The Dynamic Freight Routing Problem for Less-than-Truckload Carriers

Ahmad Baubaid[1,2], Natashia Boland[1], and Martin Savelsbergh[1]

[1]H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA
baubaid@gatech.edu, {natashia.boland, martin.savelsbergh}@isye.gatech.edu
[2]Department of Systems Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

November 5, 2020

Abstract

Less-than-Truckload (LTL) carriers transport freight shipments from origins to destinations by consolidating freight using a network of terminals. As daily freight quantities are uncertain, carriers dynamically adjust planned freight routes on the day of operations. We introduce the Dynamic Freight Routing Problem (DFRP) and model this problem as a Markov Decision Process (MDP). To overcome the curses of dimensionality of the MDP model, we introduce an Approximate Dynamic Programming (ADP) solution approach that uses a lookup table to store value function approximations, and introduce and compare a number of aggregation approaches which use features of the post-decision state to aggregate the post-decision state space, thereby reducing the number of entries in the lookup table. Furthermore, since the decision subproblems are integer programs (IPs), we present a framework for integrating lookup tables into the decision subproblem IPs. This framework consists of: (1) a modeling approach for the integration of lookup table value function approximations into subproblem IPs to form extended subproblem IPs, (2) a solution approach, PDS-IP-Bounding, which decomposes the extended subproblem IPs into many smaller IPs and uses dynamic bounds to reduce the number of small IPs that have to be solved, and (3) an adaptation of the ϵ-greedy exploration-exploitation algorithm for the IP setting. Our computational experiments show that despite the post-decision state of the DFRP being high-dimensional, a two-dimensional aggregation of the post-decision space is able to produce policies that outperform standard myopic policies. Moreover, our experiments demonstrate that the PDS-IP-Bounding algorithm provides computational advantages over solving the extended subproblem IPs using a commercial solver.

1 Introduction

Less-than-truckload (LTL) carriers are freight consolidation carriers that collect and transport freight shipments from origins to destinations. Since individual shipments are typically small in size relative to the size of trailers, it is not economically feasible to transport them directly from origins to destinations, and therefore, consolidation of shipments from multiple shippers is necessary for a profitable operation. To facilitate consolidation, LTL carriers operate a hub-and-spoke network of terminals, referred to as the *linehaul* network. The spokes in this network are called *end-of-line*

(EOL) terminals and collect freight from shippers located in their area of service. The hubs are called *breakbulk* (BB) terminals and are larger terminals which, in addition to collecting freight, serve as major consolidation hubs for freight.

Much of the literature on LTL carriers focuses on tactical planning – generally referred to as *service network design* (see, for example, Crainic (2000), Jarrah, Johnson, and Neubert (2009), Erera et al. (2013a), Lindsey, Erera, and Savelsbergh (2016) or Baubaid, Boland, and Savelsbergh (2020)). Examples of such tactical decisions include determining the number of trailers to operate between any two terminals (if any), the schedules of those trailer movements, and a *load plan* for the various origin-destination pairs in the network. Load plans dictate the sequence of terminals in the linehaul network that freight shipments will follow starting with the origin terminal and concluding with the destination terminal. Therefore, a load plan also prescribes how freight will be consolidated.

Traditionally, load plans for LTL carriers were such that all shipments arriving at a terminal i with terminal d as their ultimate destination are placed on trailers headed to the *unique* next terminal j. In other words, the next terminal is determined by the current location of the shipment and its ultimate destination, regardless of where that shipment originally came from. Moreover, there is a *single* next terminal option for a given pair of current location and ultimate destination.

Recently, Baubaid, Boland, and Savelsbergh (2020) have shown that to accommodate inherent demand uncertainty, relaxing the uniqueness requirement by allowing two next terminal options for location-destination pairs can lead to substantial savings. Furthermore, they show that these savings are comparable to the savings generated when operating with a load plan in which, for each location-destination pair, all terminals can be next terminal options – a plan which is much more complex to manage operationally. They refer to load plans with two next terminal options as 2-alt load plans, whereas traditional load plans are referred to as 1-alt load plans. Figure 1 depicts, for a single destination, d, an example of a 2-alt load plan superimposed on a 1-alt load plan. The arrows in the figure show the next terminal options which freight with destination d can be sent to when located at any of the other terminals in the network.

This work is motivated by the findings in Baubaid, Boland, and Savelsbergh (2020) and focuses on how to effectively operate a 2-alt load plan. Although the ideas explored here can be applied to a more general load plan structure, we restrict our attention to 2-alt load plans due to the benefits they provide relative to the operational overhead they incur. Specifically, this paper studies the operational decisions that LTL carriers have to make dynamically on a day-to-day basis to route freight shipments from origins to destinations. Allowing for an additional next-terminal option for location-destination pairs means that carriers have to dynamically make decisions related to *which* of those two next terminals a particular shipment will be sent to – in addition to *when* it will be dispatched (the only decision for a 1-alt load plan). Moreover, these decisions have to be made recognizing that an uncertain number of new shipments will be collected from shippers and enter the linehaul network the next day. The problem is further complicated by the scale of operations of these carriers; for instance, an LTL carrier can operate a linehaul network of around 150 terminals while handling around 10,000 commodities (defined here as shipments with the same origins, destinations, collection time, and due time) each week (Lindsey, Erera, and Savelsbergh (2016)).

We seek to model this dynamic freight routing problem to address the following question: "What kind of decision-making tool is needed to make good routing decisions in a 2-alt load plan setting in the presence of demand uncertainty?". For the purpose of this study, we will assume that the

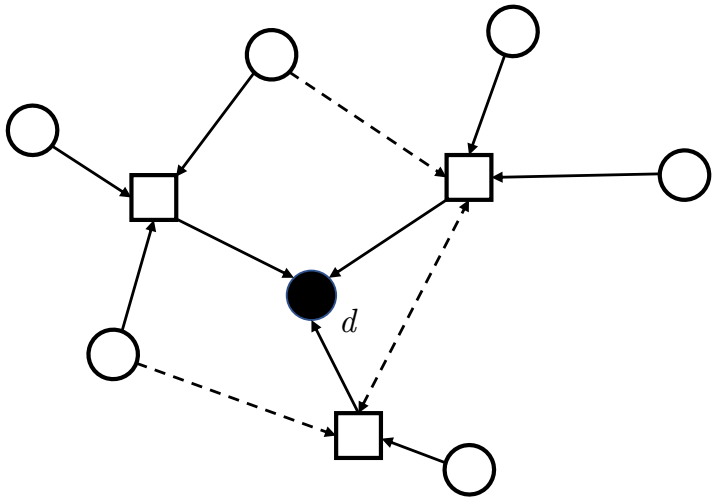


Figure 1: Example of a load plan (shown only for destination d); squares represent BB terminals and circles represent EOL terminals; solid lines depict the 1-alt load plan, and solid and dashed lines together depict the 2-alt load plan.

tactical plan is determined a priori and is fixed for the operating season. This means that we are given, for each day of the week, the number of trailers that operate between every pair of terminals in the linehaul network. For simplicity, we consider an idealized setting in which we assume that all trailer movements take a single day to complete. In addition, we assume that we are given a set of commodities along with distributional information for their daily quantities (measured in pallets). Furthermore, the trailer movement patterns and the commodity patterns repeat from week to week in the operating season. Although our focus here is on the LTL/freight industry, our model is general enough to adapt to any dynamic routing system, e.g., that of the express parcel industry. We will refer to this problem as the Dynamic Freight Routing Problem (DFRP).

This paper makes the following contributions to the literature. We first introduce the DFRP and model this problem as a Markov Decision Process (MDP). Due to the scale of the problem, solving the MDP exactly is hopeless except for the smallest of instances, and so we introduce an offline-online Approximate Dynamic Programming (ADP) solution approach which helps overcome the infamous "curses of dimensionality". The ADP solution approach consists of an offline *learning* or *training* phase in which the values of different states are learned via simulation, followed by an online *execution* or *testing* phase (which corresponds to implementation in practice), and uses a lookup table as the value function approximation method to store the learned values of the states. At each decision epoch in the ADP algorithm, we solve an integer program to determine the best decision to implement. Such a decision is selected based on the one-period contribution to the reward, and an estimate of the reward-to-go represented by the values of the states in the lookup table. We contribute to the ADP literature by investigating and providing a framework for integrating lookup tables into integer programs (IPs) to solve the decision subproblem in the ADP algorithm. To the best of our knowledge, this has not been previously studied. The framework consists of: (1) a modeling approach for the integration of lookup table value function approximations into decision subproblem IPs to form extended decision subproblem IPs, (2) a solution approach, PDS-

IP-Bounding, which decomposes the extended subproblem IPs into many smaller IPs and uses dynamic bounds to reduce the number of small IPs that have to be solved, and (3) an adaptation of the ϵ-greedy exploration-exploitation algorithm for the IP setting. Furthermore, the use of lookup tables (and indeed the integration of lookup tables with an IP) poses size issues that we overcome by aggregating the space of post-decision states (the state of the system right after our decision is implemented and before the uncertainty of the next decision epoch is revealed). We, therefore, compare different aggregation schemes for our problem context, and show that despite the original post-decision state being high-dimensional, by using relatively simple features to "summarize" them, high-quality policies can be produced.

The rest of this paper is organized as follows. In Section 2, we review relevant literature. In Section 3, we state and formally model the DFRP. In Section 4, we present the ADP solution approach which uses lookup tables to store value function approximations and discuss various state aggregation approaches for our problem context. We also demonstrate how to integrate lookup tables with the (mixed-)integer programs that need to be solved at each decision epoch, and present a solution approach for these extended (mixed-)integer programs that exploits their inherent structure. In Section 5 we present our computational study in which the various state aggregation approaches are compared, and the effectiveness of our solution approach demonstrated. Finally, we conclude the paper with a summary and discuss future research directions in Section 6.

2 Literature Review

This paper focuses on the linehaul operations of LTL carriers. That is, we focus on the inter-terminal operations as freight shipments travel between linehaul terminals on their way from origins to destinations. This is in contrast with city operations, which involve the pickup/delivery of shipments from/to individual customers in a relatively small geographical region served by each of the terminals. Since city operations can be modeled as standard vehicle routing problems, they have been studied extensively in the literature. With recent advances in technology, dynamic variants of these problems have also been studied. For a review of dynamic vehicle routing problems we refer the reader to Pillac et al. (2013) and Ritzinger, Puchinger, and Hartl (2016).

When considering a carrier's linehaul operations, the decisions that have to be made can either be strategic, tactical, or operational. Strategic decisions primarily consist of determining the location of the linehaul terminals and how the terminals should be connected. Examples of these problems for LTL carriers can be seen in Cunha and Silva (2007), Campbell (2009), Lin and Lee (2018), and, for a more general reviews of hub/terminal location problems, Alumur and Kara (2008) and Campbell and O'Kelly (2012).

Much of the literature on the linehaul operations of LTL carriers focuses on *tactical* planning. Service Network Design (SND), for instance, is a class of tactical planning problems (not necessarily specific to LTL carriers) that is used ahead of each operating season to determine: (1) the number and types of trailers needed by the carrier, (2) their schedules and movements, and (3) a freight flow plan, which dictates how freight with a given origin and destination will travel through the linehaul network. Therefore, the design of the service network determines how the carrier will meet the demand of the upcoming operating season. We refer the reader to Crainic (2000) and Wieberneit (2008) for reviews of this class of problems.

In the freight/LTL context, the freight flow plan is often referred to as a load plan. A common load plan configuration that LTL carriers use in practice is to stipulate that each origin-destination

pair will have a single freight flow path, and that, for each destination, all freight flow paths into that destination must form an in-tree. This implies that there is only a single next terminal option in freight flow paths for shipments that are at an intermediate terminal and headed to the same destination. To the best of our knowledge, Powell and Sheffi (1983) was the first work on designing load plans for LTL carriers. In addition to introducing the load planning problem, they present a mathematical formulation for the problem along with a local improvement heuristic. The work was subsequently extended in Powell (1986), Powell and Sheffi (1989) and Powell and Koskosidis (1992). Jarrah, Johnson, and Neubert (2009) model the load planning problem on a time-space network to accurately represent consolidation timings, and present a heuristic that uses decomposition techniques combined with slope scaling to solve large-scale instances. Erera et al. (2013a) and Lindsey, Erera, and Savelsbergh (2016) develop IP-based neighborhood search methods for solving large-scale instances of the load planning problem. Our study in this work is motivated by Baubaid, Boland, and Savelsbergh (2020) which considers slightly relaxing the in-tree load plan requirements by allowing freight at an intermediate terminal that is destined for a particular destination to be dispatched on trailers heading to one of *two* possible next terminal options. The authors call this type of load plan a 2-alt load plan, and show empirically that this produces considerable cost savings (on the order of 6%) over in-tree load plans when operating in an environment in which demand is uncertain. Furthermore, these cost savings are comparable to the savings achieved by relaxing the load plan requirements completely. In this work, we, therefore, seek to gain insight into how such a load plan can be implemented in practice operationally. Because these 2-alt load plans allow for two options instead of the traditional one in the in-tree load plans, decisions have to be made not only about *when* to dispatch a pallet from a terminal, but also *where* to send it to next.

