layer, and σ is a non-linear activation function. A neural network can be decomposed into the input, hidden, and output layers, where Equations (2.26), (2.27), and (2.28) represent them, respectively.

Backpropagation [30] is an algorithm that uses the gradient of the loss function to recursively compute the gradient with respect to the weights and biases of each layer. The weights and bias are then updated in the direction opposite to the gradient to minimize the loss function on the training data. SGD and variants of SGD are the most commonly used approaches used in the optimization of neural networks. SGD randomly samples a subset of the training data, computes predictions, computes the gradient of the loss for the predictions and true values, and updates the weights in the opposite direction of the gradient. Batches of the training data are randomly sampled until a stopping criteria is reached.

As mentioned previously, a neural network can be used to approximate any function. Generally, a larger network is able to approximate more complex functions. This is often referred to as the capacity. The capacity can be increased by increasing either the number of hidden layers or the dimension of the hidden layers.

A variety of other architecture decisions are possible with neural networks. Two of the main choices being the activation function and regularization. The choice of activation function is typically not a crucial decision and activation functions such as rectified linear unit (ReLU), given by the max of 0 and the input, generally perform well. Regularization on the other hand can be a very important component. Given the high capacity of neural networks, overfitting on the training data is a common occurrence. To limit overfitting, methods such as L1 or L2 regularization on the magnitude of the weights, dropout [31], and early stopping can be used. The reader is referred to [32] for a more complete discussion of neural networks.

2.4 Booking Control

This section introduces a general capacity control formulation as a T-stage optimal control problem, similar to that described in [6], and presents some of the most common approaches for obtaining control policies.

2.4.1 Problem Formulation

Let $\mathcal{N}$ denote a set of requests with cardinality $|\mathcal{N}| = n$. The decision-making problem is defined over T periods indexed by $t \in \{1, \ldots, T\}$. The probability that request $j \in \mathcal{N}$ is made at time t is given by p_j^t and the probability that no request occurs at period t is given by p_0^t. The time intervals are defined to be small enough such that at most one request is received in each period. The revenue associated with accepting a request of type j is given by r_j, and the reward function is given by $R : \mathcal{N} \times \{0, 1\} \rightarrow \mathbb{R}$, where $R(j, a) = r_j a$. This reward function gives a reward of r_j if the request is accepted and 0 otherwise.

To formulate this as a MDP, let s_j^t denote the number of requests of tyype j accepted before time t and $\mathbf{s}_t \in \mathbb{R}^n, t = 1, \ldots, T+1$, denote the state vector where the j-th index is given by s_j^t. Furthermore, let $\mathcal{S}_t$ denote the set of all possible states at period t. A request can only be accepted or rejected, thus the set of actions is given by $\mathcal{A} = \{0, 1\}$, with 1 being to accept and 0 to reject. The action is denoted by $a_j \in \{0, 1\}$, where $a_j = 1$ if an offer for request j

is accepted at time t. The deterministic transition is given by $\mathbf{s}_{t+1} = \mathbf{s}_t + a\mathbf{e}_j$, with $\mathbf{e}_j \in \mathbb{R}^n$ having a j-th component equal to 1 and every other equal to 0.

An operational decision-making problem occurs at the end of the booking period and is related to the fulfilment of the accepted requests. This problem only depends on the state at the end of the time horizon, $\mathbf{s}_{T+1}$. The cost given by the end-of-horizon problem is denoted by $\Gamma : \mathbb{R}^n \to \mathbb{R}_-$.

With the above definitions, the value function is defined by

$$V_t(\mathbf{s}_t) = p_0^t V_{t+1}(\mathbf{s}_t) + \sum_{j \in \mathcal{N}} p_j^t \max_{a_j \in \{0,1\}} \{r_j a_j + V_{t+1}(\mathbf{s}_t + a_j \mathbf{e}_j)\}, t = 1, \dots, T, \tag{2.29}$$

$$V_{T+1}(\mathbf{s}_{T+1}) = \Gamma(\mathbf{s}_{T+1}), \tag{2.30}$$

with $\mathbf{s}_1 = \mathbf{0}$. This formulation is general enough to capture a wide variety of capacity control problems. In applications where the constraints cannot be violated Γ can simply be $-\infty$ if constraints are violated and 0 otherwise, while in applications such as [5, 33], Γ can represent the cost associated with fulfillment of the accepted requests and penalties for overbooking.

2.4.2 Algorithms

As discussed in Chapter 1, booking limit and bid-price policies are commonly used approaches for determining policies in capacity control problems. Booking limit and bid-price policies are often obtained through LP-based approximations that leverage the structure of the end-of-horizon problem and capacity constraints directly. These LP-based approaches have by far been the most studied and widely adopted approaches in practice. In this section, two of the most commonly used approaches for obtaining control in RM are presented to the general formulation introduced in the previous section. These approaches are generally applicable to most capacity control RM problems; however, simplifying assumptions and approximations typically need to be made in order to account for the end-of-horizon operation problem.

Deterministic Linear Programming

Deterministic linear programming (DLP) can be used to obtain both booking limit and bid-price policies through a LP that directly models capacity constraints and the cost associated with the fulfillment of the requests. The LP is formulated to maximize the quantity of each request type to accept without violating capacity constraints. Specifically, the DLP is given by

$$\begin{aligned} \max \quad & \sum_{j \in \mathcal{N}} r_j \cdot x_j && (2.31) \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} && (2.32) \\ & \mathbf{0} \leq \mathbf{x} \leq \mu && (2.33) \end{aligned}$$

where $\mu \in \mathbb{R}^n$ is the vector of expected demand for each request type, $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ denote the capacity constraints, and $\mathbf{x} \in \mathbb{R}^n$ is the variable that provides the quantity of each request type to accept. Booking limit control can be defined directly through the primal solution by not accepting more requests than x^* for each type. Bid-price control can be obtained by determining if the revenue from accepting a request is greater than a function of the dual variables and constraints. This approach does not account for time and realizations of the requests, however, the LP can be re-solved throughout the booking period to provide updated primal and dual solutions. DLP generally works well and is widely used in practice [34]. Randomized linear programming (RLP) is a popular alternative approach to DLP that uses sample trajectories rather than expectations to formulate the LP can be formulated using a set of sampled trajectories. The reader is referred to [6] for a more detailed description on RLP.

Approximate Dynamic Programming

Approximate dynamic programming (ADP) is another approach commonly used to obtain bid-price policies. In RM the approach presented in this section is often referred to as ADP, however approximate dynamic programming within the context of optimal control and RL is unrelated. Originally introduced by Adelman [35], the idea is to approximate the linear programming formulation of the value function. The value function defined in Equations (2.29) - (2.30) can be expressed as a LP [36] given by

$$\begin{aligned} \min \quad & V_1(\mathbf{0}) && (2.34) \\ \text{subject to} \quad & V_t(\mathbf{s}_t) \geq p_0^t V_{t+1}(\mathbf{s}_t) + \\ & \sum_{j \in \mathcal{N}} p_j^t (r_j a_j + V_{t+1}(\mathbf{s}_t + \mathbf{e}_j a_j)), \forall \mathbf{s}_t \in \mathcal{S}_t, a_j \in \{0, 1\}, t \in \{1, \dots, T\}. && (2.35) \end{aligned}$$

Solving the above LP provides an optimal value function, however, in practical applications it is intractable and even more computationally challenging than exact value iteration [7].

To make the above LP tractable, one can approximate the value function by

$$V_t(\mathbf{s}_t) \approx \theta_t + \mathbf{v}_t \cdot \mathbf{s}_t, \tag{2.36}$$

where $\theta_t \in \mathbb{R}$ and $\mathbf{v}_t \in \mathbb{R}^n$ are the set of variables in the approximate optimization problem. The approximate LP formulation is then given by

$$\min_{\mathbf{v}_1,\ldots\mathbf{v}_T,\theta_1,\ldots,\theta_T} \quad \theta_1 + \mathbf{v}_1 \cdot \mathbf{0} \tag{2.37}$$

$$\begin{aligned} \text{subject to} \quad & (\theta_t + \mathbf{v}_t \cdot \mathbf{s}_t) \geq p_0^t(\theta_{t+1} + \mathbf{v}_{t+1} \cdot \mathbf{s}_t) + \\ & \sum_{j \in \mathcal{N}} p_j^t(r_j a_j + \theta_{t+1} + \mathbf{v}_{t+1} \cdot (\mathbf{s}_t + \mathbf{e}_j a_j))), \forall \mathbf{s}_t \in \mathcal{S}_t, a_j \in \{0,1\}, t \in \{1,\ldots,T\}. \end{aligned} \tag{2.38}$$

The number of constraints is the same as the LP in Equations (2.34)-(2.35), however the number of variables is significantly reduced from the entire state space to $T \times (n+1)$. To handle the large number of constraints, column generation [37], a technique for solving large scale LPs, is applied to the dual of the LP until a stopping criteria is met. Bid-price control can then be directly obtained from the primal variables. Specifically, assuming request $j \in \mathcal{N}$ arrives at time t with a current state of $\mathbf{w}_t$, the value function maximizing action is given by

$$\arg\max_{a \in \{0,1\}} \{r_j a + \theta_{t+1} + \mathbf{v}_{t+1} \cdot (\mathbf{s}_t + \mathbf{e}_j a)\}. \tag{2.39}$$

The linear approximation above can be generalized to a set basis function [35], however, approximations of this form are less studied.

2.5 Related Work

This section positions the contribution of this thesis. The objective of this work is to provide a novel approach to capacity control RM through the combination of general-purpose DP or RL algorithms combined with supervised learning. The related work can be divided into three categories: (1) the application of RL to capacity control, (2) existing approaches to the set capacity control problems examined in this work, (3) the application of machine learning to mathematical optimization problems.