In terms of the decision hierarchy, the Dynamic Freight Routing Problem for LTL carriers that we present and study in this work is a linehaul *operational* problem. In contrast to strategic and tactical planning problems, there is relatively little literature on the daily linehaul operations of an LTL carrier. The importance of using dynamic models in transportation/logistics, in general, has been highlighted in works such as Powell, Jaillet, and Odoni (1995) and Powell (2003). As most research in this area assumes traditional load plan structures, the majority of studies relate to deciding the dispatch/closing timings of trailers (in environments where trailer dispatches are not scheduled), some even considering only a single link for the purposes of their analysis. Kleywegt and Papastavrou (1998) develop a Markov Decision Process model to study accepting customer requests and trailer dispatching for a distribution problem between multiple origins and destinations. Cheung and Muralidharan (1999) develop a simulation system that integrates many practical operational decisions (e.g. work rules, load plan, trailer-closing policies, loading times) to study the impact of these factors on service levels. Using this simulation model, they show that making decisions about routing shipments dynamically can significantly improve service levels. Furthermore, they formulate a dynamic programming model to determine a trailer-closing policy dynamically depending on the state of the system. This work was further extended by Cheung and Muralidharan (2000), where the authors develop routing strategies that route shipments adaptively using local/terminal real-time information. Shi et al. (2011) then extended this work by incorporating a correction term in the model that accounts for the interactions of shipments in the system. In, Erera, Karacık, and Savelsbergh (2008) and Erera et al. (2013b), the authors present algorithms for creating cost-effective detailed driver schedules for tactical plans where they take various operational issues and regulations into account. Hejazi and Haghani (2007) present a decision-making

framework for accepting or rejecting new requests based on the real-time status of the linehaul network. Every time a new request arrives, a mixed-integer program is solved to determine what changes would need to be made to profitably accommodate the new request, and a decision is made based on some acceptance criteria to accept the request (and make the necessary operational changes to trailer schedules and the load plan) or reject it. More recently, Simao and Powell (2019) propose an IP lookahead approach to determine the load plan dynamically based on the state of the shipments (both the ones in the system as well as the ones forecast to enter) and drivers in the system. As this lookahead IP is too large to solve directly, they suggest a time-decomposition approach in which small consecutive intervals of the horizon are solved at a time, and the results of each linked together via simulation. Ridouane et al. (2020) design a decision support system for daily load planning. Because of demand uncertainty, they consider the use of alternate paths for freight if there is not enough capacity on the preferred path of the load plan, and use fast heuristics to allow for near real-time load plan adjustments to improve on-time performance. They find that it is possible to improve on-time performance without resorting to additional capacity.

A subset of the work on the operational aspects of LTL applications involves the use of Approximate Dynamic Programming (ADP). ADP is a (heuristic) framework which is used to overcome the curses of dimensionality exhibited by Markov Decision Process models, and has also been applied to a variety of transportation problems including full-truckload carriers (e.g. Powell, Simao, and Bouzaiene-Ayari (2012)), driver scheduling (e.g. Powell, Shapiro, and Simão (2002)), and dyanmic VRP problems (e.g. Ulmer, Mattfeld, and Köster (2017)) among many others. We refer the reader to Powell (2011) for a comprehensive reference on ADP techniques, and Powell, Simao, and Bouzaiene-Ayari (2012) for a reference on ADP in transportation problems. Topaloglu and Powell (2006) present an ADP approach for solving dynamic stochastic integer multi-commodity flow problems, a class of problems to which the DFRP belongs to. The authors compare the use of linear, piece-wise linear, and hybrid value function approximations.

With regards to ADP approaches used in the LTL context, Dall'Orto et al. (2006) present a model that is similar to the one we present in this paper, but only for a single node/terminal in the network. The model decides not only which shipments to route to which next terminals, but also dynamically decides the dispatching of trailers. Furthermore, they present a solution strategy that is based on ADP techniques and approximate the value function (which represents the downstream consequences of their decisions) using a linear model which estimates the unit value of having a particular commodity at a particular location. In our work, not only do we consider the state of the entire system, but we also use "features" in a lookup table approach to estimate the value of states. Our goal is to gain insight into which features better capture the state of the system; these can then be used to design or enhance heuristics for this problem, or to provide key performance indicators that quantify the state of the system as a result of decisions made on a given day. Some of the work has also focused on city operations, such as van Heeswijk, Mes, and Schutten (2015) and later van Heeswijk, Mes, and Schutten (2019) who study the dispatching of freight received by a terminal to local customers in the region. They model the problem as an MDP and use an ADP framework (with linear value function approximations to estimate downstream costs). They propose a number of basis functions for the linear value function approximation, and evaluate their performance.

Finally, in this work, we make contributions to the general ADP methodology. In ADP, the value function is approximated by one of many possible approaches; the most common two are using linear functions of some features of the (post-decision) state variable or using lookup tables. Powell

(2011) and Ulmer and Thomas (2019) discuss the advantages and disadvantages of each approach. Generally speaking, lookup table approaches can produce accurate approximations (when given enough computational time) but, in a basic implementation, would only provide values for states actually visited in the ADP algorithm, i.e., the accuracy of a value of a state in the lookup table is proportional to how many times that state has been visited in the training part of the ADP approach. On the other hand, while linear (or piecewise-linear) models impose structure on the value function, they offer a functional form that, once learned, can readily estimate a value for any state in the model. Traditionally, most ADP problems in which the subproblem is a MIP rely on linear functional forms to estimate the value function since they are easily incorporated into the MIP (see Topaloglu and Powell (2006), Dall'Orto et al. (2006), Toriello, Nemhauser, and Savelsbergh (2010), van Heeswijk, Mes, and Schutten (2019) for examples). To the best of our knowledge, we are the first to provide a basic framework for integrating lookup tables into a MIP framework.

3 The Dynamic Freight Routing Problem for LTL Carriers

In this section, we first define the DFRP and then present an MDP model for the DFRP. We assume that the operating season's tactical plan (the service network design) has been determined exogenously and given is available as input to the DFRP, i.e., we are given the weekly trailer movement patterns and a 2-alt load plan (two next terminal options for all location-destination pairs). Although our presentation assumes a 2-alt load plan structure, any type of load plan can be used.

3.1 Problem Statement

The Dynamic Freight Routing Problem for LTL carriers (DFRP-LTL) is an operational problem in which multiple freight shipments are dynamically routed through the carrier's linehaul network from origins to destinations. The goal is to make routing decisions for the freight so as to maximize the profits for the carrier.

In our setup, we assume that trailer movement patterns and demand patterns repeat weekly throughout the operating season (demand patterns repeat weekly, but are stochastic). Since the operating season is relatively long compared to the representative week that we model, we will assume an infinite horizon for this problem. We also discretize time into days, i.e., each time period represents one day of operations. This level of discretization is appropriate for this application considering that, each workday, LTL carriers typically collect freight during the day from shippers, and collected freight enters the linehaul network in the evening hours (around 7 P.M.) when drivers bring that freight to their respective end-of-line terminals. The collection of freight from shippers during the day is known as the *city operation* (a typical vehicle routing problem with the terminal as the depot) and is not modeled explicitly in this research. We, therefore, take the start of each *operational* day (around 7 P.M.) to represent a decision epoch in the model, where routing decisions need to be made for freight that is currently at a line-haul network terminal (and that is yet to be delivered to its ultimate destination). Such freight can either be: (1) new freight that has been collected in the city operation and brought to that terminal, or (2) transiting freight that is passing through that terminal on its way to its ultimate destination. The number of pallets of a shipment represents its quantity.

Since many aspects of this problem repeat cyclically, we use a cycle length T to capture the length of the desired cycle (e.g., a week) and use the set $\mathcal{T} = \{0, 1, 2, \ldots T-1\}$ to represent the *days* in a cycle (we use cycle and week interchangeably). We represent the operations of the linehaul network of the carrier by a wrap-around time-space network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with cycle length T. Each node in this network, $n = (i, \tau),\ i \in \mathcal{L},\ \tau \in \mathcal{T}$ represents a terminal location i (in a set of terminal locations, $\mathcal{L}$) and a day τ (in the set $\mathcal{T}$). The set $\mathcal{A}$ consists of two sets of arcs: (1) a set of *movement* arcs, $\mathcal{M}$, whose elements are of the form $((i_1, \tau_1), (i_2, \tau_2)),\ i_1 \neq i_2$ and represent scheduled trailer dispatches from terminal i_1 to terminal i_2 leaving on day τ_1 and arriving on day τ_2, and (2) a set of *inventory* arcs, $\mathcal{I}$, whose elements are of the form $((i_1, \tau_1), (i_2, \tau_2)),\ i_1 = i_2$ and represent holding freight at terminal i_1 from day τ_1 to day τ_2. Note that in some cases, we will use the term *flat network* to refer to the linehaul network without the time dimension, i.e., the graph $\mathcal{G}^{flat} = (\mathcal{L}, \mathcal{A}^{flat})$ where $\mathcal{A}^{flat}$ represents how the terminals are connected.

For simplicity's sake, we assume that all movement arcs $m \in \mathcal{M}$ have a travel time of one day. As long as all travel times consist of multiples of some base time period, this is without loss of generality as multi-period arcs can be handled by adding dummy nodes to the set $\mathcal{L}$ and breaking down such arcs to accommodate this. This approach of handling multi-period travel arcs is the one we will assume to simplify the presentation. Alternatively, these arcs can be handled by augmenting the state vector to include expected arrival times of pallets traveling on such arcs as in Topaloglu and Powell (2006). In our setting, therefore, freight will always be located at the terminals as opposed to being in transit on a trailer at the start of a decision epoch. Note that arcs starting from the nodes of day $T-1$ will wrap around and have their end points on day $(T-1+1) \bmod T = 0$. Associated with each movement arc $m = ((i_1, \tau_1), (i_2, \tau_2)) \in \mathcal{M}$ is a number $Q_{i_1 i_2 \tau_1}$ representing the capacity given by the tactical plan on the scheduled trailers traveling on that arc. Similar to the movement arcs, we assume that all inventory arcs are of the form $((i_1, \tau_1), (i_2, \tau_1 + 1)),\ i_1 = i_2$, i.e., inventory arcs have a duration of one day. Note that we assume that inventory arcs are uncapacitated, and therefore, holding a pallet at a terminal is always a feasible option.

As mentioned above, in our setting, demand patterns also repeat from week to week, but are stochastic. Specifically, we define a *commodity* as a group of pallets that share a common set of attributes: origin o, destination d, availability *day* $e \in \mathcal{T}$, and due *day* $l \in \mathcal{T}$. Associated with each commodity is a random variable $D_{o,d,e,l} \in \mathbb{Z}_{\geq 0}$ with finite support which represents the number of pallets that originate at origin o on day e and are promised to be delivered by day l at location d, with distributional information available for each. We also assume that the random variables $D_{o,d,e,l}$ and $D_{o',d',e',l'}$ are independent for $e \neq e'$ (this allows for the optimal policy to be Markovian and deterministic; Topaloglu and Powell (2006)). Note that the origin and destination are terminals in the network as far as the linehaul network is concerned; but the actual pickup/delivery operation from/to the shipper/consignee is part of the city operation, and is exogenous to the problem. We also note that the due day is not considered a hard deadline, and that freight is allowed to be delivered late subject to penalties. However, freight that is in danger of being "too late" (we use a parameter to specify how many days past the due day is considered "too late") will be outsourced to a third-party carrier and will incur outsourcing penalties. The details of the penalties associated with pallets delivered late or outsourced are discussed in Section 3.2. To illustrate the commodity information, consider the commodity $(i_1, i_5, 2, 0)$. This is a commodity that becomes available at terminal i_1 on day 2, and whose delivery is promised by day 0 (of the following week) at terminal i_5. The random variable determining the quantity of this commodity is $D_{i_1,i_5,2,0}$, and in a given

week, the value of this random variable is realized.

Finally, we note that pallets can leave the system in one of two ways: (1) they are delivered either on time (by the due day) or late, or (2) their delivery is outsourced by a third party. We also stipulate that the *sojourn time* for each commodity (defined as the total time a commodity can possibly be in the system) is no more than T, and therefore, an "instance" of a commodity is delivered (either by the carrier or by a third party) before a new instance can potentially appear.

3.2 Markov Decision Process Model for the DFRP

In this section, we formally model the DFRP as a Markov Decision Process (MDP). To that end, we will define each of the components of the MDP in turn.

3.2.1 State

As described in Section 3.1, we discretize time into days and take the start of each operational day to represent a decision epoch. The state of the system at the beginning of decision epoch $t \in \mathbb{Z}_{\geq 0}$ is partly determined by the number of pallets with a given ultimate destination and due day, or (d, l) pair, that are present at each of the locations in the set $\mathcal{L}$ at the start of that decision epoch. Note that an implicit assumption here is that we do not differentiate commodities by their origins; such a differentiation may be necessary if, for example, commodities with the same (d, l) pair but different origins or availability times have different revenues or penalties. While we do not consider such differences for simplicity's sake, they can be accommodated by adjusting the definition of the state variable to differentiate individual commodities but this adjustment is at the expense of increasing the size of the state vector. We let R_{it}^{dl} be an "inventory vector" representing the number of pallets that are present at terminal $i \in \mathcal{L}$ at the beginning of decision epoch t and that should be delivered to terminal $d \in \mathcal{L}$ by day $l \in \mathcal{T}$.

In addition, we need to specify the trailer capacity available at a decision epoch in our state variable as that information is needed to determine the set of feasible decision vectors. To do this, it suffices to include the corresponding day of the cycle for that decision epoch in the description of the state vector as that determines the trailer movement patterns for the day. Therefore, the complete state of this system at decision epoch t, is taken to be $S_t = ([t], R_t)$ where $R_t = (R_{it}^{dl})_{l \in \mathcal{T},\, i,d \in \mathcal{L},\, i \neq d}$ and $[t] := t \bmod T$. The inclusion of the day of the cycle in the state variable also allows the problem to be transformed from a non-stationary periodic MDP model to an "augmented" MDP that is stationary (Riis (1965) and Veugen, van der Wal, and Wessels (1983)). Since the problem then becomes a stationary (discounted) infinite horizon Markov Decision Process, we can rely on MDP theory to conclude that there exists a deterministic stationary policy that is optimal (under some mild conditions that are satisfied here; Puterman (2005)). Having a stationary policy that is optimal on this "augmented" MDP is equivalent to saying that the policy may be day-differentiated for the MDP model that does not include $[t]$ in the state vector, i.e., if two vectors R_{t_1} and R_{t_2} are such that $R_{t_1} = R_{t_2}$, but $[t_1] \neq [t_2]$, then the decisions chosen by the policy need not be the same.

3.2.2 Decisions

Our only decisions in this model are routing decisions. Let x_{ijt}^{dl} represent the number of pallets with ultimate destination d and due day l that are loaded onto trailers traveling along arc $((i, [t]), (j, [t+1])) \in \mathcal{A}$; this represents movement of freight if $i \neq j$ and represents holding freight at terminal i

if $i = j$. Therefore, a decision vector at decision epoch t is denoted by $x_t = (x_{ijt}^{dl})_{l\in\mathcal{T},\, i,j,d\in\mathcal{L}}$, i.e., a routing decision for all pallets that are currently in the system. If the process is in state S_t at the beginning of decision epoch t, the set of feasible decisions is then given by

$$\begin{aligned}\mathcal{X}(S_t) = \{x_t : & \sum_{j\in\mathcal{L}:((i,[t]),(j,[t+1]))\in\mathcal{A}} x_{ijt}^{dl} = R_{it}^{dl} \qquad \forall i,d\in\mathcal{L},\ l\in\mathcal{T},\\ & \sum_{l\in\mathcal{T}}\sum_{d\in\mathcal{L}} x_{ijt}^{dl} \le Q_{ij[t]} \qquad \forall((i,[t]),(j,[t+1]))\in\mathcal{M},\\ & x_{ijt}^{dl}\in\mathbb{Z}_{\ge 0} \qquad \forall l\in\mathcal{T}, i,j,d\in\mathcal{L},\\ & x_t\in\mathcal{Y}\},\end{aligned} \tag{1}$$

where $\mathcal{Y}$ defines the set of routing decisions that complies with the imposed 2-alt load plan structure.

3.2.3 Transitions

We describe how the system transitions from decision epoch t to decision epoch $(t+1)$ in two steps. As the transition of the $[t]$ part of the state variable is clear, we focus on the transitions of the R_t part of the state variable. First, the system transitions deterministically from R_t to a post-decision state $\bar{R}_t$ which is the state of the system after implementing feasible decision $x_t \in \mathcal{X}(S_t)$ but just before the uncertainty of the next decision epoch is revealed, i.e., before new freight enters the system. Next, a probabilistic transition from $\bar{R}_t$ to state R_{t+1} occurs.

An important part of the deterministic transition in our setting is the outsourcing of pallets that are expected to be "too late". These pallets are assumed to be delivered by a third party and are subject to an outsourcing penalty. A pallet can be late by up to p days before it is deemed "too late" with p being the parameter controlling this tolerance. We assume that pallets are outsourced as soon as it becomes clear that the pallet will arrive too late. Outsourcing is handled as part of a transition: at decision epoch t, say a pallet with destination d is at location i, and a decision is made to send the pallet from location i to location j, then if the shortest path in the time-space network $\mathcal{G}$ from j to d is such that the earliest possible arrival time for this pallet is p days after its due day, then the outsourcing penalty is assessed at time t and the pallet will be outsourced and removed from the system, i.e., it will not appear in decision epoch $t+1$.

Therefore, if the process is in state R_t, and feasible decision $x_t \in \mathcal{X}(S_t)$ is taken, then the process transitions to the post-decision state $\bar{R}_t = (\bar{R}_{it}^{dl})$ given by

$$\bar{R}_{i,t}^{dl} = \begin{cases} \sum_{j\in\mathcal{L}:((j,[t]),(i,[t+1]))\in\mathcal{A}} x_{jit}^{dl} & \forall d\in\mathcal{L},\ i\in\mathcal{L},\ l\in\mathcal{T},\ i\neq d,\ \Delta^{out}\big(t+1,i,d,p\big)=0,\\ 0, & \text{otherwise,}\end{cases} \tag{2}$$

where $\Delta^{out}\big(t+1,i,d,p\big)$ returns the value 1 if a pallet will be outsourced (arrives p or more days after the due day l) based on the shortest path in the time-space network $\mathcal{G}$ from i to d starting at time $t+1$, and 0 otherwise. Given the wrap-around nature of the network, Δ^{out} would first determine whether or not a given due day l is in the past or in the future by leveraging the sojourn time assumption stated at the end of Section 3.1. Then, based on where that day actually is relative to the current day, it determines whether or not the pallet will inevitably be outsourced. The post-decision update above captures how many pallets of each (d,l) pair are at each terminal just before the uncertainty of the next decision epoch is revealed. The update also reflects the removal of pallets that are delivered to their ultimate destinations and pallets that will be outsourced.

Then, the (probabilistic) transition from the post-decision state to the next state is given by

$$R^{dl}_{i,t+1} = \bar{R}^{dl}_{i,t} + D_{i,d,[t+1],l} \quad \forall d \in \mathcal{L},\ l \in \mathcal{T},\ i \neq d. \tag{3}$$

3.2.4 Rewards

When the process is in state S_t and feasible decision $x_t \in \mathcal{X}(S_t)$ is taken, then the one-period contribution/reward to the objective function is composed of three parts:

1. Revenue for delivering a pallet to its final destination:

$$C^1_t(S_t, x_t) = \beta \sum_{d,l,i,j} x^{dl}_{ijt} \mathbb{1}_{\{j=d\}},$$

   where β is a per-pallet revenue. Note that this does not include outsourced deliveries.

2. Penalty paid for delivering freight late to its ultimate destination:

$$C^2_t(S_t, x_t) = -c^{late} \sum_{d,l,i,j} \Lambda\left(t,l,p\right) x^{dl}_{ijt} \mathbb{1}_{\left\{j=d \ \wedge\ \Delta^{late}\left(t,l,p\right)=1\right\}},$$

   where c^{late} is a per pallet per day penalty for late delivery, $\Delta^{late}\big(t,l,p\big)$ returns the value 1 if a pallet will be late if delivered to its ultimate destination d by the *end* of decision epoch t (0 otherwise), and $\Lambda\left(t,l,p\right)$ returns the number of days a pallet will be late if $j = d$ and $\Delta^{late}\big(t,l,p\big) = 1$ (0 otherwise). Therefore, this penalty is proportional to the number of days a pallet is late as well as the quantity.

3. Penalty paid for commodities that are delivered by a third party:

$$C^3_t(S_t, x_t) = -c^{out} \sum_{d,l,i,j} \delta(j,d)\ x^{dl}_{ijt} \mathbb{1}_{\left\{j\neq d \ \wedge\ \Delta^{out}\left(t+1,j,d,p\right)=1\right\}},$$

   where c^{out} is a per pallet outsourcing penalty, and $\delta(j,d)$ is the shortest path from terminal j to destination d in the *flat network* $\mathcal{G}^{flat}$, i.e. the outsourcing penalty is proportional to the remaining travel distance in the linehaul network for that pallet.

Thus, the total contribution to the objective function is given by $C_t(S_t, x_t) = C^1_t(S_t, x_t) + C^2_t(S_t, x_t) + C^3_t(S_t, x_t)$. Note that we take the cost parameters β, c^{late} and c^{out} to be the same for all commodities for convenience.

3.2.5 Policy

A policy π is a function that maps each state S_t to a feasible decision vector $X^\pi(S_t) \in \mathcal{X}(S_t)$, where $X^\pi(S_t)$ is the decision induced by the policy π for state S_t. Note that since the time-space network repeats in a cyclical fashion, we seek a deterministic stationary policy where the policy may be day-differentiated from the perspective of the R_t part of the state variable (recall that the day of the cycle is part of the state variable, i.e., $S_t = ([t], R_t)$), and where the expected total discounted reward is maximized, i.e.,

$$\pi^* = \arg\max_\pi \mathbb{E}\left\{ \sum_{t=0}^{\infty} \gamma^t C_t(S_t, X^\pi(S_t)) \middle| S_0 \right\},$$

where S_0 is an initial state and $0 < \gamma < 1$ is an appropriate discount factor.

The Bellman optimality equation for this problem can be written as

$$V^*(S_t) = \max_{x_t \in \mathcal{X}(S_t)} \left\{ C_t(S_t, x_t) + \gamma \mathbb{E}\left[V^*(S_{t+1}) \middle| S_t \right] \right\}, \qquad \forall S_t. \tag{4}$$

where $V(S_{t+1})$ represents the value of being in state S_{t+1} (given by the transition partly described in Equation (3)), and represents the expected total future reward from that state onwards. We choose an optimal decision at each decision epoch according to the decision rule

$$X^{\pi^*}(S_t) = \arg\max_{x_t \in \mathcal{X}(S_t)} \left\{ C_t(S_t, x_t) + \gamma \mathbb{E}\left[V^*(S_{t+1}) \middle| S_t \right] \right\}, \qquad \forall S_t. \tag{5}$$

That is, we choose a feasible decision that optimizes the sum of the *one-period reward* and the discounted *expected reward-to-go* for the states we might transition to in the next decision epoch.

4 Approximate Dynamic Programming Heuristic

The MDP presented in Section 3.2 can only be solved for small instances. Complex dynamic optimization problems such as the DFRP suffer from the well-known curse of dimensionality, rendering the exact solution of all but the smallest of instances beyond our reach. Therefore, we have to resort to heuristic approaches such as ADP.

In this work, we use a version of ADP called Approximate Value Iteration (AVI). This approach is essentially a combined simulation and learning approach that steps forward in time learning the values of different states as the algorithm progresses. Since we modeled the problem as an infinite horizon MDP, we will use a finite simulation horizon H to approximate the infinite horizon, where H (possibly $> T$) is a long-enough simulation horizon, chosen such that any additional discounted rewards that are accumulated in the simulation beyond H are negligible.

Before describing the specifics of the AVI algorithm, we first note that using the definition of the post-decision state, we can re-write Equation (5) as

$$X^{\pi}(S_t) = \arg\max_{x_t \in \mathcal{X}(S_t)} \left\{ C_t(S_t, x_t) + \gamma \bar{V}(\bar{S}_t) \right\}. \tag{6}$$

where $\bar{V}(\cdot)$ is a value function defined on the post-decision states $\bar{S}_t$ (Powell (2011)). The AVI algorithm, therefore, seeks to learn/approximate this value function through a number of simulation iterations. In this work, we use a lookup table to store and update the approximated values for the post-decision states that we visit in our simulation.

The idea of AVI is as follows. For each post-decision state, we have a value $\bar{V}^{\pi_{t,N}}(\bar{S}_t)$ that corresponds to the current approximation of the value of being in this post-decision state (where $\pi_{t,N}$ is the incumbent policy at time t and iteration N). This value will be updated as the algorithm progresses.

We first set a maximum number of simulation iterations, N^{max}. We start the procedure with an initial state S_0 and initial approximation values for all post-decision states, $\bar{V}^{\pi_{0,0}}(\bar{S}_t)$. During a simulation iteration, N, at any decision epoch $t < H$, we find ourselves in a state S_t^N. We then need to choose a decision vector and an associated post-decision vector to transition into the next state at time $t+1$. How we go about finding such a vector depends on whether we are in an *exploration*

step (which we are in with probability ϵ) or an *exploitation* step (probability $1 - \epsilon$). That is, we deal with the exploration-exploitation question using a standard ϵ-greedy algorithm which we adopt due to its simplicity. In an exploitation step, we solve Equation (6) using the most recent value function approximations for the post-decision states. On the other hand, in an exploration step, we randomly select a decision vector and an associated post-decision vector. The purpose of exploration steps is to avoid getting stuck in local optima and to force the algorithm to explore more of the post-decision state space thereby visiting states that it might not otherwise visit, and consequently observe their values. The details of how we select decision vectors and post-decision vectors in an exploration step follow in Section 4.1. Regardless of the current step type, we obtain a decision vector x_t^N, and a corresponding post-decision state vector $\bar{S}_t^N$. Let

$$\hat{v}_t^N := C_t(S_t^N, x_t^N) + \gamma \bar{V}(\bar{S}_t^N). \tag{7}$$

We use this value to update the value of the *previous* post-decision state $\bar{V}^{\pi_{t,N}}(\bar{S}_{t-1}^N)$ using the expression:

$$\bar{V}^{\pi_{t+1,N}}(\bar{S}_{t-1}^N) = (1 - \alpha_{n-1})\bar{V}^{\pi_{t,N}}(\bar{S}_{t-1}^N) + \alpha_{t,N}\hat{v}_t^N \tag{8}$$

where $\alpha_{t,N}$ is a parameter called the step size (its value in our implementation is actually post-decision-state-specific and depends on the number of times a post-decision state's value has been updated). From the post-decision state vector $\bar{S}_t^N$, we then simulate the arrival of new commodities and their quantities according to their distributions, and complete the probabilistic transition to a new state S_{t+1}^N using Equation (3). This process for simulation iteration N repeats until decision epoch $t = H$, after which it terminates, and a new iteration $N + 1$ is started. After iteration $N = N^{max}$ is completed, the AVI algorithm terminates.