As discussed in Section 2.1, simulation-based approaches have been able to achieve high quality control in challenging sequential decision-making problems. Despite the success of simulation-based approaches in applications with intractable state spaces, the applications within RM have mainly been limited to airline seat allocation problems. Specifically, Gosavi et al. [38] propose a tabular TD-based algorithm for approximating the state-action value function, Lawhead et al. [39] propose a bounded actor-critic method for determining approximate control directly, and Bondoux et al. [40] propose a deep Q-learning based approach for deriving control.

A major limitation for the direct application of simulation-based approaches to capacity control problems is in the time required for simulating the system. Indeed, the computational cost associated with solving the end-of-horizon operational decision-making problem is non-negligible, which leads to a prohibitively expensive computational cost for a simulation-based approach. For this reason, the applications studied in this thesis are based on solving challenging MILP problems for the end-of-horizon problem. Specifically, the end-of-horizon problems studied in this work are the vector packing and vehicle routing, which arise in airline cargo management [4] and distributional logistics [5], respectively.

Airline cargo management was introduced by Kasilingam [4] and has since become a major focus in capacity control over recent years. Generally, each request has a weight and volume that are uncertain until the departure. At the end of the booking period the set of accepted requests need to be packed on an aircraft and if there is not enough capacity on the aircraft, the unpacked items present an off-loading penalty. This end-of-horizon problem is equivalent to vector packing or a multidimensional knapsack problem. Airline cargo management can be formulated as a single-leg or network problem, where the single-leg problem focuses on control for a single flight and the network problem is concerned with a set of flights in which cargo may need to be put on multiple flights to reach the correct destination. Amaruchkul et al. [41] propose a variety of heuristics based on DLP for single-leg airline cargo management, however off-loading penalty is simplified to linear function of the excess weight and volume. Levin et al. [42] introduce a Lagrangian relaxation of the capacity constraints in the off-loading problem which are updated during the booking period to obtain control. Levina et al. [43] introduce a infinite horizon network cargo management problem with a solution based on ADP. Barz et al. [33] examine several approaches derived from DLP and ADP for both the single-leg and the network problems. The off-loading costs given by the vector packing problem are also considered in this work.

For distributional logistics, the problem was originally introduced by Giallombardo et al. [5] and is much less studied. In this problem, collection requests are received from a set of locations. At the end of the booking period the requests must be fulfilled, resulting in a VRP. The VRP is a significantly more challenging end-of-horizon problem as the exact solution to even a single VRP can be intractable. Giallombardo et al. [5] propose a MILP formulation, similar to a DLP formulation, but with the inclusion of integer variables to model the underlying routing problem. A booking limit policy is then obtained through primal solutions to the MILP. Unfortunately, solving the MILP exactly is intractable for even small instances, thus Giallombardo et al. [5] terminate after a small number of incumbent solutions are found in the solving process.

A key aspect of this thesis is the use of machine learning to predict solutions to MILP problems. The success of machine learning in a wide range of applications has led to the growing interest within the optimization community. Machine learning for MILP problems in particular has seen a significant amount of interest [44]. Generally, machine learning approaches for optimization problems are developed to predict solutions, however, the problem of predicting objective value still has been of interest in some research. Fischetti et al. [45] use supervised learning to predict optimal objective value of MILP problems for a wind energy application. Prates et al. [46] propose a combination of supervised learning and binary search to predict optimal solution value to the traveling salesman problem (TSP).

CHAPTER 3 METHODOLOGY & PROBLEM FORMULATIONS

This section describes the methodology proposed in this thesis to obtain approximate control to booking control problems. Following the general formulation, two booking control problems from the literature are formally introduced and details regarding the application of the approach presented in this thesis are provided.

3.1 Methodology

This section introduces an approximation to the T-stage control problem formulation and describes how supervised learning can be used as an efficient method for the approximation.

3.1.1 Approximate MDP Formulation

Solving the MDP (2.29)–(2.30) can be intractable even for small instances. For this reason, approximate DP or RL appears to be a natural choice. However, these algorithms typically rely on some form of policy evaluation that consists in simulating trajectories of the system under a given policy. In the context of capacity control problems, such policy evaluation is computationally costly due to (2.30). To overcome this, the use of an approximation of Γ is proposed. Specifically, the approximation of Γ is denoted as $\phi : \mathbb{R}^m \to \mathbb{R}_-$ and given by

$$\Gamma(\mathbf{s}_{T+1}) = \phi(g(\mathbf{s}_{T+1})), \tag{3.1}$$

where $g : \mathbb{R}^n \to \mathbb{R}^m$ is a mapping from the state at the end of the horizon to an m dimensional representation. The approximate MDP is then given by

$$\tilde{V}_t(\mathbf{s}_t) = p_0^t \tilde{V}_{t+1}(\mathbf{s}_t) + \sum_{j \in \mathcal{N}} p_j^t \max_{a \in \{0,1\}} \{r_j a + \tilde{V}_{t+1}(\mathbf{s}_t + a\mathbf{e}_j)\}, t = 1, \ldots, T, \tag{3.2}$$

$$\tilde{V}_{T+1}(\mathbf{s}_{T+1}) = \phi(g(\mathbf{s}_{T+1})). \tag{3.3}$$

Here, the only change is in the value function at the end of the time horizon, however, the value will be propagated to the entire value function as it is recursive. The approximate MDP in (3.2)-(3.3) can then be solved exactly or approximately with approximate DP or RL.

3.1.2 Supervised Learning

The approximation $\phi(\cdot)$ in (3.3) can be defined in various ways, such as a problem specific heuristic, a MILP solved to a time limit or optimality gap or predicted by ML. This thesis focuses on prediction by ML and, in particular, on the use of supervised learning to develop a mapping from the end-of-horizon state to the associated operational cost. Specifically, a supervised learning model is trained offline to separate the problem of accurately predicting Γ from solving (3.2)–(3.3).

Another approach which could be considered is to learn the end-of-horizon problem and the state-action value function jointly through approximate DP or RL, rather than offline beforehand. The main motivation for choosing offline supervised learning is that it can easily be done offline before learning control and is independent of any general-purpose RL or DP algorithm used to solve the booking control problem. One potential drawback of an offline supervised learning approach is the fact that the control policy may arrive at a different distribution of operational problems than the policy that was used to obtain labeled data for supervised learning. Although this may be a major drawback of offline supervised learning, it can be mitigated by retraining with a subset of the operational problems observed during the development of control algorithms.

To train a supervised learning model, a feature mapping from the state to the input to the model and a set of labeled states at the end of the time horizon are required. The feature mapping is problem specific, but in general is viewed as the function $g(\cdot)$ in (3.3). In this thesis, two different problems are considered and details of the specific features are given in their respective sections.

To obtain labeled data, a stationary random policy which accepts a request with a given probability p is used to simulate trajectories. At the end of the time horizon, a state $\mathbf{s}_{T+1}$ is obtained and a label, $\Gamma(\mathbf{s}_{T+1})$, is computed. This process is then repeated for different values of p. The idea is to have a representation of feasible final states in the data (optimal and sub-optimal ones). The set of N labeled data points is denoted as $\mathcal{D} = \{(\mathbf{s}^1_{T+1}, \Gamma(\mathbf{s}^1_{T+1})), \ldots, (\mathbf{s}^N_{T+1}, \Gamma(\mathbf{s}^N_{T+1}))\}$.

3.2 A Distributional Logistics Problem

This section introduces a distributional logistics problem described by Giallombardo et al. [5] and discusses the details required for building a predictor for the end-of-horizon problem.

3.2.1 Problem Description

In this context, booking requests correspond to pickup activities and the operational decision-making problem to a VRP. Each pickup request has an associated location and revenue. The cost incurred at the end of the booking period is the cost of the VRP solution and hence depends on all the accepted requests.

The problem can be formulated as (2.29)–(2.30) with the boundary function described below. The end-of-horizon VRP consists of a fixed set of K_0 vehicles, each with capacity $Q \in \mathbb{R}_+$. The depot location is indexed by location 0. The set of all nodes $\mathcal{V}$ is given by $\mathcal{N} \cup \{0\}$, i.e., the union of the location requests and the depot. The set of arcs is given by $\mathcal{V} \times \mathcal{V}$ and denoted as $\mathcal{E}$. The cost of an arc is given by $c_{ij} \in \mathbb{R}_+$, for $(i,j) \in \mathcal{E}$. The optimal objective value is denoted by $z^*(\mathbf{s}, K)$, where K denotes the number of vehicles used. If more than K_0 vehicles are required, then additional outsourcing vehicles are allowed to be used at an additional fixed cost of $C \in \mathbb{R}_+$ each. Giallombardo et al. [5] do not consider the inclusion of outsourcing vehicles and instead assign an infinite operational cost if the requests cannot be fulfilled. To accomplish the same objective, C can be chosen large enough such that the cost for adding an additional vehicle is larger than any potential revenue from the requests it can fulfill. The operational cost is given by

$$\Gamma(\mathbf{s}) = \max_{K \geq K_0, K \in \mathbb{Z}} \{-z^*(\mathbf{s}, K) - C(K - K_0)\}, \tag{3.4}$$

where $z^*(\mathbf{s}, K)$ denotes the cost of the routing problem given K vehicles and $C(K - K_0)$ denotes the cost of any outsourcing vehicles used. The routing problem can be represented as a MILP and is given by

$$z^*(\mathbf{s}, K) = \min \quad \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij}\alpha_{ij}^k \tag{3.5}$$

$$\text{s.t.} \quad \sum_{(i,j)\in\mathcal{A}} \alpha_{ij}^k = \beta_i^k \qquad \forall i \in \mathcal{V}, \forall k \in \{1,\dots,K\} \tag{3.6}$$

$$\sum_{(i,j)\in\mathcal{A}} \alpha_{ji}^k = \beta_i^k \qquad \forall i \in \mathcal{V}, \forall k \in \{1,\dots,K\} \tag{3.7}$$