Our ADP algorithm consists of two phases. The first phase is an offline *learning* phase where the value function approximations are learned and updated in the lookup table. Its steps correspond exactly to the AVI algorithm that we outlined above. We note that the exploration probability, ϵ, can either be fixed throughout the training phase or be made to decrease with the iteration counter, N; we adopt the former for simplicity. The second phase is an online *execution* phase where the policy induced by the lookup table values (at the end of the learning phase) is fixed, i.e., the lookup table values are no longer updated. In addition to the lookup table values being fixed in all simulation iterations, this phase consists purely of exploitation steps, i.e. it is the optimization of Equation (6) that determines the choice of action and post-decision state at every decision epoch. Other than that, the steps of the execution phase are identical to that of the learning phase. In our work, we use the execution phase as a *testing* phase to evaluate the policy induced by the lookup table values by performing a number of AVI iterations with the now-fixed lookup table.

4.1 Integration of Lookup Tables with IPs

Our AVI implementation relies on a lookup table to store the value function approximations. Furthermore, as mentioned above, we need to solve Equation (6) at every decision epoch to determine the choice of decision and post-decision state vectors. For the DFRP, this involves solving an IP that integrates the most recent approximation values in the AVI algorithm into the model. In this section, we demonstrate how we can integrate these values to construct the IP we need to solve at every decision epoch.

During a simulation run, N, and at decision epoch t, the AVI algorithm is in state S_t, and we need to solve the decision epoch subproblem (6) to determine a transition to the next decision

epoch. Note that we suppress the iteration number, N, in the notation for the state variable to reduce notational clutter as it should be understood that all of this occurs in some simulation run. Recall that the feasible set of the decision epoch subproblem (given by (1)) consists of integer points. Therefore, even ignoring the reward-to-go term in Equation (6), we would still need to solve an IP to find a decision vector. To add the reward-to-go term, we extend the IP to embed the most recent lookup table values in the model.

The first step is determining which post-decision state vectors are *compatible* with the current state S_t. For the current state, a post-decision state vector, $\bar{S}_t$ is compatible if there exists a decision vector that can be used to transition from the current state to that post-decision state using Equation (2). Because enumerating a number of potential post-decision states and checking their compatibility is considerable work, we instead rely on the IP itself to determine which post-decision state vectors are compatible with the current state. To that end, we enumerate a superset of the set of feasible post-decision vectors and use this superset to define the extended IP. The enumeration of this superset is done by checking what possible values each entry in the post-decision vector can take, and then taking the cross-product of all those possible values. This naturally results in a large number of vectors, most of which will be incompatible, but we mitigate this by using aggregation (Section 4.2) and a decomposition solution approach (Section 4.3). We denote this enumerated superset by $\mathcal{P}_t$.

To integrate the lookup table values into the IP, we introduce the following new binary decision variables:

$$z_p = \begin{cases} 1, & \text{if post-decision state vector } p \text{ is chosen,} \\ 0, & \text{otherwise,} \end{cases} \qquad \forall p \in \mathcal{P}_t.$$

Furthermore, as we check compatibility of post-decision state vectors within the IP, we add variables $\bar{R}^{dl}_{i,t}$ that determine the value of the components of the post-decision vector in the model, and then match the resulting vector to a vector $p \in \mathcal{P}_t$ (and set its corresponding z_p value to 1). For exploitation steps, we can then write the *extended decision epoch subproblem* (EDES) IP as follows:

$$\max_{x_t, z} \quad C_t(S_t, x_t) + \gamma \sum_{p \in \mathcal{P}_t} \bar{V}^{\pi_{t,N}}(p) z_p, \tag{9a}$$

$$\text{s.t.} \quad x_t \in \mathcal{X}(S_t), \tag{9b}$$

$$\sum_{p \in \mathcal{P}_t} z_p = 1, \tag{9c}$$

$$\bar{R}^{dl}_{i,t} = \sum_{j \in \mathcal{L}:((j,[t]),(i,[t+1])) \in \mathcal{A}} x^{dl}_{jit}, \qquad \forall d \in \mathcal{L},\ i \in \mathcal{L},\ l \in \mathcal{T},\ i \neq d,\ \Delta^{out}\big(t+1, i, d, p\big) = 0, \tag{9d}$$

$$\bar{R}^{dl}_{i,t} = \sum_{p \in \mathcal{P}_t} p^{dl}_{i,t} z_p, \qquad \forall i \in \mathcal{L},\ d \in \mathcal{L},\ l \in \mathcal{T}, i \neq d, \tag{9e}$$

$$z_p \in \{0, 1\}, \qquad \forall p \in \mathcal{P}_t. \tag{9f}$$

In the above model, $\bar{V}^{\pi_{t,N}}(p)$, represents the most recent entry in the lookup table for post-decision state vector p. The objective function in this model is just a re-formulated version of that of Equation (6). Since only one z_p variable can be equal to one, the two expressions are equivalent. Constraints (9b) ensure that the selected decision vector is feasible. Constraints (9d) and (9e) are

the aforementioned compatibility constraints that check for compatibility between the current state, feasible actions, and potential post-decision state vectors. In addition, they ensure that the correct z_p variable is set to one, i.e., for the vector p whose components exactly match the post-decision state vector determined by the model. We denote by (x_t^*, z^*) the optimal solution of this extended IP, and it is this solution that is used to transition to the next decision epoch in the AVI algorithm.

We use a similar model in exploration steps in the learning phase, but the difference here is that we would like to encourage the algorithm to choose a state vector that is not necessarily optimal with respect to Equation (9a). To accomplish that, we replace the values $\bar{V}^{\pi_{t,N}}(p)$ in (9a) with randomly generated coefficients drawn from a uniform distribution $U \sim [a, b]$ where a and b are either pre-specified numbers or set as the minimum and maximum current values in the lookup table. Although simple, this resulted in considerable improvements in the values of the resulting policies as seen in Section 5.4.

It is clear, however, that the size of this IP can become an issue. Specifically, the number of z variables is proportional to the number of enumerated post-decision state vectors (which may indeed be quite large). In addition, the number of constraints (9d) and (9e) is proportional to the dimensions of the post-decision state variables. To help alleviate this issue, in Section 4.2, we consider aggregation schemes that will help reduce both the dimensionality of the post-decision state vectors and the number of enumerated post-decision vectors at each decision epoch, thereby reducing the size of this extended IP. Furthermore, in Section 4.3, we present a solution approach for this extended IP that takes advantage of its inherent structure and that can be more efficient than solving the extended IP directly.

4.2 Aggregation Approaches

A major challenge when using a lookup table as the form of the value function approximation is that the accuracy of its entries is a function of how many observations we have for the corresponding post-decision states. Therefore, the more times we visit a post-decision state in the AVI algorithm, the more accurate its value will be in the lookup table. However, the number of possible post-decision states grows rapidly with the number of terminals and the number of commodities. This is not only a problem for the accuracy of the values in the lookup table, but also for the size of the EDES IP as discussed in Section 4.1. Aggregation, therefore, helps reduce the number of post-decision states and their dimensionality by grouping "similar" post-decision vectors together and simplifying their representation. In this section, we present a number of aggregation approaches for the DFRP, and we compare their performance later in Section 5.2.

An aggregation of the post-decision state space is a function $\mathcal{U} : \mathcal{P} \mapsto \mathcal{Q}$ where typically $\mathcal{Q}$ is of a much smaller dimension than $\mathcal{P}$. An aggregation scheme addresses the question: "Is it possible to come up with more compact representations of the post-decision state space such that different post-decision state vectors that are similar (with respect to a set of problem features) are mapped into the same compact representation and share the same lookup table entry?". Therefore, after aggregation, the lookup table entries will consist of entries for the *aggregated* post-decision state vectors. Ideally, an aggregation should have the following features (but there are inherent trade-offs here): (1) the space $\mathcal{Q}$ is low-dimensional, and (2) the aggregation captures or retains most of the problem's important features and such features should inform the approximated values in the lookup table. Furthermore, because of the structure of our EDES IP, we also would like our aggregation to be expressible in the context of an IP, i.e., we should be able to replace constraints (9d) with whatever new definition we obtain for an aggregation post-decision state so that the extended IP

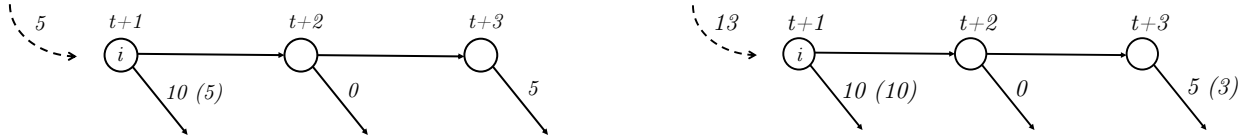
can internally perform this mapping.

The expected reward-to-go in Bellman's equation can be seen as consisting of two main parts: (1) the expected reward that we earn in the future from commodities that are in the system at time t but have not yet generated a reward, and (2) the expected reward that we earn from commodities that will enter the system in future. In this work, we focus on the former when developing aggregation schemes since as rewards are discounted, it is likely that commodities that are currently in the system will become reward-generating earlier than commodities that are not yet in the system. Therefore, "prioritizing" these commodities when choosing a post-decision state is likely to be more beneficial.

To ensure that the aggregation captures the essence of a post-decision state for commodities in the system, there are a number of important post-decision state features that we should consider. These include the current location of pallets (and how that relates to their ultimate destination), their due days, their quantities, and the transportation capacity of the network (which the commodities will ultimately compete for). Below, we propose a number of aggregation schemes for the DFRP, which we later compare computationally. In all the aggregation schemes presented below, although we focus on aggregating the inventory vector component of the post-decision state vector, $\bar{R}_t$, we note that our lookup table approximations are day-differentiated, i.e. the lookup table entries are stored in the form $([t],\mathcal{U}(\bar{R}_t))$. Including the day of the cycle helps capture demand and capacity patterns, and preliminary experiments showed benefits when including the day of the cycle in the aggregated representation.

Slacks: In this scheme, we have an aggregated post-decision state vector with dimension $T-1$ and elements $\bar{R}_{jt} = \sum_{S(i,t,d,l)=j} \bar{R}^{dl}_{it}$ for $j \in [-p, T-1-(p+1)]$. In other words, this vector captures and tallies the number of pallets in the post-decision state vector that have *slack* values equal to j. Specifically, if we let, $\bar{l}$ be the *decision epoch* in which delivery of a pallet is promised (for a commodity with due day l), then we can define the slack $S(i,t,d,l)$ as $\bar{l} - (t+1+\delta_{[t+1]}(i,d))$. This value can be negative if the earliest the pallet can arrive is beyond the promised delivery decision epoch. While it captures due days and incorporates information on how soon the pallets can arrive at their destinations, this aggregation does not consider any capacity information as it considers each pallet individually and does not account for any interactions between pallets.

Total Slack + Congestion (Lookahead): In this scheme, we use a compact two-dimensional aggregated post-decision state vector. The first component of this aggregated vector represents the total slack in the system, i.e., the sum of the slack values of all the pallets in the post-decision state (where slack values are defined as above). The second component is a congestion measure introduced to capture capacity interactions between commodities that are at the same location in the post-decision state, and are headed to the same destination (since they have the same alts and will likely compete for capacity). Specifically, this component captures the total capacity deficit across all terminal-destination pairs in the post-decision state. Some pairs may have no outgoing capacity in the post-decision state (on day $[t+1]$), but may have considerable outgoing capacity on day $[t+2]$. To capture this, we "look ahead" to find the earliest positive outbound capacity and use that to compute the capacity deficit. That is, if we let κ^t_{id} denote the total outgoing capacity on decision epoch t for terminal-destination pair (i,d) according to the 2-alt load plan, then we define the outgoing capacity of terminal-destination pair (i,d) in the post-decision state as $\kappa_{id} = \kappa^{t^*}_{id}$ where $t^* = \min\{\ell | \ell \geq t+1 \wedge \kappa^\ell_{i,d} > 0\}$. Then, we can write the second component in the vector as $\sum_{i,d} \max\{\sum_l \bar{R}^{dl}_{it} - \kappa_{id}, 0\}$. Note that this "lookahead" effectively collapses the time dimension and, therefore, does not take waiting into account. In some cases, waiting can still allow the pallets



(a) Congestion measure = 0

(b) Congestion measure = $3 \cdot 2 = 6$

Figure 2: Illustration of pallet-days congestion measure.

to reach their destinations and thus might be favorable, but waiting can be detrimental if it causes pallets to be late or outsourced. Preliminary experiments showed that incorporating this lookahead is beneficial in our instance settings.