$$\sum_{k\in\mathcal{K}} \beta_j^k \leq 1 \qquad \forall j \in \mathcal{N} \tag{3.8}$$

$$\sum_{k\in\mathcal{K}} \beta_0^k \leq K \tag{3.9}$$

$$\sum_{i\in\mathcal{S}} \sum_{j\in\mathcal{V}\setminus\mathcal{S}} \alpha_{ij}^k \geq \beta_h^k \qquad \forall \mathcal{S} \subset \mathcal{V} : 0 \in \mathcal{S}, \forall h \in \mathcal{V}\setminus\mathbb{S}, \forall k \in \{1,\dots,K\} \tag{3.10}$$

$$q_j^k \leq QB_j^k \qquad \forall j \in \mathcal{N}, k \in \{1,\dots,K\} \tag{3.11}$$

$$\sum_{j\in\mathcal{N}} q_j^k \leq Q \qquad \forall k \in \{1,\dots,K\} \tag{3.12}$$

$$\sum_{k\in\mathcal{K}} q_j^k = s_j \qquad \forall j \in \mathcal{N} \tag{3.13}$$

$$\alpha_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \{1,\dots,K\} \tag{3.14}$$

$$\beta_i^k \in \{0,1\} \qquad \forall i \in \mathcal{V}, \forall k \in \{1,\dots,K\} \tag{3.15}$$

$$q_j^k \geq 0 \qquad \forall j \in \mathcal{N}, \forall k \in \{1,\dots,K\} \tag{3.16}$$

The decision variables are α_{ij}^k, β_i^k, and q_j^k: The binary variable α_{ij}^k equals 1 if vehicle k uses arc (i,j). The binary variable β_i^k equals 1 if vehicle k visits node i. The quantity of accepted requests from vehicle k at location j is given by the non-negative variable q_j^k. Objective function (3.5) minimizes the routing cost. Constraints (3.6) and (3.7) ensure that each vehicle goes in and out of the visited nodes. Constraints (3.8) assert that each vehicle can visit a node at most once, and (3.9) limits the number of vehicles to at most K. Constraints (3.10) ensure that every tour is connected, and (3.11) restrict vehicle k from collecting more than the capacity at location j. Finally, Constraints (3.12) guarantee that the capacity is not exceeded, and (3.13) ensure that the accepted requests at each location are fulfilled.

In order to obtain solutions to $z^*(\mathbf{s}, K)$, Giallombardo et al. [5] propose to solve the MILP to a limited number of incumbent solutions. The general MILP formulation can be quite limiting due to the number of constraints and difficulty of obtaining solutions. For this reason, leveraging VRP specific heuristics can be quite effective as they are able to obtain high quality solutions within reasonable computing times. In this thesis, FILO [47], a heuristic solver for VRPs, is used. As the MILP given in (3.5) is a nonstandard VRP formulation, some considerations need to be made. FILO is an unlimited fleet solver, so to ensure that a minimal number of vehicles are used, the depot location is offset. Furthermore, the MILP allows for splitting items on different vehicles, which is approximated by splitting demand into distinct locations at the same coordinates.

3.2.2 Prediction Task

The objective is to produce an accurate approximation of (3.4) that is fast to compute. In general, one can predict the sum of the MILP objective function value and the outsourcing cost, however, this is not necessarily the ideal for supervised learning. The outsourcing cost may result in a large change in the label with only a minor change in the features, which is not ideal for many supervised learning models. For this reason, ML is used to predict $z^*(\mathbf{s}, K)$ while the outsourcing cost, $C(K - K_0)$, is computed separately with a bin-packing solver (namely, MTP [48]).

The features are computed from the state at the end of the time horizon, $g(\cdot)$. For the sake of simplicity, a fixed-size input structure for the ML model is used. The number of locations can vary depending on the set of accepted requests. Therefore, features from capacity, depot location, total number of accepted requests per location and aggregate statistics of the locations are computed, namely, the distance between locations and the depot, and relative distances between locations are used. For each of these, we compute the min, max, mean, median, standard deviation, and 1st/3rd quartiles.

3.3 An Airline Cargo Problem

This section introduces an airline cargo management problem described by Barz et al. [33] (where [33] builds on the formulation presented by Amaruchkul et al. [41]), and discusses the details required for building a predictor for the end-of-horizon problem. Generally, an airline cargo management problem can be formulated as a single-leg or network problem. A single-leg problem is limited to cargo traveling between an origin-destination pair, while a network problem allows for cargo to travel on multiple flights within a network. This thesis focuses on the single-leg problem as it is more rigorously studied and the generalization to the network problem is trivial with the methodology presented here. As single-leg problems are considered, the MILP formulation of the end-of-horizon problem from Barz et al. [33] can be simplified to the MILP in the following section.

3.3.1 Problem Formulation

In this context, booking requests correspond to shipments that are being transported on commercial airline flights. Typically, commercial airlines have a non-negligible quantity of space remaining after passenger luggages are stored. With the excess capacity, commercial shipments are often transported between the origin and destination, resulting in a significant source of additional revenue. Each request is considered to be of a specific class, such as electronics, fresh food, or raw materials. The cost incurred at the end of the booking period is given by a vector packing problem [49]. Here, the cost is given by the set of accepted requests which cannot be loaded due to the capacity of the aircraft and is referred to as the off-loading cost. The remainder of this section formally introduces the end-of-horizon problem.

In airline cargo, each request has an associated weight and volume and the flight has both a weight and volume capacity. All of these quantities are uncertain until the time of departure, however, it is assumed that their expected value is known. The realized weight and volume capacities of the flight are given by $Q_w \in \mathbb{R}_+$ and $Q_v \in \mathbb{R}_+$, respectively. The realized weight and volume associated with the i-th request of the j-th shipping class are denoted by w_{ji} and v_{ji}, respectively. For airline cargo, it is often convenient to express the profit and off-loading cost as a function of the weight and volume. This is done with a chargeable weight, which is defined by a weighted maximum of the weight and volume for a given request. The chargeable weight allows requests to be charged in terms of the capacity dimension which they will require a larger portion of. Specifically, the chargeable weight given by $\max\{w_{ji}, v_{ij}/\eta\}$ where $\eta \in \mathbb{R}_+$ is a constant representing the weight-volume ratio of a shipment. Using the chargeable weight, the off-loading cost is then given by $c_j \cdot \max\{w_{ji}, v_{ij}/\eta\}$. Finally, the operational cost is then given by

$$\Gamma(\mathbf{s}) = -z^*(\mathbf{s}), \tag{3.17}$$

where $z^*(\mathbf{s})$ denotes the off-loading cost associated with the vector packing problem, which is given by

$$z^*(\mathbf{s}) = \min \quad \sum_{j\in\mathcal{N}} \sum_{i=1}^{s_j} c_j \max\{w_{ji}, v_{ji}/\eta\}(1 - x_{ji}) \tag{3.18}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{N}} \sum_{i=1}^{s_j} x_{ji} w_{ji} \leq Q_w \tag{3.19}$$

$$\sum_{j\in\mathcal{N}} \sum_{i=1}^{s_j} x_{ji} v_{ji} \leq Q_v \tag{3.20}$$

$$x_{ji} \in \{0, 1\}, \quad \forall j \in \mathcal{N}, \forall i \in \{1, \ldots, s_j\} \tag{3.21}$$

The binary variables x_{ji} are equal to 1 if the i-th accepted request of the j-th shipping class is loaded, and 0 otherwise. Objective function (3.18) minimizes the off-loading cost of shipments. Constraints (3.19) and (3.20) ensure that the weight and volume of the loaded items respect the capacity of the flight.

In order to obtain solutions to $z^*(\mathbf{s})$, the MILP is tractable enough to solve exactly for a large number of instances. In this thesis, SCIP [19] is used to obtain MILP solutions.

3.3.2 Prediction Task

The objective is to produce an accurate approximation of (3.17) that is fast to compute. For this purpose, ML is used to predict $z^*(\mathbf{s})$.

The features are computed from the state at the end of the time horizon, $g(\cdot)$. Similar to the application in distributional logistics, a fixed-size input structure for the ML model is used. The number of items can vary depending on the set of accepted requests. Therefore, features are derived from the capacities, number of items, and aggregate statistics of the items, namely, the weights, volumes, and off-loading costs are used. For each of these, the min, max, mean, median, standard deviation, and 1st/3rd quartiles are computed.

CHAPTER 4 EXPERIMENTS

The objective of this chapter is to compare the profit obtained by the proposed methodology presented in Chapter 3 with existing approaches from the literature. To achieve this, instances of the applications in distributional logistics and airline cargo management are introduced. The baselines for each problem are also implemented and evaluated against various RL and DP approaches which leverage the supervised learning predictions. This section presents the results for both the offline supervised learning and control for the applications described in Chapter 3.

4.1 Application I - Distributional Logistics

This section details the instances, supervised learning results, baselines, control algorithms, and control results within the context of the distributional logistics problem.

4.1.1 Sets of Instances

For this application, 4 sets of instances with 4, 10, 15, and 50 locations are generated. The locations were determined uniformly at random and are split into groups, where the revenue in each group differs. The request probabilities are defined such that locations with a higher revenue have a greater probability of occurring later in the booking period. For a detailed description of the parameters that characterize each set of instances, see Appendix A.1.

4.1.2 Supervised Learning Results

One data set is generated for each of the four sets of instances (4, 10, 15 and 50 locations) using the algorithm described in Section 3.1.2, with $p \in \{0.10, 0.25, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 0.99, 1.0\}$. The data is partitioned into training/validation $\mathcal{D}^{\text{TV}}$ and test $\mathcal{D}^{\text{T}}$ sets for each set of instances. The size of all the datasets can be found in Table 4.2.

To develop a supervised learning model, linear regression, support-vector machine (SVM) [50], and random forests were compared. The hyperparmeters for each of these models were tuned using cross validation and the results on the test set for each set of instances is reported in Table 4.1. All models were implemented with Scikit-learn [51]. Note that feature and label normalization was originally used to ensure the training of the models was not impacted by the magnitude of the features and labels, however tree-based models were chosen so normalization is not needed. The prediction performance is evaluated using two performance metrics: MSE and MAE, which are computed with respect to the sum of the prediction and outsourcing cost obtained through bin packing. Unfortunately, the random policy which generates data can have booking periods with no accepted requests, resulting in a zero valued target, so percentage-based evaluation metrics are not directly applicable.

The results reported in Table 4.1 show that the MAE and MSE obtained by random forests are lower on both the train and test data for all sets of instances. Generally, the random forest models tend to overfit the training data more than SVM or linear regression, however, despite this, the test MAE and MSE are still significantly lower. For this reason, random forest models were used for the end-of-horizon predictions within the control tasks.

Overall, with respect to random forests, relatively good performance is achieved but it deteriorates with the size of the instances. In particular, the MSE is quite large, but this is numerically expected due to the squaring of values which are simply large in magnitude. Furthermore, since the magnitude of the labels increases with the number of locations, an increase in both MSE and MAE is expected. In addition, the mean error is low in comparison to the magnitude of the target, so it may not have a significant impact when used within control algorithms, however, this claim has yet to be empirically validated within the context of this thesis. The prediction task here is relatively simple as it only requires predicting a single value as opposed to the routes themselves, thus it can be easily modelled through a set of linear features. That being said, the use of more flexible models such as graph neural networks [52] and pointer networks [53] is a direction for future research as they have been used for a variety of routing problems to determine solutions [54] and even routing costs [46] for TSP. Table 4.2 reports the sizes of the datasets, the time required to generate data, and the time to train random forest models.

4.1.3 Baselines

Three benchmarks are used to evaluate the performance of the approach presented in this thesis. The booking limit policy (BLP) and booking limit policy with reoptimization (BLPR) as described in [5] and implemented using SCIP [19] as MILP solver. The BLP formulation in [5] is given by the inclusion of the routing variables and constraints into a standard DLP, resulting in a MILP. The booking limit policy is then obtained by not accepting more requests than the values given in the primal solution of the MILP. The BLPR simply re-solves the MILP halfway through the booking period, resulting in updated booking limits based on the remaining capacity. In [5], the MILP solving is terminated once 2 incumbent solutions are found, which is done since solving the MILP to optimality is intractable. For the sets of

Table 4.1 Distributional Logistics: Supervised Learning Results.

Locations	Model	Training MSE	Test MSE	Training MAE	Test MAE
4	Random Forest	**1.82**	**5.30**	**0.70**	**1.24**
	SVM	5.42	6.09	1.38	1.51
	Linear Regression	5.31	5.33	1.74	1.75
10	Random Forest	**3.12**	**13.80**	**1.26**	**2.79**
	SVM	18.66	17.27	3.27	3.17
	Linear Regression	17.80	17.18	3.30	3.23
15	Random Forest	**1.45**	**11.17**	**0.90**	**2.53**
	SVM	13.70	15.34	2.84	2.99
	Linear Regression	12.06	15.33	2.74	3.06
50	Random Forest	**23.22**	**139.41**	**3.68**	**9.24**
	SVM	801.90	868.35	22.42	22.98
	Linear Regression	211.98	201.43	11.59	11.14

Table 4.2 Distributional Logistics: Supervised learning times (seconds).

Locations	$\|D^{TV}\|$	$\|D^{T}\|$	Data Generation Time	Training Time
4	1000	250	285.77	0.19
10	2000	500	1577.34	0.76
15	2000	500	2698.57	0.90
50	2000	500	2379.68	1.88

instances with more than 10 locations, the implementation in SCIP did not find incumbent solutions within a reasonable time, so the BLP and BLPR baselines were omitted for the 15 and 50 location instances. This limitation likely occurs since the MILP is implemented directly as described in [5], without any row generation procedure to efficiently handle the exponential amount of constraints. Note that in [5], control was derived for instances with up to 50 locations, however, the procedure in which they do so is not discussed.

As a third baseline the stationary random policy (rand-p) with an acceptance probability p giving the highest mean profit is used, where p is from the same values as used in data generation.

It is possible to solve the 4 location instances with backward induction to obtain an exact value function. So, the exact value function is included in the set of baselines for this instance set, and, more precisely the exact value function can be computed with the costs given by FILO (DP-Exact) and with predicted costs (DP-ML). Both DP-Exact and DP-ML are included as baselines.

4.1.4 Control Algorithms

To obtain control, SARSA and MCTS as described in Section 2.1 are used. For SARSA, a neural network is used to estimate the action-value function. Two different variants of the MCTS algorithm are used and distinguished by their base policy: one uses a simple random policy (MCTS-rand-X) and the other SARSA (MCTS-SARSA-X). Here, "X" denotes the number of iterations each algorithm uses for simulation and in the experiments, 30 and 100 simulations per stage are used. Often, MCTS with a simple random base policy is quite good and, in some cases, difficult to improve upon with a more robust base policy, thus both are included. Each of the above algorithms uses the approximation of (3.4) when computing a policy (i.e., the sum of the predicted operational cost and the outsourcing cost by bin packing computed with MTP). However, FILO is used in the last step to compute the final operational cost.

4.1.5 Reinforcement Learning Training

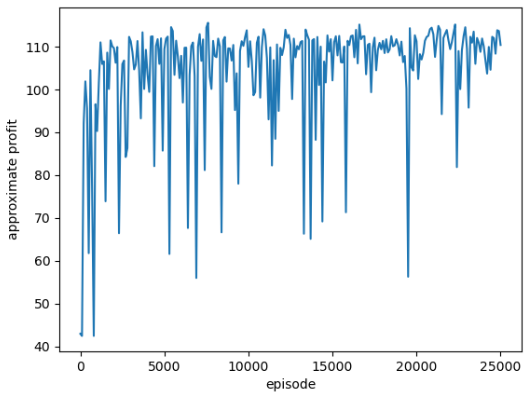
For all SARSA implementations, the probability of taking a random action is $\epsilon = 0.10$. The neural network that approximates the state-action value function was implemented in Pytorch [55] with Adam [56] as optimizer. The state is given by the concatenation of the number of requests accepted for each location, the current request, and the period. Each set of instances uses a neural network with one hidden layer and learning rate that varies depending on the problem setting. The parameters for each instance set are reported in Table 4.3 and were selected by choosing the highest profit over a set of validation instances. In each setting, SARSA is trained for 25,000 iterations and the mean profit over 50 validation instances is computed every 100 episodes. In the remainder of the experiments, the SARSA model that obtains the highest mean profit over the validation instances is used. Figure 4.1 shows the mean performance in terms of approximate profit on the 50 validation instances. Overall, profit generally tends to increase and saturate as the number of episodes increase, however, there is still quite a bit of variance in all instances and the profit of the 50 location setting does not converge. It is common to average these curves over multiple runs, but due to the time of training a single model, this was limited to a single training run.

4.1.6 Control Results

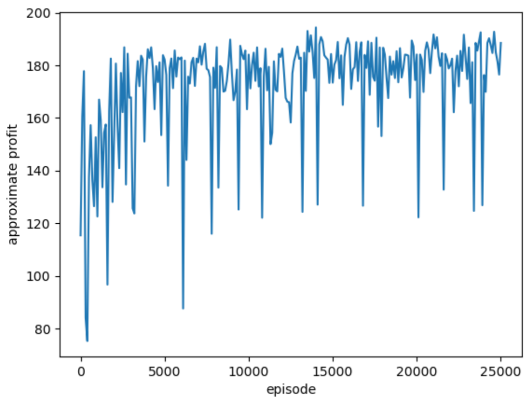
Tables 4.4 reports the mean profit over 50 test instances obtained by each approach as well as the offline and online computing times. The online computing time refers to the time required when evaluating the policy on the test instances, while the training offline computing time

Table 4.3 Distributional Logistics: SARSA neural network hyperparameters.

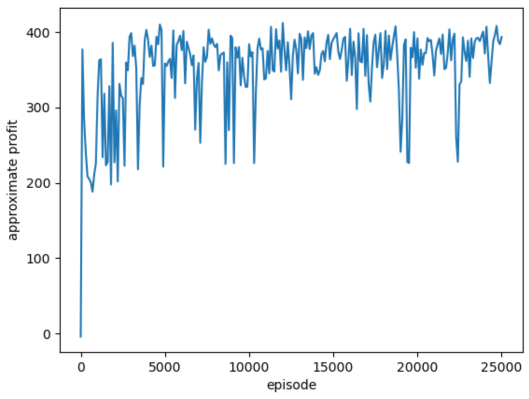
Locations	Hidden Layer Dimension	Learning Rate
4	128	$1e^{-3}$
10	256	$1e^{-3}$
15	256	$1e^{-3}$
50	1024	$1e^{-5}$



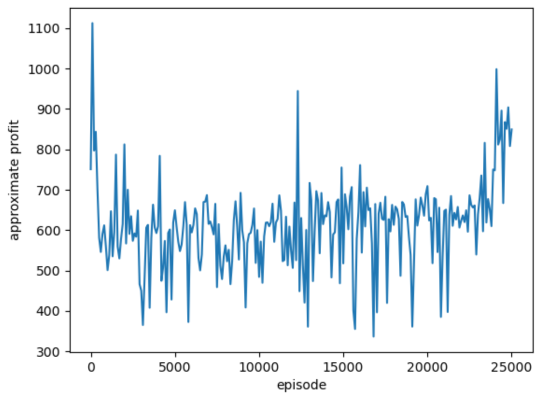
(a) 4 Locations



(b) 10 Locations



(c) 15 Locations



(d) 50 Locations

Figure 4.1 Distributional Logistics: SARSA training curves for 1 run.

includes any time required to compute an approximate action-value function, a booking limit, or bid-price policy. Generally, in capacity control, decisions should be made immediately when the request arrives, thus a short online computing time is ideal. Offline computing is not as crucial, since there should be sufficient time to compute policies beforehand, unless they are prohibitively expensive. In the control algorithms presented, DP-Exact, DP-ML, SARSA and MCTS-SARSA-X compute an exact or approximate value function offline. Similarly, the initial booking limit policy (BLP and BLRP) is computed offline, while any reoptimization of the booking limits and the solution of the VRP at the end of the time horizon contribute to the online computing time. For MCTS-rand-X and rand-p, the policies are computed entirely online. The time associated with generating data and training the ML models adds to the offline computing time of the corresponding algorithms.