Total Slack + Congestion (Pallet-Days): In this scheme, we again use a compact two-dimensional aggregated post-decision state vector similar to Total Slack + Congestion (Lookahead). While the first component here is exactly the same as the one used in the Lookahead variant, we modify the congestion component to address the waiting issue discussed above. Specifically, the congestion component now represents the total number of pallet-days required to clear out any deficits remaining from terminal-destinations pairs in the post-decision state. We illustrate this congestion component for a single terminal-destination pair in Figure 2. In this figure, a portion of the time-space network for terminal i is shown for 3 time periods starting from time period $t+1$; the horizontal arrows indicate holding arcs while the diagonal arrows indicate the total outgoing capacity from terminal i that is available for destination d according to the 2-alt load plan. In the first example shown in Figure 2a, suppose a decision vector at time t sends a total of 5 pallets to terminal i (destined for d). Then, since there is sufficient outgoing capacity on that load plan to handle those 5 pallets, there is no deficit, and the congestion measure in this case is zero. However, in Figure 2b, we send a total of 13 pallets to terminal i at time t (destined for d), and we will have a deficit of 3 pallets in the post-decision state that need to be cleared from terminal i. Since there is no outgoing capacity at time $t+2$, those 3 pallets are expected to remain in the system for two time periods until they can be moved out of terminal i, so the pallet-days congestion measure's value is $3 \cdot 2 = 6$. As this accounts for time and capacity, this is an improvement over the Lookahead congestion measure. However, the number of pallet-days vectors can be quite large, and in Section 5.2, we discuss how we use scaling/partitioning in our implementation to handle this issue.

Finally, observe that all three aggregation schemes are expressible in the EDES IP; we omit the presentation for the sake of brevity.

We have also experimented with a hierarchical decision-making process:

1. We first solve the EDES IP using the Slacks aggregation scheme and obtain the optimal objective function value, $\hat{v}_t$.

2. We add the following constraint to the IP:

$$C_t(S_t, x_t) + \gamma \sum_{p \in \mathcal{P}_t} \bar{V}^{\pi_{t,N}}(p) z_p \geq \hat{v}_t,$$

and re-optimize the IP with the objective of maximizing the *projected profit* for all commodities currently in the system. We define the projected profit as the profit obtained by

emptying out the system, i.e., delivering or outsourcing all the pallets currently in the system at time t. To optimize for this projected profit, we need to optimize over a maximum horizon with length equal to the cycle length T (with all its associated arcs and their capacities) that ensures all commodities can leave the system either via delivery or outsourcing. This means new variables will be added to the model for future time periods (from t to $t+T$). We do not consider new arrivals in this optimization and are only concerned with commodities that are currently in the system.

However, our experiments showed that this hierarchical decision-making process did not yield policies that were consistently or noticeably better.

4.3 Extended IP Solution Algorithm

In this section, we present an *exact* solution algorithm for the EDES IP (9). This algorithm solves the extended IP by solving many small IPs where each IP corresponds to a single post-decision vector. Furthermore, it incorporates dynamic bounds to eliminate post-decision state vectors from consideration thereby reducing the number of small IPs that have to be solved.

The approach heavily relies on fixing a potential post-decision state vector, $\bar{R}_t$, in the EDES IP (9). Notice that by fixing a potential post-decision state vector, the problem reduces to finding a decision vector (if the post-decision state vector is compatible with the current state) that optimizes the one-period reward $C_t(S_t, x_t)$. This IP is much smaller in size and we will refer to it as the post-decision state IP (PDS IP).

The approach is motivated, in part, by the fact that most likely only a small percentage of the commodities in the system at time t are reward-generating, and, therefore, that the majority of the reward comes from the reward-to-go part of the objective function (especially after the values in the lookup table have been updated). For this reason, we will sort the potential post-decision state vectors that we enumerate at time t in non-increasing order of their current lookup table values, i.e., $\bar{V}_t^{\pi_{t,N}}(p^{(1)}) \geq \bar{V}_t^{\pi_{t,N}}(p^{(2)}) \geq \cdots \geq \bar{V}_t^{\pi_{t,N}}(p^{(|\mathcal{P}_t|)})$. Our approach relies on going through this ordered list starting from $\bar{V}_t^{\pi_{t,N}}(p^{(1)})$, solving a small PDS IP for each vector to determine its value, and keeping track of the best objective found in the process.

Furthermore, we incorporate a simple bounding mechanism to enhance this search procedure. Specifically, we first solve the EDES myopically, i.e., with only the one-period reward term of the objective function and without any consideration of the reward-to-go of the post-decision state we will end up in. That will give us an upper bound on the one-period reward we can earn in that decision epoch, which we will call c^{UB}. Then, going down the sorted list of enumerated post-decision state vectors, as soon as a *compatible* vector is found with objective value $\hat{v}_t^p$ for the EDES IP, we can remove from the list all vectors whose lookup table value are less than or equal to $v_t^{cutoff} := \frac{\hat{v}_t^p - c^{UB}}{\gamma}$, as they cannot yield a better objective value for the EDES IP. This reduces the number of PDS IPs that we need to solve. For an illustration of this idea, see Figure 3. In this example, the first vector in the list was found to be incompatible. The second vector, however, was found to be compatible with an objective value of $\hat{v}_t^p$ for the EDES IP, and so any vector with a lookup table value less than or equal to v_t^{cutoff} was removed from the list.

This means that we can terminate the search procedure as soon as we find a vector that is compatible and has an objective function value for the EDES IP that is provably optimal. As we check post-decision state vectors in non-increasing order of their values, it is easy to see that when checking the j^{th} vector, an upper bound on the objective value of the EDES is given by

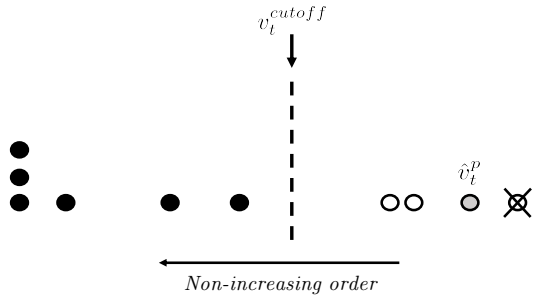


Figure 3: Example illustrating the bounding idea for post-decision state vectors whose PDS IPs need to be solved; circles represent the lookup table values of the vectors arranged on the number line (stacked circles indicate that multiple vectors have the same value); black-filled circles represent vectors that need not be considered.

$\bar{V}_t^{\pi_{t,N}}(p^{(j)}) + c^{UB}$. Therefore, this implies that as soon as a vector achieves a *PDS IP* objective value of c^{UB} the algorithm can terminate having found a provably optimal solution.

Putting all of this together, in Algorithm 1, we present the pseudocode detailing the steps of the algorithm (which we call PDS-IP-Bounding) to solve the EDES IP for decision epoch t.

Algorithm 1 PDS-IP-Bounding

Input: $\mathcal{P}_t$.

1: Initialize: $\hat{v}_t^{best} = -\infty$, $x_t^{best} = \emptyset$, $p_t^{best} = \emptyset$.
2: Determine c^{UB} by solving Model (9) myopically.
3: $PDSList \leftarrow$ Sort vectors in $\mathcal{P}_t$ in non-increasing order of their lookup table values.
4: **while** $PDSList \neq \emptyset$ **do**
5: $p \leftarrow$ First vector in $PDSList$.
6: Remove p from $PDSList$.
7: Solve PDS IP for vector p.
8: **if** p is compatible **then**
9: $\hat{v}_t^p \leftarrow$ Objective value for EDES IP for vector p.
10: **if** $\hat{v}_t^p = \bar{V}_t^{\pi_{t,N}}(p) + c^{UB}$ **then** ▷ Checking for optimality.
11: $\hat{v}_t^{best} \leftarrow \hat{v}_t^p$.
12: Update decision vector, x_t^{best}, and post-decision state vector, p_t^{best}.
13: Optimal solution found. Go to line 24.
14: **end if**
15: **if** $\hat{v}_t^p > \hat{v}_t^{best}$ **then**
16: $\hat{v}_t^{best} \leftarrow \hat{v}_t^p$.
17: Update decision vector, x_t^{best}, and post-decision state vector, p_t^{best}.
18: **end if**
19: Remove all post-decision state vectors with lookup table values less than or equal to $\frac{\hat{v}_t^p - c^{UB}}{\gamma}$ from $PDSList$.
20: **else**
21: Go to line 4.
22: **end if**
23: **end while**
24: **return** Optimal solution: $(\hat{v}_t^{best}, x_t^{best}, p_t^{best})$.

5 Computational Experiments

We conduct a set of computational experiments to:

1. demonstrate the effectiveness of the exploration scheme proposed in Section 4.1 for the IP decision subproblems,

2. compare (as part of the ADP solution approach) the various aggregation approaches proposed in Section 4.2 in terms of the quality of the policies they produce as well as the runtimes of their corresponding ADP algorithms, and

3. demonstrate the effectiveness of the PDS-IP-Bounding algorithm for solving the EDES IPs proposed in Section 4.3 compared to directly solving them as given by Equations (9) using a commercial solver.

We organize this section as follows. First, we describe the instance generation procedure and relevant problem parameters in Section 5.1. Next, in Section 5.2, we compare the aggregation approaches for the post-decision states, and benchmark them against a standard myopic policy. In

Section 5.3, we analyze the performance of the proposed PDS-IP-Bounding algorithm for solving the EDES IPs. Finally, in Section 5.4, we demonstrate the effectiveness of the exploration scheme in the context of our EDES IPs.

All algorithms were coded in Python and all experiments were performed on a 20-core machine with Intel(R) Xeon(R) 2.30GHz processors and 256 GB of RAM running Red Hat Enterprise Linux Server 7.6, and with Gurobi 9.0.1 as the IP solver.

5.1 Instances and Model Parameters

After specifying a cycle length, T (we use $T = 5$), our instances are generated using a three-step procedure:

1. Generation of network topology: We generate the linehaul network, $\mathcal{G}^{flat}$, as a layered graph with four layers. The first layer contains a set of origin EOL terminals (origin layer), the second and third layers contain sets of BB terminals, and the final layer contains a set of destination EOL terminals (destination layer). In our implementation, only nodes in consecutive networks are connected with the arcs always directed towards the destination layer, and we include all such arcs. An example of a layered network (for one of the configurations that we use) can be seen in Figure 4. We will use the configuration of the network, i.e. the number of nodes in each layer starting with the origin layer and ending with the destination layer, to partially differentiate our instances. For example, the network in Figure 4 will be referred to as a 4-3-3-3 instance. One could think about this layered structure as an abstraction of the transcontinental part of a U.S. carrier's service network; in particular, the origin layer may contain terminals on the East Coast while the destination layer may contain terminals on the West Coast, and the two intermediate layers are BB terminals that are somewhere in between.

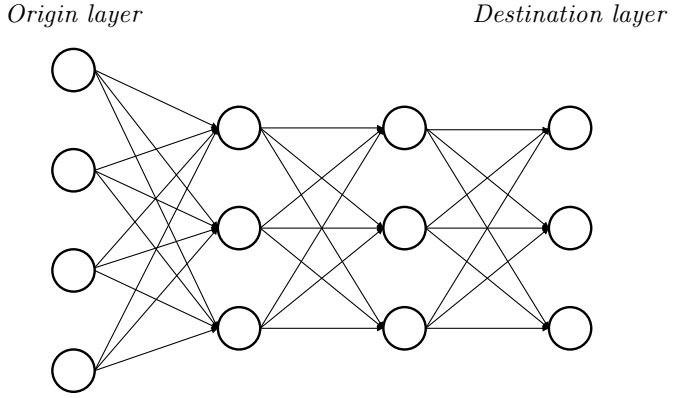


Figure 4: Example of layered network (4-3-3-3).

2. Generation of commodities: For each instance, and for each origin-destination pair, (o, d), we include a commodity of the form $(o, d, 0, 4)$ and one of the form $(o, d, 3, 2)$, i.e. since the minimum number of hops across the graph is three, each commodity has one day of slack while making that journey. Therefore, for a 4-3-3-3 instance, for example, the number of commodities is $4 \cdot 3 \cdot 2 = 24$. Depending on the instance, commodity demands either follow a

uniform distribution over a common support of $[0, q]$ where q is a user-defined parameter for the maximum number of pallets for a commodity, or a discrete triangular distribution (Kokonendji and Zocchi (2010)) with a randomly generated mode, and a minimum and maximum of zero and q, respectively. We assume that all commodity random variables are mutually independent for convenience.

3. Determining the load plan and capacity: To determine the service network for the instances, our overall objective is to have, for each commodity, two types of paths: (1) tightly capacitated paths that enable pallets of that commodity to reach their destination one time period early, and (2) paths with more capacity that enable the commodities to arrive exactly on time (and some options that enable the pallets to arrive late). The network $\mathcal{G}^{flat}$ is first converted into a wrap-around time-space network, $\mathcal{G}$, by adding inventory arcs, and replicating arcs, $(i, j) \in \mathcal{G}^{flat}$ so that on each day $\tau \in \mathcal{T}$, there is a copy, $((i, \tau), (j, (\tau + 1) \bmod T))$, of the original arc (i, j). Then, we use the following two-phased approach to design the service network:

   a. We first remove the (intermediate-layer) BB terminals from the bottom half of the graph; when "splitting" the graph into halves and the number of terminals in a layer is odd, the bottom half has the smaller number of terminals. Next, we solve a 2-alt deterministic optimization problem (Baubaid, Boland, and Savelsbergh (2020)) using the expected demand values and a uniform trailer capacity of $\bar{Q}$ while stipulating the two following conditions: (1) nodes in the origin layer only have one alt available in this phase thereby choosing a single alt to the BB terminals in the top half of the first intermediate layer, and (2) 80% of the expected demand of each commodity arrives exactly on time, and 20% arrives one day late (we set $p = 1$).

   b. Add the BB terminals back, and augment the design obtained in Part a. by solving a 2-alt deterministic optimization problem using 20% of the expected demand values and with a trailer capacity of $\bar{Q}$ while requiring that: (1) flows into the destinations arrive one day early, and (2) nodes in the origin layer have to have outgoing flows going through a BB in the bottom half of the first intermediate layer (the second alts for the origin nodes).