In Figure 4.2, box plots are provided to show the distribution of the percentage gaps to the best known solution (i.e., the highest profit solution for each instance) for each algorithm. Figure 4.2a shows that, as expected, DP-Exact has the lowest gaps with a median at zero. The same algorithm but with approximate operational costs (DP-ML) results in gaps close to those of DP-Exact, demonstrating the effectiveness of predicting the operational cost. Comparing the approaches from this thesis to the baselines BLP and BLPR in Figures 4.2a and 4.2b, one can see that the policies obtained via any of the proposed control algorithms achieve smaller gaps than those of BLP, BLPR, and rand-p baselines. Moreover, the BLP and BLPR policies report larger gaps than those of rand-p for 10 locations. For all sets of instances, the MCTS algorithms consistently achieve the smallest gaps, with some variations depending on the number of simulations and the base policy. As expected, a larger number of simulations leads to smaller variance.

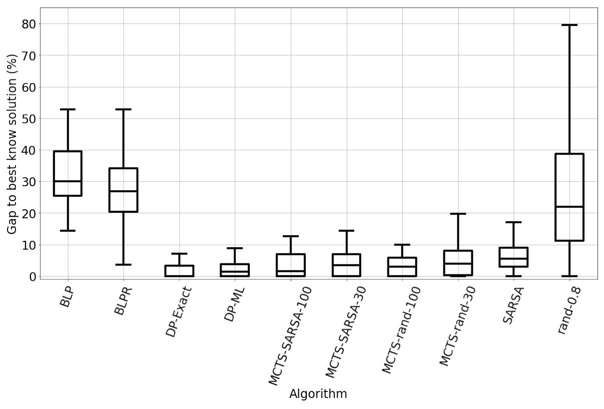
The online computing times are the shortest for SARSA and rand-p with an average of less than 2 seconds for all sets of instances. On the 10-location instances, the bid-price policies (BLP and BLPR) have an average online computing time comparable to MCTS-rand-30 and MCTS-SARSA-30. The computing time of the MCTS algorithms depends on the instance size. On average, they approximately double for 15-locations compared to 10-locations and the 50-location instances are on average 8 times more time consuming to solve than the 10-location ones in the case of MCTS-rand-X (11 times in the case of MCTS-SARSA-X). This leads to average online computing times between 1.2 (MTCS-rand-30) to 7.5 minutes (MCTS-SARSA-100) for the largest instances. It should be noted that the latter is the algorithm achieving the highest quality solutions. However, using 100 simulations instead of 30 increases the online computing time by approximately a factor of 4 for all sets of instances.

The trade-off between solution quality and online computing time becomes clearly visible for the larger instances (15 and 50 locations). As highlighted in Figure 4.2, this trade-off can be partly controlled by the simulation budget. Noteworthy is the performance of SARSA on the largest instances: On average, the gap to the best known solution is 4.7% and the online computing time is only 1.31 seconds.

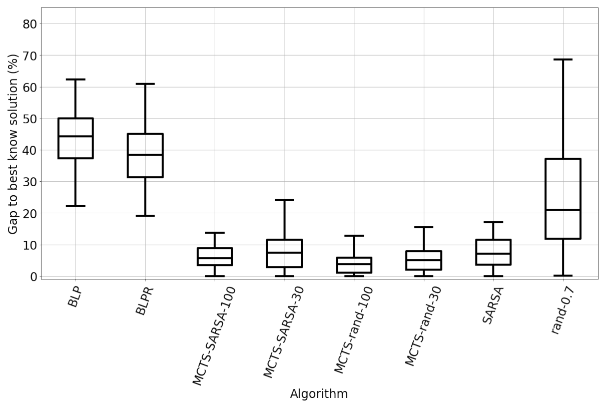
Finally, it is important to comment on the number of evaluations of the operational costs. When comparing DP-Exact and DP-ML, the former calls FILO 10,000 times to compute the value functions (offline), while DP-ML only solves 1,000 VRPs when generating data for supervised learning. In the case of SARSA, the use of supervised learning becomes more important as training SARSA requires 25,000 episodes of simulation, in turn requiring the objective values of 25,000 VRPs. With the proposed methodology, only 1,000-2,000 VRPs need to be solved in order to train the model, while if using FILO directly, 25,000 VRPs would need to be solved. The MCTS-based algorithms require a number of cost evaluations (online) proportional to the number of simulations times T.

Table 4.4 Distributional Logistics: Profits and computing times (seconds).

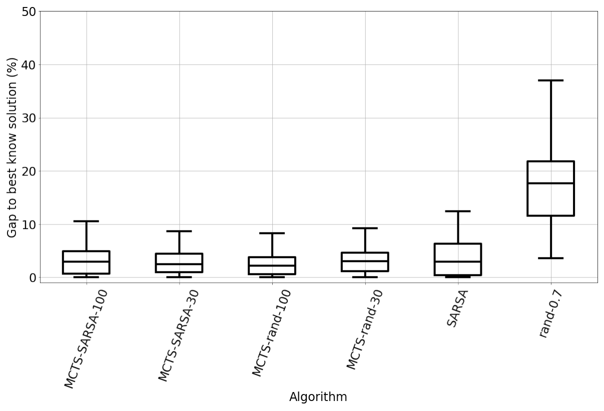
Locations	Control Algorithm	Mean Profit	Online Time	Offline Time
4	DP-Exact	**113.88**	0.28	2874.83
	DP-ML	113.13	0.30	29.23
	SARSA	109.33	0.27	558.18
	MCTS-rand-30	109.58	5.47	-
	MCTS-rand-100	112.23	22.01	-
	MCTS-SARSA–30	111.40	6.35	558.18
	MCTS-SARSA-100	111.03	24.46	558.18
	BLP	78.35	**0.02**	0.01
	BLPR	82.81	0.03	0.01
	rand-0.6	71.49	0.32	-
10	SARSA	189.25	**0.91**	1040.38
	MCTS-rand-30	199.45	9.06	-
	MCTS-rand-100	**201.95**	36.04	-
	MCTS-SARSA–30	194.54	11.81	1040.38
	MCTS-SARSA-100	197.11	43.77	1040.38
	BLP	118.94	10.93	2.84
	BLPR	130.01	12.63	2.84
	rand-0.7	158.64	0.95	-
15	SARSA	400.43	1.72	1640.48
	MCTS-rand-30	422.19	18.30	-
	MCTS-rand-100	**425.14**	74.87	-
	MCTS-SARSA–30	423.38	25.32	1640.48
	MCTS-SARSA-100	421.59	97.51	1640.48
	rand-0.7	357.95	**1.55**	-
50	SARSA	1098.78	1.31	7045.92
	MCTS-rand-30	1095.02	74.92	-
	MCTS-rand-100	1118.60	296.10	-
	MCTS-SARSA–30	1112.45	123.47	7045.92
	MCTS-SARSA-100	**1153.71**	450.05	7045.92
	rand-0.7	862.52	**1.04**	-



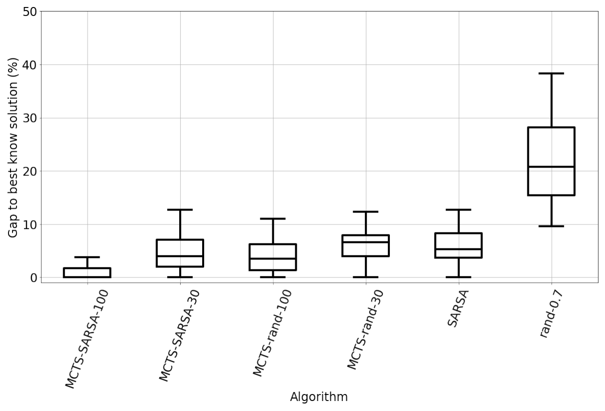
(a) 4 Locations



(b) 10 Locations



(c) 15 Locations



(d) 50 Locations

Figure 4.2 Distributional Logistics: Optimality gaps to best known solutions.

4.2 Application II - Airline Cargo

This section details the instances, supervised learning results, baselines, control algorithms, and control results within the context of the airline cargo management problem.

4.2.1 Instance

For this application, the single-leg setting with the lowest cargo capacity from Barz et al. [33] is considered. There are 24 shipment classes and 60 time periods. Similar to the application in distributional logistics, requests that offer a higher revenue per unit of chargeable weight have an increased probability of occurring later in the booking period. For a detailed description of the parameters that describe each set of instances, see Appendix A.2.

4.2.2 Supervised Learning Results

One data set is generated for the single-leg instance using the algorithm described in Section 3.1.2, with $p \in \{0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.0\}$. The data is partitioned into training/validation $\mathcal{D}^{\mathrm{TV}}$ and test $\mathcal{D}^{\mathrm{T}}$ sets, where $|D^{TV}| = 5000$ and $|D^{T}| = 1000$.