   Thus, the top half of our network should have reasonable capacity to service commodity demands with the catch that they will "only" arrive on time, while the quicker paths in the bottom half arrive earlier but are tightly capacitated and may easily result in congestion. As alts primarily help alleviate congestion, this congestion element is useful in our instance design to see if our approach can exploit the alt structure and avoid situations which result in congestion.

We summarize the instance settings in Table 1. The first two digits of the instance label indicates the number of terminals in the origin layer, followed by three single digits for each of the remaining three layers; U/T represents the demand distribution (uniform and triangular, respectively); and this is followed by the value of $q = \bar{Q}$.

We use $\gamma = 0.8$ as the discount factor, $\beta = 3$ as the revenue for the delivery of a pallet (only applies if delivered by the carrier), $c^{late} = 2$ as the penalty for delivering a pallet late, and $c^{out} = 1$ as the penalty for outsourcing the delivery of a pallet. We set $\epsilon = 0.15$ as the probability that a training decision epoch is an exploration step; preliminary experiments with a few instances has shown this

Table 1: Instance settings

Instance	No. of EOLs	No. of BBs	Total No. of Commodities	q	Q
04333-U-5	7	6	24	5	5
06333-U-5	7	6	24	5	5
08433-U-5	11	7	48	5	5
10433-U-5	13	7	60	5	5
04333-U-10	7	6	24	10	10
04333-T-5	7	6	24	5	5
06333-T-5	7	6	24	5	5
08433-T-5	11	7	48	5	5
10433-T-5	13	7	60	5	5
04333-T-10	7	6	24	10	10
12544-U-5	16	9	96	5	5
06333-U-10	7	6	24	10	10
08433-U-10	11	7	48	10	10
10433-U-10	13	7	60	10	10
12544-U-10	16	9	96	10	10
12544-T-5	16	9	96	5	5
06333-T-10	7	6	24	10	10
08433-T-10	11	7	48	10	10
10433-T-10	13	7	60	10	10
12544-T-10	16	9	96	10	10

value to provide substantial improvements to the resulting policy. For AVI, we set $H = 20$ as the finite simulation horizon length, i.e., there are 4 cycles in each AVI iteration. We initialize all entries in the lookup table to be zero at the start of training, and set $\alpha := 1/\Theta(\bar{S}^a, N)$ where $\Theta(\bar{S}^a, N)$ is a function that keeps track of the number of times a particular entry $\bar{S}^a = ([t], \mathcal{U}(\bar{R}_t))$ in the (aggregated) lookup table has been observed up to AVI iteration N and time period t.

To ensure that our ADP algorithm learns a good policy no matter the starting state, we follow the approach of van Heeswijk, Mes, and Schutten (2019) to generate a set of realistic initial states, $\mathcal{S}^{initial}$, which we will sample from at the start of the ADP training and testing iterations. To accomplish this, we add an 'initialization' phase (prior to training) which is used to build the set $\mathcal{S}^{initial}$ (initialized to be empty). We run 50 AVI initialization iterations; in each of these, AVI starts from a randomly generated initial state vector (which is most likely not realistic), and follows a myopic policy, i.e., a policy that completely disregards the reward-to-go component in the objective function of the decision epoch subproblems. The state encountered at decision epoch $t = \frac{H}{2} = 10$ in each iteration is then added to the set $\mathcal{S}^{initial}$ (this helps to avoid any warm-up or cool-down effects). Subsequently, in the training phase of ADP, we perform 1,000 AVI iterations (a total of 20,000 time periods), and at the start of each, we sample a random initial state from $\mathcal{S}^{initial}$ with a probability proportional to the frequency of initialization iterations with which it was observed. In the testing phase of ADP, we perform 200 AVI iterations (4,000 time periods) with the fixed policy, but each batch of 25 iterations will start from the same randomly sampled initial state. That is, for each of the initial states sampled in testing, we perform 25 sample paths with the goal of getting an accurate estimate for the value of that initial state. As a consequence, a maximum of eight initial states will be drawn from $\mathcal{S}^{initial}$ in testing.

5.2 Analysis of Aggregation Approaches

To ensure a valid comparison in this experiment, for a given instance and phase (initialization, training, and testing), we use the same sample paths across all aggregation approaches. All ADP runs of this experiment utilized the PDS-IP-Bounding algorithm to solve the EDES IPs.

For the Total Slack + Congestion (Pallet-Days) scheme, we scale the aggregated post-decision state space along the congestion measure axis to reduce the number of PDS vectors. Specifically, we present two scaling variants:

1. Fixed Scaling: The total number of pallet-days is divided by 30 and rounded down, e.g., PDS vectors with the same total slack component and with a total number of pallet-days in the range from 0 to 29 all share a single lookup table cell.

2. Varying Scaling: The total number of pallet-days is divided by the number of terminal-destination pairs in the PDS vector for which we might find a positive number of pallets and rounded down, i.e., an "average" number of pallet-days is computed. The number of terminal-destination pairs in the PDS vector for which we might find a positive number of pallets can be determined at the start of each time period before setting up the PDS IP and passed along to the IP as a parameter.

In columns 3-6 of Table 2, we compare the average total testing (discounted) sample path reward (taken over the 200 testing sample paths) for the aggregation schemes: Slacks, Total Slack + Congestion (Lookahead), Total Slack + Congestion (Pallet-Days Varying), and Total Slack + congestion (Pallet-Days Fixed). Because the Slacks scheme was relatively time-consuming to solve on many of the instances, we limit our runs for this approach to instances for which the total runtime is less than 100 hours. We include averages for the first ten instances (which are those for which Slacks values were obtained) and overall averages for the other aggregation approaches in the penultimate pair of rows.

Initially focusing on the first ten instances, we observe that the two TS+C (Pallet-Days) variants offer an improvement over the other two aggregation approaches. In particular, their policies offer an average improvement of about 8% over the Lookahead variant. On the other hand, the average improvement over the Slacks aggregation scheme is less pronounced, although there is a noticeable improvement in the case of the U instances. The improvement in performance offered by the TS+C (PD) variants is even more pronounced when all 20 instances are considered. In particular, the policies found by the two TS+C (PD) variants yielded a much higher overall average improvement of 28-29% over TS+C (L). These two approaches perform fairly evenly with TS+C (PD Fixed) performing slightly better overall.

Despite the TS+C (PD) variants performing only marginally better than Slacks on the first ten instances, we argue that they are the better choices for an aggregation scheme. While the first component of the TS+C (PD) aggregation representation is an aggregated version of the Slacks vectors, which may reduce its accuracy, the second component contains additional information about a different aspect of the system. In particular, TS+C (PD) captures capacity/congestion information whereas Slacks does not. Furthermore, the TS+C variants offer a more compact representation of the post-decision state when compared to Slacks. The competitive performance of Slacks on the first ten instances is most likely because of the equivalence between capacity/congestion and slack values in the paths created by our instance generation procedure in the service network. Recall that we create paths that arrive a day early (a slack of 1) that are tightly

Table 2: Average ADP testing reward for aggregation approaches

Instance	Myopic	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333-U-5	59.93	84.47	96.68	96.77	98.62
06333-U-5	-3.57	64.65	86.44	79.67	74.45
08433-U-5	163.25	220.81	183.99	223.33	244.15
10433-U-5	73.85	200.16	174.53	206.16	197.45
04333-U-10	93.54	172.45	168.82	189.79	174.91
04333-T-5	64.66	103.62	107.92	108.76	108.91
06333-T-5	105.31	161.13	158.00	163.51	163.84
08433-T-5	261.09	228.31	224.02	217.06	228.40
10433-T-5	208.48	283.07	227.99	280.05	267.86
04333-T-10	127.79	176.38	179.43	174.66	178.52
12544-U-5	214.47	-	287.97	307.79	311.79
06333-U-10	-21.47	-	81.84	144.81	162.76
08433-U-10	351.83	-	334.35	429.82	408.97
10433-U-10	399.61	-	426.38	520.65	508.77
12544-U-10	502.76	-	337.89	539.35	648.14
12544-T-5	213.47	-	314.36	329.62	340.61
06333-T-10	16.17	-	148.65	173.54	178.36
08433-T-10	316.77	-	376.29	402.57	458.35
10433-T-10	489.81	-	406.19	589.12	585.99
12544-T-10	886.10	-	477.13	966.91	863.36
Average (First 10)	115.43	169.51	160.78	173.98	173.71
Overall Average	226.19	-	239.94	307.20	310.21
% Impr. Myopic (First 10)	-	46.84	39.29	50.72	50.49
% Impr.	-	-	6.08	35.81	37.14

capacitated, and paths that arrive on time (a slack of 0) that are less capacitated. Therefore, it is likely that the ADP algorithm was able to learn to avoid paths that offer lower slack values (and therefore more congestion) by virtue of this equivalence. However, in much larger real-world settings where there may exist multiple paths with the same slack values but different capacities, the addition of the congestion measure would provide more useful information about the state of the system.

We also include in Table 2 the reward values obtained by using a standard myopic policy which, at each time period, optimizes only the one-period reward without any explicit regard for the impact of the resulting decisions on future time periods. Due to the nature of the immediate reward of the DFRP, it is highly likely that there will be many optimal solutions for the myopic IP: for instance, there might be many feasible decision vectors that result in zero immediate reward (no pallets lost to outsourcing but no pallets delivered). To ensure that the myopic policy chooses decisions that are reasonable, we introduce a small positive reward into the objective function for moving pallets closer to their destinations. This reward is proportional to the reduction in the shortest path distance to pallets' destinations in the service network $\mathcal{G}$ (and is allowed to be negative). It is only used in the myopic IP to choose reasonable decisions – decisions that move pallets closer to their destinations as is commonly done in practice by local terminal dispatchers – and is subtracted from the objective function value after the IP is solved. We also include in the last pair of rows the percentage improvement in the average reward values that the ADP approaches provide over the myopic approach.

We first note that all four ADP approaches yield improvements over the myopic policy. This demonstrates that the ADP approaches are capable (to varying degrees) of learning how to avoid some of the pitfalls that the myopic approach will run into. Even when the myopic policy was unable to find a policy with a positive total reward such as in the cases of 06333-U-5 and 06333-U-10, the ADP approaches were able to find policies that have high positive reward values. On average, the TS+C (L) approach yielded a 6% improvement over the myopic policy when all 20 instances are considered, but notably seemed to struggle on the largest two instances, 12544-U-10 and 12544-T-10 falling behind the myopic policy. On the other hand, the two TS+C (PD) approaches yielded substantial improvements of 35-37% over the myopic value. Furthermore, they are more consistent and perform much better on the largest two instances unlike their (L) counterpart.

Since the myopic policy prefers to keep pallets moving along the shortest path in the network $\mathcal{G}$, this will likely cause congestion as the pallets are funneled through the more capacitated paths in the service network. On the other hand, the ADP approaches are able to learn (through the features we use) the value of different post-decision states, identify better system states, and exploit the availability of the alts in the service network, thereby leading to higher reward values. Figure 5 shows the average performance metrics (taken over all 20 instances), % On Time, % Late, and % Outsourced, which highlight where the differences in the average reward values are coming from. These percentages are based on commodities whose entire sojourn time fits within the time period window $[5, 15]$ within an AVI iteration to avoid warm-up and cool-down effects, i.e., they are computed for these commodities in each sample path in the testing phase, and the average over all testing sample paths is reported in the table. We observe that, on average, the two TS+C (PD) schemes exhibit a much higher on-time delivery percentage than the Myopic approach and even the TS+C (L) approach. Namely, the on-time percentage jumps from about 60% in the Myopic case to just under 77% in the TS+C (PD) approaches. This is coupled with a noticeable decrease in the percentage of outsourced pallets from 37% to about 21%. Thus, we see that, overall, the ADP

approaches manage to deliver more pallets on time (and sometimes even more late, especially in the TS+C (L) and the TS+C (PD Var) cases), while avoiding the capacity traps that cause pallets to be lost to outsourcing. We depict a 3D visualization of the final (learned) lookup table for the instance 6333-T-10 with TS+C (PD Fix) in Figure 6. The figure shows that, generally speaking, states with the highest reward values (darker) are concentrated more in the area of the graph with high total slack and low congestion values which is to be expected. Therefore, choosing decision vectors that result in higher total system slack (e.g., by moving pallets closer to their destinations) and lower congestion measures (e.g., making use of alts to balance the distribution of pallets) are likely to yield higher rewards, and heuristics that attempt to find such decision vectors but that are more efficient and less complex than an ADP algorithm may possibly be developed along these lines.