Similar to the application in distributional logistics, random forests, SVM, and linear regression are compared using MAE and MSE in Table 4.5. All models were implemented in Scikit-learn [51] and hyperparameter selection was done using cross validation. From Table 4.5, random forests obtain significantly lower MAE and MSE on the test set, despite having a larger generalization gap between the train and test set. In this application the magnitude of difference in MAE and MSE is much more significant, with random forests achieving an MAE almost 6.5x lower than the other models. For this reason, random forest models were used for the end-of-horizon predictions within the control tasks. From the table, a relatively high MSE value can be observed, but when compared to the magnitude of labels, which has a mean value of 6020.88, this result is somewhat expected. The times to generate the training data and train random forest models were 6655.88 and 5.60 seconds, respectively.

Table 4.5 Airline Cargo: Supervised Learning Results.

Model	Training MSE	Test MSE	Training MAE	Test MAE
Random Forest	59142.00	372627.85	117.78	306.08
SVM	5366208.18	5294121.73	1956.25	1961.14
Linear Regression	6261576.66	6213429.80	2018.58	2021.19

4.2.3 Baselines

Three benchmarks are used to evaluate the performance of the approach presented in this thesis: First come first serve (FCFS), deterministic linear programming (DLP), and dynamic programming decomposition (DPD) as described in [33]. More precisely, FCFS simply accepts requests until the expected weight or volume of the accepted requests exceeds the capacity. In the DLP formulation, the end-of-horizon problem is handled by relaxing the integrality constraints and allowing for partial loading of items. Bid-price controls are then obtained using the optimal dual solution. Finally, DPD uses the dual solution obtained by DLP to formulate two separate DP problems where the state is defined by the expected remaining weight and volume, respectively. The resulting DPs can then be solved through backtracking and combined to obtain control. For each of these algorithms, the off-loading cost is computed as a solution of the MILP (3.18)-(3.21) with SCIP [19].

4.2.4 Control Algorithms

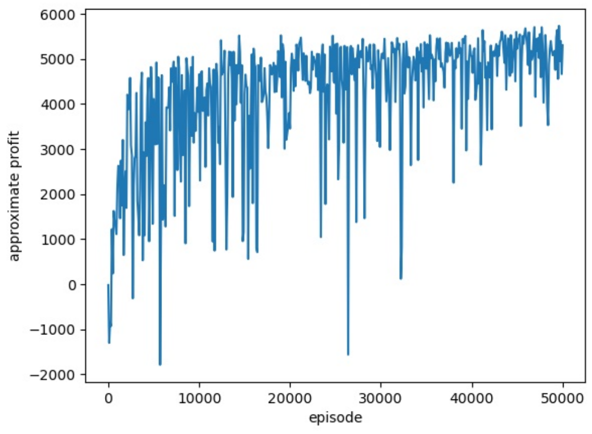
The baselines for this problem are significantly stronger than those used in the distributional logistics application, since they compute policies offline, which can then be applied to obtain real-time control. For this reason, the control algorithms considered are restricted to those which can be evaluated efficiently, which includes SARSA with a neural network to estimate the action-value function (SARSA). To provide more robust control policies, an ensemble of trained SARSA models is used, resulting in the policy (SARSA-Y), where "Y" denotes the number of ensembled models. Note that SARSA uses the approximation of (3.17) when computing a policy, however, SCIP is used in the last step to compute the actual off-loading cost. When training SARSA, the realization of weights and volumes can greatly impact the cost in the final time steps, so in order to reduce the variance, 50 realizations are sampled and the mean of their predicted off-loading costs is used when updating the parameters of the neural network.

4.2.5 Reinforcement Learning Training

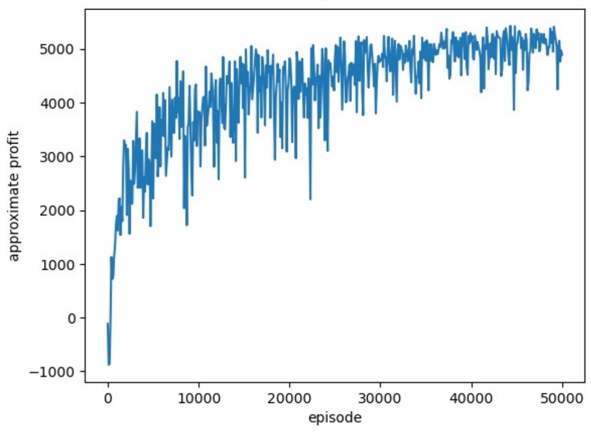
In this application, the probability of taking a random action is $\epsilon = 0.10$. The state is given by the concatenation of the number of requests for each class, the class of the current request, and the current period. The neural network which approximates the state-action value function is a two-layer network with a hidden layer dimension of 1024 and was trained using a learning rate of $1e^{-3}$. The hyperparmaters were selected by varying the hidden dimension and learning rate, then choosing the model which achieved the highest profit over a set of validation instances. Similarly to the previous application, the neural network is implemented in PyTorch and optimized using the Adam optimizer. Moreover, SARSA is trained for 50,000 iterations and evaluated every 100 episodes over a set of 100 validation instances. In the remainder of the experiments, the SARSA model which obtains the highest mean profit over the validation instances is used. Figure 4.1 presents the mean profit obtained over the validation instances. Figure 4.3a is the training performance of a single SARSA model, while Figure 4.3b shows the mean profit over the models used in the ensemble implementation.

4.2.6 Control Results

Tables 4.6 reports the mean profit obtained by each approach as well as the offline and online computing times. In this case, the online computing time is relatively small and comparable between all of the approaches, while the offline computing time is significantly higher for SARSA. The online computing time in this case mostly depends on the time required to



(a) SARSA (1 run)



(b) SARSA-5 (5 run mean)

Figure 4.3 Airline Cargo: SARSA training curves.

solve the end-of-horizon problem, so algorithms which tend to cause the least overbooking will have a lower computing time since solving the vector packing problem is trivial when all items can be packed. In terms of profit, the DLP baseline achieves the highest mean profit, closely followed by the DPD baseline. Unfortunately, the policies obtained through control algorithms presented in this thesis do not improve on the baselines. Improving on the baselines for airline cargo is difficult: on the one side, the area has been explored significantly more, so the available policies are more competitive. On the other hand, the end-of-horizon problem is similar and the advantage of the ML approximation is reduced. Although not quite as good, the performance obtained using supervised learning and off-the-shelf RL algorithms is still quite close to the problem specific algorithms from Barz et al. [33], which highlights a strong potential for this approach. Furthermore, more experimentation and the use of more robustly implemented RL algorithms provide two potential directions to improve upon the results presented in this thesis.

In Figure 4.4, box plots are provided to show the distribution of the percentage gaps to the best known solution (i.e., the highest profit solution for each instance) for each algorithm. Overall, the results are inline with those presented in Table 4.6, with DLP and DPD generally having the smallest gaps, followed by SARSA-5 and SARSA, and FCFS performing significantly worse. Table 4.7 shows the number of times each algorithm achieved the highest profit solution on an instance. In this case, DPD obtained the highest profit on a significantly larger number of instances, which interestingly was not reflected in the mean profit. The number of best solutions obtained by DLP, SARSA, and SARSA-5 are relatively similar, while FCFS unsurprisingly finds the least.

In order to better understand the difference between the performance of the control algorithms, a more detailed comparison of the best performing algorithm presented in this work (SARSA-5) and the work by [33] (DLP) is provided. Figure 4.5 presents a histogram of the differences in both profit and online compute time of the two algorithms. The x-axis represents the profit or time difference of DLP and SARSA-5, where values above 0 indicate DLP achieving a higher profit or lower time on an instance, while values below 0 indicate the opposite. The y-axis represents the number of instances. From Figure 4.5a, one can see a skew to the right of 0, indicating that DLP generally has slightly higher profit solutions, rather than the difference in mean performance being from outliers. Figure 4.5b presents the difference in times. In Figure 4.5b the outliers are removed to improve readability, but in general there is a very small difference between the online computing time of the two algorithms as most values are close to zero.

Table 4.6 Airline Cargo: Profits and computing times (seconds).

Control Algorithm	Mean Profit	Online Time	Offline Time
SARSA	5601.52	0.83	8169.10
SARSA-5	5813.49	0.30	34070.88
FCFS	3648.64	0.74	-
DLP	**6110.94**	**0.20**	0.007
DPD	6104.52	0.32	82.65

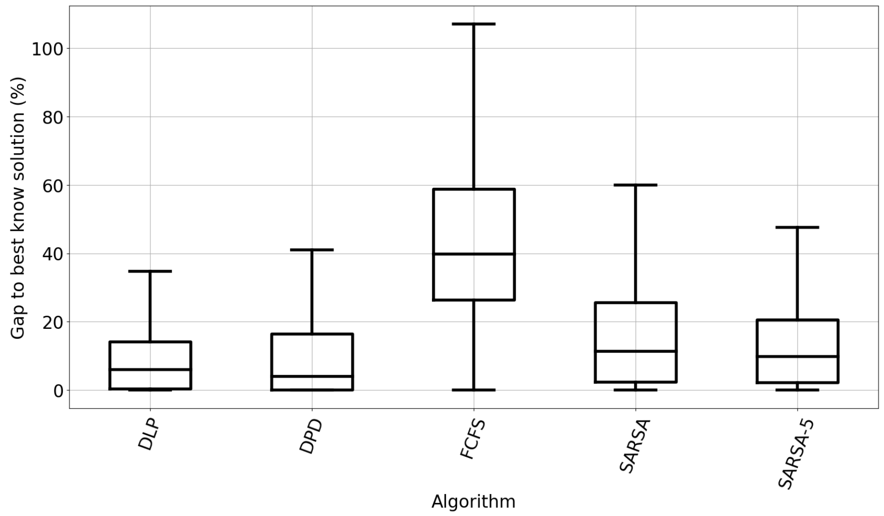
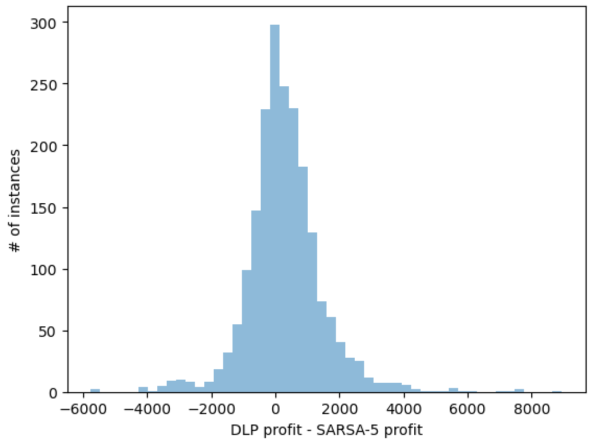