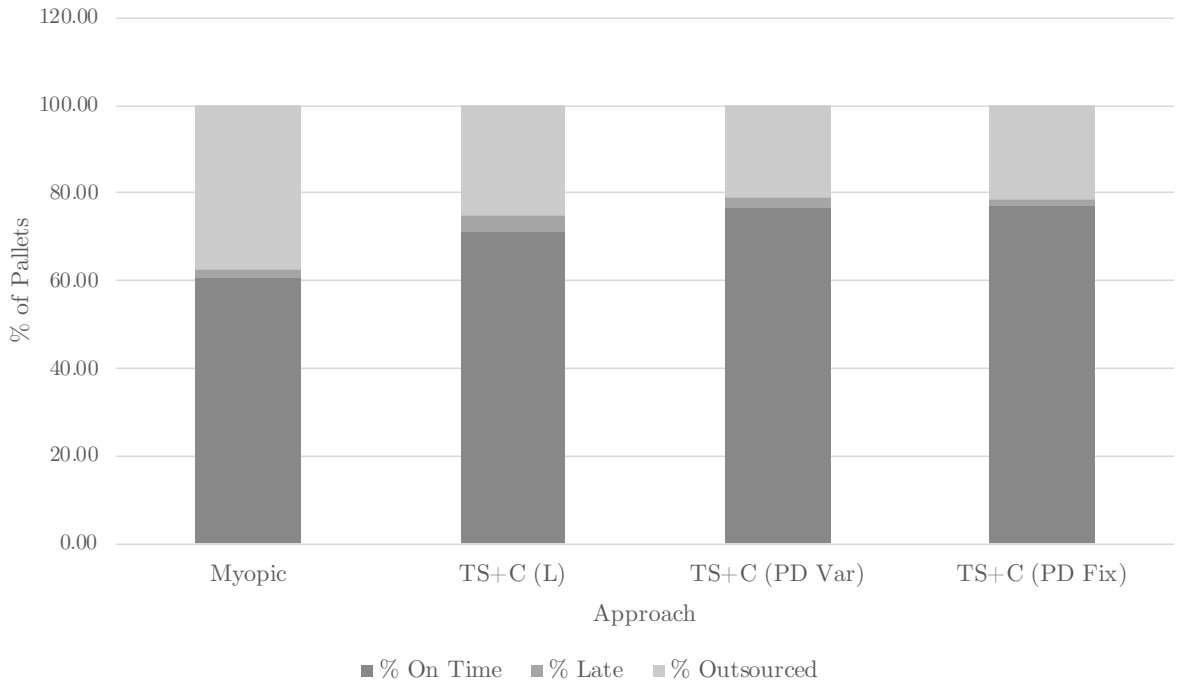


Figure 5: Performance metrics

Comparing the total runtimes (in hours) of these approaches in Figure 7, we immediately observe from the first figure that, for the first ten instances, Slacks takes considerably more time to run while the other approaches have more comparable runtimes. On average, the three TS+C approaches offer almost a factor six speedup in runtime over Slacks, and, at least in the case of TS+C (PD), similar (if not better) results. This gap in runtime is simply due to the Slacks approach having a much larger superset size with many more PDS vectors enumerated compared to the other three approaches. For example, in the case of the 10433-U-5 instance, the Slacks approach enumerated, on average, supersets with about 261,751 PDS vectors per time period. This is in contrast with the 24,610 and the 4,856 vectors per time period enumerated by the TS+C (L) and TS+C (PD Fixed) approaches, respectively. From the second figure, we can see more clearly that the two TS+C (PD) variants consistently outperform the TS+C (L) in terms of runtime. Indeed, on average, the TS+C (PD) approaches are twice as fast as TS+C (L).

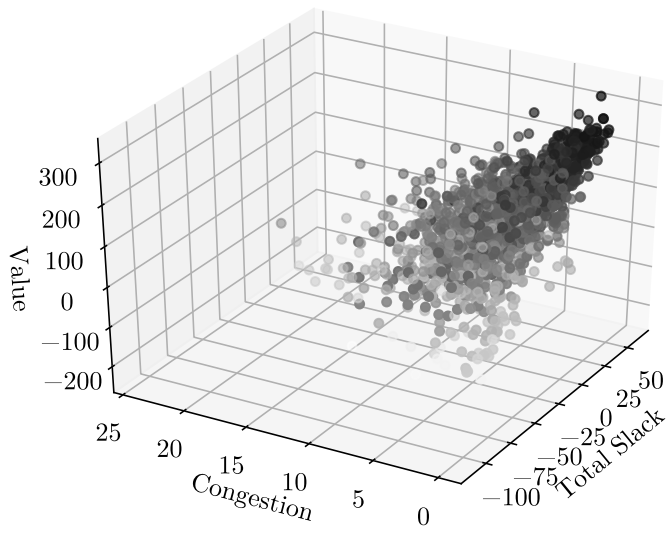


Figure 6: Visualization of final lookup table for instance 6333-T-10 with TS+C (PD Fix)

These experiments demonstrate that despite the original post-decision state vector being high-dimensional, a much simpler two-dimensional vector such as TS+C (PD) can produce a policy of good quality and a more efficient approach. Finally, we point out that these reward-to-go elements can also be used to augment (or design) other heuristics for this problem. For instance, in rolling horizon solution approaches in which only short horizons can be used at a time (because of the size of the instances to be solved), a reward-to-go approximation can help ensure that the decisions made in the short horizons are less myopic.

5.3 Effectiveness of PDS-IP-Bounding Solution Approach

In this section, we demonstrate the effectiveness of the proposed PDS-IP-Bounding solution approach presented in Section 4.3. Our ADP approach involves enumerating the vectors of a superset, $\mathcal{P}_t$, which may be very large, and consequently affects the size and solution times of the corresponding Extended IPs. The PDS-IP-Bounding solution approach provides a mechanism for working around this limitation by solving a number of small PDS IPs instead of a monolithic Extended IP.

For this comparison, at each time period, we solve the encountered decision epoch subproblem first as a monolithic Extended IP (using Gurobi), then we resolve the subproblem using the PDS-IP-Bounding algorithm and use this latter solution to transition to the next state. We note that all recorded times include any setup/overhead required by either of the two approaches, i.e., after enumerating the superset, we record the full time it takes either approach to choose a decision vector. In Tables 3-5, we report the following statistics (which are averaged over *all* decision epoch subproblems):

- **Ext. Time**: The average runtime (in seconds) of solving the decision epoch subproblems

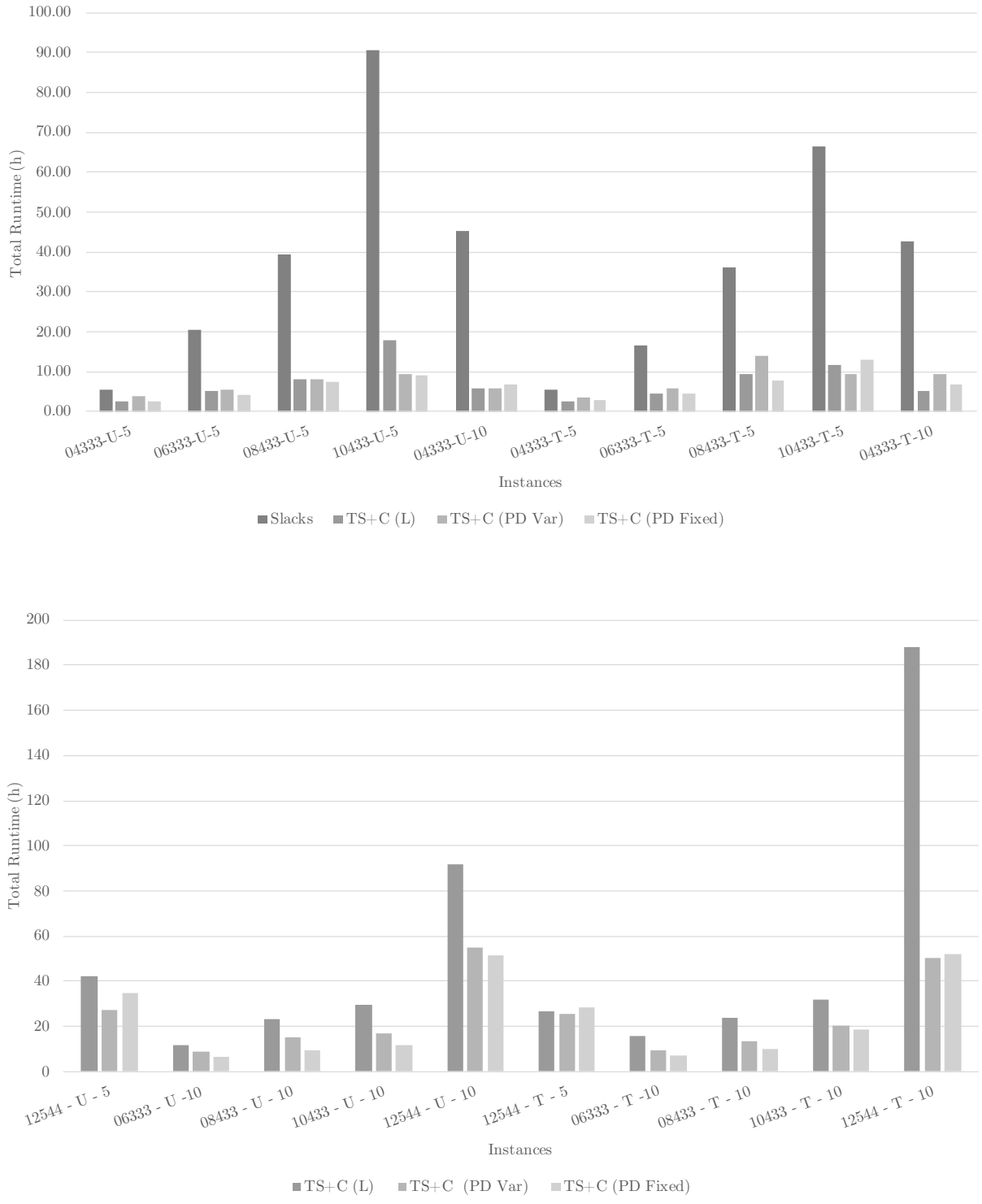


Figure 7: ADP algorithm runtimes for aggregation approaches

using the *Extended IP* approach.

- **PDS-IP Time**: The average runtime (in seconds) of solving the decision epoch subproblems using the *PDS-IP-Bounding* approach.
- **% PDSs Discarded**: The average percentage of PDS vectors in the superset $\mathcal{P}_t$ that were discarded during the solution process in the PDS-IP-Bounding approach (due to bounding).
- **# Enumerated Vectors**: The average number of enumerated PDS vectors, i.e. the average superset size.
- **Speedup Factor (SF)**: The speedup factor PDS-IP-Bounding provides over Extended IP, i.e. a speedup factor greater than one indicates that PDS-IP-Bounding is faster than Extended IP, and vice versa.

To keep running times reasonable, a time limit of 900s was imposed on the Extended IPs and a time limit of 120s on each individual PDS IP, as there were some cases in which the solver was unable to close the gap causing the branch-and-bound tree to exhaust the system's memory. We further note that because of the increase in runtime due to the inclusion of the Extended IP solution approach, a few more instances of the Slacks aggregation scheme were excluded from this experiment. We also include separate averages for the six instances for which we present results (Avg. SF (Slacks)).

From Table 3, we observe that PDS-IP-Bounding provides a very significant speedup in runtime in the six Slacks instances. Therefore, although the superset size is fairly large in the case of Slacks (see Table 5), which consequently causes the Extended IP to be much larger in size, the PDS-IP-Bounding algorithm is able to sidestep the issue and we are able to solve the subproblems more efficiently. We also see that PDS-IP-Bounding provides a decent speedup in runtime for the TS+C (L) approach in those same six instances. On the other hand, the Extended IP algorithm turns out to be faster in the two TS+C (PD) approaches. Similarly, for the averages over all 20 instances, we see that PDS-IP-Bounding provides close to a two-factor speedup in runtime when using TS+C (L), while becoming slightly more competitive in the case of TS+C (PD).

To further analyze these results, Table 4 shows the percentage of enumerated PDS vectors that were discarded during the solution process. On average, we observe that PDS-IP-Bounding only solves the PDS IPs corresponding to a very small percentage (less than 3%) of the superset PDS vectors with the vast majority being discarded using its bounding mechanism. This is true even in cases where Extended IP outperforms PDS-IP-Bounding in terms of runtime. While a majority of the time in the Extended IP approach (roughly 70-80%) is spent loading the IP due to the large superset size, the PDS-IP-Bounding approach is able to circumvent this issue and find provably optimal solutions much quicker in the cases of Slacks and TS+C (L). As the % PDS Discarded values are very high, it may be possible to implement more intelligent enumeration schemes of the superset, thereby reducing these values. Regardless, the findings here suggest that the PDS-IP-Bounding approach is generally less sensitive to the size of the superset compared to solving an Extended IP. Clearly, the combination of efficient enumeration of vectors in the superset (irrespective of their feasibility), and using the PDS-IP-Bounding approach, provides an efficient solution approach, and helps sidestep the two issues of determining the feasibility of enumerated PDS vectors (by relying on the IP) and that of the size of the resulting IP (by relying on PDS-IP-Bounding).