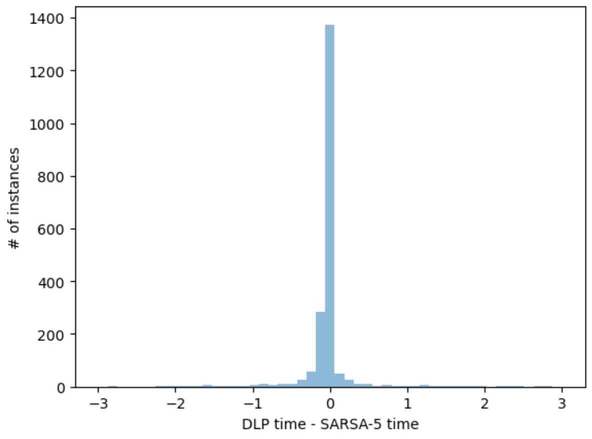
Figure 4.4 Airline Cargo: Optimality gaps to best known solutions.

Table 4.7 Airline Cargo: Number of best solutions (out of 2000 instances).

Control Algorithm	# of Best Solutions
SARSA	387
SARSA-5	376
FCFS	83
DLP	478
DPD	683



(a) DLP and SARSA-5 profit difference



(b) DLP and SARSA-5 time difference

Figure 4.5 Airline Cargo: Differences between DLP and SARSA-5.

CHAPTER 5 CONCLUSION AND RECOMMENDATIONS

In the context of complex booking control problems, this thesis presents a novel, general, and scalable approach for determining control by combining supervised learning with DP or RL algorithms. Specifically, supervised learning is used to predict the objective function value of operational decision-making problems formulated as MILPs. These predictions are then leveraged to create an approximation of the original booking control problem with significantly more efficient simulation. This in turn provides the potential direct use of a wide variety of state-of-the-art RL and DP algorithms, rather than relying on problem specific solutions, which have been the predominant approach in booking control problems. The performance of this approach is evaluated in a distributional logistics and airline cargo management settings, with off-the-shelf supervised learning and DP/RL algorithms. Despite the use of simple off-the-shelf algorithms, the policies obtained greatly improve on the baselines in the distributional logistics application, while performing only slightly worse than the stronger baselines in airline cargo management application. This approach can be extended to account for any end-of-horizon problem within booking control as long as it can be formulated as a prediction task, and may even be useful within other sequential decision-making problems which have environments where non-trivial computations limit simulation-based approaches.

Overall, this work presents a general framework, that can be improved upon in various ways. Two of the most obvious directions for future work are in the improvement of both the supervised learning and control algorithms. Specifically, the use of more elaborate supervised learning algorithms that leverage the variable input size may provide improvements in prediction quality, which will ultimately result in more accurate approximate MDP formulations. In the context of this thesis, SARSA was chosen as it is a simple algorithm that was still able to demonstrate that the methodology proposed is effective in booking control problems. This highlights one direction for future work, which is to replace SARSA with state-of-the-art RL algorithms such as policy gradients and deep Q-networks that have been robustly implemented in libraries such as Stable Baselines [57] or TF-Agents [58]. Another direction for potential future work is with respect to learning the policy and the end-of-horizon problem jointly with the state-action value function. This could be done entirely through RL, which may be intractable for end-of-horizon problems such as the one in the distributional logistics application, or perhaps through the use of a pretrained predictor to the operational problem, which is periodically updated when training an RL model.

REFERENCES

[1] P. Bowes, "Parcel shipping index," 2019.

[2] P. P. Belobaba, "Survey paper—airline yield management an overview of seat inventory control," *Transportation science*, vol. 21, no. 2, pp. 63–73, 1987.

[3] R. Klein, S. Koch, C. Steinhardt, and A. K. Strauss, "A review of revenue management: Recent generalizations and advances in industry applications," *European Journal of Operational Research*, vol. 284, no. 2, pp. 397–412, 2020.

[4] R. G. Kasilingam, "Air cargo revenue management: Characteristics and complexities," *European Journal of Operational Research*, vol. 96, no. 1, pp. 36–44, 1997.

[5] G. Giallombardo, F. Guerriero, and G. Miglionico, "Profit maximization via capacity control for distribution logistics problems," *arXiv preprint arXiv:2008.03216*, 2020.

[6] K. T. Talluri and G. J. V. Ryzin, *The Theory and Practice of Revenue Management.* Springer US, 2004.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018.

[8] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[9] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games.* Springer, 2006, pp. 72–83.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[11] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2006, pp. 282–293.

[12] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems.* University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[13] D. P. Bertsekas, *Reinforcement learning and optimal control.* Athena Scientific Belmont, MA, 2019.

[14] G. B. Dantzig, "Origins of the simplex method," in *A history of scientific computing*, 1990, pp. 141–151.

[15] A. Schrijver, *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[16] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations.* Springer, 1972, pp. 85–103.

[17] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[18] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[19] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 7.0," Optimization Online, Technical Report, March 2020. [Online]. Available: http://www.optimization-online.org/DB_HTML/2020/03/7705.html

[20] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[21] A. Lodi, "Mixed integer programming computation," in *50 years of integer programming 1958-2008.* Springer, 2010, pp. 619–645.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] J. Friedman, T. Hastie, R. Tibshirani *et al.*, *The elements of statistical learning.* Springer series in statistics New York, 2001, vol. 1, no. 10.

[25] S. Arlot and A. Celisse, “A survey of cross-validation procedures for model selection,” *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[26] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees.* Routledge, 2017.

[27] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] J. R. Quinlan, “Induction of decision trees,” *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[29] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[33] C. Barz and D. Gartner, “Air cargo network revenue management,” *Transportation Science*, vol. 50, no. 4, pp. 1206–1222, 2016.

[34] E. L. Williamson, “Airline network seat inventory control: Methodologies and revenue impacts,” Ph.D. dissertation, Massachusetts Institute of Technology, 1992.

[35] D. Adelman, “Dynamic bid prices in revenue management,” *Operations Research*, vol. 55, no. 4, pp. 647–661, 2007.

[36] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

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

[38] A. Gosavi, 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.

[39] R. J. Lawhead and A. Gosavi, “A bounded actor–critic reinforcement learning algorithm applied to airline revenue management,” *Engineering Applications of Artificial Intelligence*, vol. 82, pp. 252 – 262, 2019.

[40] N. Bondoux, A. Q. Nguyen, T. Fiig, and R. Acuna-Agost, “Reinforcement learning applied to airline revenue management,” *Journal of Revenue and Pricing Management*, vol. 19, no. 5, pp. 332–348, 2020.

[41] K. Amaruchkul, W. L. Cooper, and D. Gupta, “Single-leg air-cargo revenue management,” *Transportation science*, vol. 41, no. 4, pp. 457–469, 2007.

[42] Y. Levin, M. Nediak, and H. Topaloglu, “Cargo capacity management with allotments and spot market demand,” *Operations Research*, vol. 60, no. 2, pp. 351–365, 2012.

[43] T. Levina, Y. Levin, J. McGill, and M. Nediak, “Network cargo capacity management,” *Operations Research*, vol. 59, no. 4, pp. 1008–1023, 2011.

[44] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: a methodological tour d’horizon,” *European Journal of Operational Research*, 2020.

[45] M. Fischetti and M. Fraccaro, “Using or+ ai to predict the optimal production of offshore wind parks: a preliminary study,” in *International Conference on Optimization and Decision Science.* Springer, 2017, pp. 203–211.

[46] M. Prates, P. H. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi, “Learning to solve np-complete problems: A graph neural network for decision tsp,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4731–4738.

[47] L. Accorsi and D. Vigo, “A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems,” University of Bologna, Tech. Rep., 2020.

[48] S. Martello and P. Toth, “Knapsack problems: algorithms and computer implementations,” *Wiley-Interscience series in discrete mathematics and optimization*, 1990.

[49] A. Caprara and P. Toth, “Lower bounds and algorithms for the 2-dimensional vector packing problem,” *Discrete Applied Mathematics*, vol. 111, no. 3, pp. 231–262, 2001.

[50] I. Steinwart and A. Christmann, *Support vector machines.* Springer Science & Business Media, 2008.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[52] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[53] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” *Advances in Neural Information Processing Systems*, vol. 28, pp. 2692–2700, 2015.

[54] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems!” in *International Conference on Learning Representations*, 2018.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: an imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019, pp. 8026–8037.

[56] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[57] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines,” https://github.com/hill-a/stable-baselines, 2018.

[58] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, “TF-Agents: A library for reinforcement learning in tensorflow,” https://github.com/tensorflow/agents, 2018, [Online; accessed 25-June-2019]. [Online]. Available: https://github.com/tensorflow/agents

APPENDIX A SETS OF INSTANCES

A.1 Distributional Logistics Instances

In this section, the parameters that define the 4 sets of instances in the first application are detailed. In each of the instances the capacity is determined to be proportional to the inverse demand of the locations. Specifically, a load factor, LF, is used, and the capacity is given by

$$Q = \left\lfloor \frac{\sum_{j \in \mathcal{N} \cup \{0\}} \sum_{t=1}^{T} \lambda_j^t}{K_0 \cdot LF} \right\rfloor. \tag{A.1}$$

The remainder of this section details parameters to generate each set of instances for the distributional logistics applications.

A.1.1 4 Locations

The locations given by $\mathcal{N} = \{1, \dots, 4\}$, the number of periods $T = 20$, the revenue for accepting a request from each location are defined by $p_1 = 4$, $p_2 = 8$, $p_3 = 12$, $p_4 = 16$. The probability of no request is $\lambda_0^t = 0.10$, for $t \in \{1, \dots, T\}$. The initial request probabilities for each location are $\lambda_1^1 = 0.45$, $\lambda_2^1 = 0.40$, $\lambda_3^1 = 0.10$, $\lambda_4^1 = 0.05$. The remainder of the request probabilities are then given by $\lambda_j^{t+1} = \lambda_j^t - 0.01$ for $j \in \{1, 2\}$ and $\lambda_j^{t+1} = \lambda_j^t + 0.01$ for $j \in \{3, 4\}$. The coordinates for each location are sampled uniformly at random in the interval $[0, 10]$. The number of vehicles available at no cost is $K_0 = 2$ and the cost for each additional over K_0 is $C = 100$. The capacity is determine using a load factor $LF = 1.1$.

A.1.2 10 Locations

The locations given by $\mathcal{N} = \{1, \dots, 10\}$, the number of periods $T = 30$, the revenue for accepting a request from each location are defined by $p_j = 10$ for $j \in \{1, \dots, 4\}$, $p_j = 12$ for $j \in \{5, \dots, 8\}$, and $p_j = 20$ for $j \in \{9, 10\}$. The probability of no request is $\lambda_0^t = 0.10$, for $t \in \{1, \dots, T\}$. The initial request probabilities for each location are $\lambda_j^1 = 0.125$ for $j \in \{1, \dots 4\}$, $\lambda_j^1 = 0.075$ for $j \in \{5, \dots, 8\}$, and $\lambda_j^1 = 0.05$ for $j \in \{9, 10\}$. The remainder of the request probabilities are then given by $\lambda_j^{t+1} = \lambda_j^t - 0.001$ for $j \in \{1, \dots 4\}$, $\lambda_j^{t+1} = \lambda_j^t$ for $j \in \{5, \dots, 8\}$, and $\lambda_j^{t+1} = \lambda_j^t + 0.002$ for $j \in \{9, 10\}$. The coordinates for each location are sampled uniformly at random in the interval $[0, 10]$. The number of vehicles available at no cost is $K_0 = 4$ and the cost for each additional over K_0 is $C = 100$. The capacity is determine using a load factor $LF = 1.2$.

A.1.3 15 Locations

The locations given by $\mathcal{N} = \{1, \ldots, 15\}$, the number of periods $T = 50$, the revenue for accepting a request from each location are defined by $p_j = 10$ for $j \in \{1, \ldots, 5\}$, $p_j = 12$ for $j \in \{6, \ldots, 10\}$, and $p_j = 20$ for $j \in \{11, \ldots, 15\}$. The probability of no request is $\lambda_0^t = 0.10$, for $t \in \{1, \ldots, T\}$. The initial request probabilities for each location are $\lambda_j^1 = 0.10$ for $j \in \{1, \ldots, 5\}$, $\lambda_j^1 = 0.06$ for $j \in \{6, \ldots, 10\}$, and $\lambda_j^1 = 0.02$ for $j \in \{11, \ldots, 15\}$. The remainder of the request probabilities are then given by $\lambda_j^{t+1} = \lambda_j^t - 0.001$ for $j \in \{1, \ldots, 5\}$, $\lambda_j^{t+1} = \lambda_j^t$ for $j \in \{6, \ldots, 10\}$, and $\lambda_j^{t+1} = \lambda_j^t + 0.001$ for $j \in \{11, \ldots, 15\}$. The coordinates for each location are sampled uniformly at random in the interval $[0, 10]$. The number of vehicles available at no cost is $K_0 = 4$ and the cost for each additional over K_0 is $C = 250$. The capacity is determine using a load factor $LF = 1.2$.

A.1.4 50 Locations

The locations given by $\mathcal{N} = \{1, \ldots, 50\}$, the number of periods $T = 100$, the revenue for accepting a request from each location are defined by $p_j = 15$ for $j \in \{1, \ldots, 30\}$, $p_j = 22$ for $j \in \{31, \ldots, 40\}$, and $p_j = 30$ for $j \in \{41, \ldots, 50\}$. The probability of no request is $\lambda_0^t = 0.10$, for $t \in \{1, \ldots, T\}$. The initial request probabilities for each location are $\lambda_j^1 = 0.0166$ for $j \in \{1, \ldots, 30\}$, $\lambda_j^1 = 0.03$ for $j \in \{31, \ldots, 40\}$, and $\lambda_j^1 = 0.01$ for $j \in \{41, \ldots, 50\}$. The remainder of the request probabilities are then given by $\lambda_j^{t+1} = \lambda_j^t - 0.0001$ for $j \in \{1, \ldots, 30\}$, $\lambda_j^{t+1} = \lambda_j^t$ for $j \in \{31, \ldots, 40\}$, and $\lambda_j^{t+1} = \lambda_j^t + 0.0003$ for $j \in \{41, \ldots, 50\}$. The coordinates for each location are sampled uniformly at random in the interval $[0, 50]$. The number of vehicles available at no cost is $K_0 = 4$ and the cost for each additional over K_0 is $C = 600$. The capacity is determine using a load factor $LF = 1.3$.

A.2 Airline Cargo Instance

In this section, the parameters that define the airline cargo instance in the second application are detailed. In this setting the number of periods is given by $T = 60$. The parameter which determines the chargeable weight is given by $\eta = 0.6$. The number of shipment classes is $n = 24$. As mentioned previously, the realization of the weight and volume are given at the end of the booking period, however, the expectations of each class is known and provided in Table A.1, where $\bar{w}_j$ and $\bar{v}_j$ denote the expected weight and volume of class j. At the end of the booking period, the realized weight and volume of a request of class j are given by sampling a multivariate normal distribution with mean $\mu = [\bar{w}_j, \bar{v}_j]$ and covariance matrix

$$\Sigma = \begin{pmatrix} (\sigma_j^w)^2 & 0.8\sigma_j^w\sigma_j^v \\ 0.8\sigma_j^w\sigma_j^v & (\sigma_j^v)^2 \end{pmatrix}, \tag{A.2}$$

where $\sigma_j^w = 0.25\bar{w}_j$ and $\sigma_j^v = 0.25\bar{v}_j$.

Table A.1 Airline Cargo: Expected weights and volumes.

Class	$\bar{w}_j$ (kg)	$\bar{v}_j$ ($1e^4$ cm^3)
1	50	30
2	50	29
3	50	27
4	50	25
5	100	59
6	100	58
7	100	55
8	100	52
9	200	125
10	200	119
11	250	100
12	250	147
13	300	138
14	400	179
15	500	235
16	500	277
17	1000	598
18	1500	898
19	2500	1488
20	3500	2083
21	70	233
22	70	17
23	210	700
24	210	52

Let w_j and v_j are the realized weight and volume of a request of class j. The off-loading cost of the request is given by $c_j \cdot \max\{w_j, v_j/\eta\}$, where $c_j = 2.4$ for $j = 1, \ldots, 24$. The profit is given by $\alpha \cdot \max\{w_j, v_j/\eta\}$, where $\alpha \in \{0.7, 1.0, 1.4\}$. Here, α is used in order to ensure more profitable requests occur later in the booking period. Table A.2 gives the probability that α takes each value across the time horizon as well as the probability that no request occurs. Given that a request occurs, the probability of the request for each class is provided in Table A.3.

Table A.2 Airline Cargo: Profit per chargeable weight probabiltiies.

Request	Periods 1-20	Periods 21-40	Periods 41-60
$Pr(\alpha = 0.7)$	0.7	0.4	0.0
$Pr(\alpha = 1.0)$	0.2	0.2	0.2
$Pr(\alpha = 1.4)$	0.0	0.4	0.7
P(No request)	0.1	0.0	0.1

Similar to the weight and volume of a request, the capacities are realized at the end of the booking period. The expected weight and volume capacities are given as a factor of the total expected weight and volume of the requests. In the context of this thesis, the most limiting scenario from [33] is considered, where the expected capacity is half of the expected requests for both the weight and volume. The expected weight capacity and volume capacity are denoted by $\bar{Q}_w$ and $\bar{Q}_v$, respectively. At the end of the booking period, the realized weight and volume capacities are given by sampling a multivariate normal distribution with mean $\mu = [\bar{Q}_w, \bar{Q}_v]$ and covariance matrix

$$\Sigma = \begin{pmatrix} (\sigma^{Q_w})^2 & 0.8\sigma^{Q_w}\sigma^{Q_v} \\ 0.8\sigma^{Q_w}\sigma^{Q_v} & (\sigma^{Q_v})^2 \end{pmatrix}, \tag{A.3}$$

where $\sigma^{Q_w} = 0.25\bar{Q}_w$ and $\sigma^{Q_v} = 0.25\bar{Q}_v$.

Table A.3 Airline Cargo: Class request probabilities.

Class	Probability
1	0.072
2	0.072
3	0.072
4	0.072
5	0.072
6	0.072
7	0.072
8	0.072
9	0.072
10	0.072
11	0.040
12	0.040
13	0.040
14	0.040
15	0.040
16	0.040
17	0.009
18	0.009
19	0.009
20	0.009
21	0.001
22	0.001
23	0.001
24	0.001