Table 3: Extended IP vs. PDS-IP-Bounding runtime comparison (overall)

Instance	Slacks			TS+C (L)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	1.30	0.50	2.59	0.19	0.17	1.11
06333 - U - 5	7.81	1.95	4.01	0.44	0.29	1.53
08433 - U - 5	15.56	2.89	5.38	0.62	0.43	1.44
10433 - U - 5	-	-	-	1.28	1.03	1.24
04333 - U - 10	-	-	-	0.65	0.31	2.09
04333 - T - 5	1.20	0.45	2.64	0.17	0.19	0.94
06333 - T - 5	9.16	1.14	8.03	0.35	0.29	1.21
08433 - T - 5	14.54	2.42	6.01	0.69	0.60	1.16
10433 - T - 5	-	-	-	0.97	0.55	1.78
04333 - T - 10	-	-	-	0.56	0.29	1.93
12544 - U - 5	-	-	-	2.59	2.51	1.03
06333 - U - 10	-	-	-	1.53	0.52	2.93
08433 - U - 10	-	-	-	2.66	1.01	2.64
10433 - U - 10	-	-	-	4.29	1.04	4.11
12544 - U - 10	-	-	-	10.61	5.78	1.83
12544 - T - 5	-	-	-	2.27	1.00	2.28
06333 - T - 10	-	-	-	1.70	0.78	2.19
08433 - T - 10	-	-	-	2.82	0.89	3.16
10433 - T - 10	-	-	-	3.87	1.29	2.99
12544 - T - 10	-	-	-	9.63	16.83	0.57
Avg. SF (Slacks)	-	-	4.78	-	-	1.23
Avg. SF	-	-	-	-	-	1.91

Instance	TS+C (PD Var)			TS+C (PD Fix)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	0.13	0.40	0.33	0.11	0.19	0.57
06333 - U - 5	0.23	0.47	0.48	0.19	0.31	0.63
08433 - U - 5	0.26	0.66	0.40	0.25	0.60	0.41
10433 - U - 5	0.28	0.86	0.32	0.30	0.80	0.38
04333 - U - 10	0.34	0.56	0.61	0.56	0.55	1.03
04333 - T - 5	0.11	0.33	0.35	0.09	0.23	0.41
06333 - T - 5	0.23	0.51	0.46	0.18	0.36	0.49
08433 - T - 5	0.29	0.84	0.34	0.43	1.02	0.42
10433 - T - 5	0.29	0.84	0.34	0.43	1.02	0.42
04333 - T - 10	0.35	1.10	0.32	0.24	0.78	0.31
12544 - U - 5	0.56	2.65	0.21	0.85	3.23	0.26
06333 - U - 10	0.66	0.67	0.99	0.51	0.47	1.09
08433 - U - 10	0.70	1.42	0.49	0.51	0.84	0.61
10433 - U - 10	0.92	1.55	0.59	0.80	0.95	0.84
12544 - U - 10	1.82	5.67	0.32	3.03	4.39	0.69
12544 - T - 5	0.70	1.92	0.37	0.88	2.13	0.41
06333 - T - 10	0.66	0.86	0.77	0.49	0.60	0.82
08433 - T - 10	0.69	1.20	0.57	0.63	0.80	0.79
10433 - T - 10	1.29	1.67	0.77	1.41	1.37	1.03
12544 - T - 10	1.85	4.77	0.39	2.66	5.07	0.52
Avg. SF (Slacks)	-	-	0.39	-	-	0.49
Avg. SF	-	-	0.47	-	-	0.61

Table 4: PDS-IP-Bounding percentage of discarded vectors (overall)

Instances	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	97.47	97.79	97.68	97.60
06333 - U - 5	96.60	98.01	97.76	97.81
08433 - U - 5	97.68	98.86	98.54	98.30
10433 - U - 5	-	98.52	98.45	98.51
04333 - U - 10	-	98.45	98.06	98.05
04333 - T - 5	97.82	98.35	97.71	97.63
06333 - T - 5	98.32	98.12	98.30	98.26
08433 - T - 5	98.01	98.45	98.36	98.14
10433 - T - 5	-	98.58	98.97	99.33
04333 - T - 10	-	98.34	98.19	97.96
12544 - U - 5	-	98.50	99.41	99.50
06333 - U - 10	-	98.53	98.11	98.17
08433 - U - 10	-	98.74	98.52	98.43
10433 - U - 10	-	99.52	99.14	98.88
12544 - U - 10	-	98.09	98.67	97.97
12544 - T - 5	-	99.46	99.17	99.34
06333 - T - 10	-	98.17	98.15	97.99
08433 - T - 10	-	99.56	99.32	98.98
10433 - T - 10	-	99.15	98.97	98.76
12544 - T - 10	-	98.24	98.77	98.50
Avg. % PDS Discarded (Slacks)	97.65	98.27	98.06	97.96
Avg. % PDS Discarded	-	98.57	98.51	98.41

Table 5: Superset sizes

Instance	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	18,298.08	3,832.79	1,650.30	1,089.20
06333 - U - 5	52,568.09	9,280.59	3,092.28	2,428.59
08433 - U - 5	145,641.56	14,292.49	3,104.20	2,717.38
10433 - U - 5	-	24,610.89	4,861.84	4,856.81
04333 - U - 10	-	15,376.80	5,716.83	10,184.19
04333 - T - 5	17,023.97	3,586.59	1,368.11	939.75
06333 - T - 5	69,291.93	8,549.13	2,615.89	1,749.49
08433 - T - 5	155,652.82	16,256.07	3,912.76	3,147.67
10433 - T - 5	-	4,216.98	4,475.94	23,972.14
04333 - T - 10	-	14,057.39	5,805.88	3,733.76
12544 - U - 5	-	56,529.30	7,440.34	10,744.64
06333 - U - 10	-	34,416.72	11,219.25	8,428.02
08433 - U - 10	-	60,253.35	11,446.49	10,226.44
10433 - U - 10	-	103,725.49	16,656.76	17,205.76
12544 - U - 10	-	208,908.03	28,215.89	42,759.37
12544 - T - 5	-	56,670.83	6,937.40	10,581.60
06333 - T - 10	-	40,167.59	12,880.83	9,608.72
08433 - T - 10	-	66,705.42	12,017.51	11,117.27
10433 - T - 10	-	101,300.79	21,694.69	22,916.36
12544 - T - 10	-	215,092.77	29,094.07	48,760.97
Avg. # Enumerated Vectors (Slacks)	76,412.74	9,299.61	2,623.92	2,012.01
Avg. # Enumerated Vectors	-	52,891.50	9,710.36	12,358.41

Curiously, while the percentage of discarded PDS remains high in the two TS+C (PD) approaches, they do not exhibit the same speedup in runtime when we use PDS-IP-Bounding. However, further investigation suggested that this is a consequence of the scaling and not the congestion measure itself; for two of the smaller instances, we were able to use a non-scaled version of TS+C (PD) and the results showed that PDS-IP-Bounding offers a speedup in runtime as is the case with Slacks and TS+C (L) – although its reward value were notably worse than the scaled versions. There are two reasons why scaling/partitioning may have this impact. The first is that scaling reduces the number of PDS vectors which enables a much smaller Extended IP, thereby reducing its runtime. The second is that instead of fixing an individual PDS vector for each PDS IP, we now fix a scaled vector, e.g. the endpoint of the partition, and the search space for the PDS IP now involves finding a feasible decision vector whose resulting PDS vector maps to the scaled (fixed) vector. This may explain why we see more extreme outliers in terms of individual PDS-IP-Bounding runtimes. That is, due to the different nature of the PDS IPs, the solver either requires more time to find an optimal solution, or exhausts the time limit trying to close the optimality gap. However, removing these few extreme outliers, we can see a much better picture for PDS-IP-Bounding. For example, for the 12544-T-10 instance with TS+C (PD Fix), we show in Figure 8 box plots comparing the individual runtimes for each time period after removing outliers, and we observe that the quartile runtimes for PDS-IP-Bounding are noticeably lower than those of Extended IP. This suggests that PDS-IP-Bounding is still offering advantages despite the average time period runtime being distorted by the presence of these outliers. It also suggests that if sufficient computational power is available, running both approaches in parallel for each decision subproblem and choosing the solution found by the first approach to terminate might be a reasonable strategy for further efficiency gains.

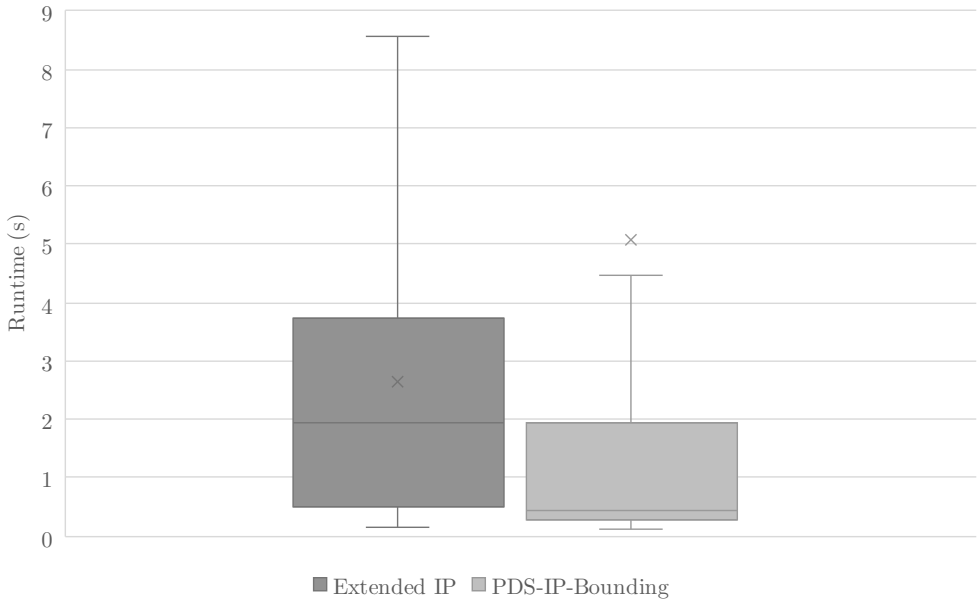


Figure 8: Box plot for individual time period runtimes for instance 12544-T-10 with TS+C (PD Fix)

There is also a noticeable difference between the overall average runtimes (which include training

and testing) and their testing counterparts. In Tables 6 and 7, we show the runtime values and the additional statistics taken only over the time periods of the 200 testing iterations. A higher percentage of PDS IPs is discarded in the testing phase and, as a result, the speedup factors are noticeably higher. One reason for these results is the presence of exploration decision epoch subproblems in training. Recall that exploration decision epoch subproblems involve replacing the lookup table values with randomly generated coefficients in the range of values of the current lookup table (specifically, in the range of values of visited states, which are most likely nonzero values). This is in contrast with their exploitation counterparts which typically involve a small number of PDSs with a nonzero lookup table value. As a consequence of these artificial coefficients and the change in the number and distribution of nonzero PDS values, we observed a slowdown in the runtimes of exploration subproblems compared with their exploitation counterparts. This can be seen in Figure 9 where we show the average runtimes for exploration and exploitation subproblems for each of the two approaches. For brevity, we only show the results for the TS+C (PD Fix) aggregation approach.

A second reason is that the first few training iterations involve a lookup table in which very few states (if any) have been visited, and therefore, almost all enumerated PDS vectors in these subproblems will have a value of zero, making it harder for the algorithm's sorting and bounding mechanisms to be effective in those early iterations. An illustration of the effect of this warm-up phenomenon on the runtimes of the PDS-IP-Bounding approach can be seen in Figure 10, where we plot the average runtime of the first 20 AVI training iterations (i.e., each point is an average of the 20 subproblems comprising that AVI iteration) for the instance 12544-T-10 (with TS+C (PD Fix)). By the 14th training iteration, this effect is diminished and PDS-IP-Bounding performs on a similar level as Extended IP. It is possible to overcome this issue (and make the algorithm even more efficient) by checking if none of the enumerated post-decision states have been previously visited (and thus, have an initial value of zero). In such cases, the subproblem simply reduces to optimizing the one-period reward and the algorithm does not have to go through the list of enumerated PDS vectors as they become irrelevant, thereby drastically reducing the runtime in such iterations.

5.4 Effectiveness of Exploration Scheme

In this section, we demonstrate the value of including some exploration in the training phase of the ADP algorithm. While we adopt a standard ϵ-greedy algorithm, we adapt the standard mechanism of choosing a random decision vector to accommodate our IP setting as described in Section 4.1. In Table 8, we show that this mechanism with $\epsilon = 0.15$ yields benefits by providing a noticeable 3.5-fold improvement in the reward values compared to having no exploration at all, with the highest improvements for the 12544-U-5 and 12544-T-5 instances. Once again, for brevity, we include only the results for the TS+C (PD) Fix approach.

Naturally, exploration is vital in our instance settings because it enables the discovery of the less-capacitated (albeit longer paths) which consequently results in improved policies with higher rewards.

Table 6: Extended IP vs. PDS-IP-Bounding runtime comparison (testing)

Instance	Slacks			TS+C (L)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	1.46	0.22	6.49	0.17	0.11	1.46
06333 - U - 5	10.79	0.41	26.62	0.47	0.22	2.12
08433 - U - 5	14.74	1.44	10.27	0.54	0.17	3.10
10433 - U - 5	-	-	-	1.34	0.33	4.08
04333 - U - 10	-	-	-	0.67	0.26	2.59
04333 - T - 5	1.29	0.18	7.10	0.17	0.14	1.20
06333 - T - 5	11.08	0.48	22.99	0.32	0.12	2.72
08433 - T - 5	12.65	0.54	23.26	0.68	0.36	1.88
10433 - T - 5	-	-		0.93	0.27	3.51
04333 - T - 10	-	-		0.52	0.16	3.25
12544 - U - 5	-	-		2.20	0.28	7.91
06333 - U - 10	-	-		1.38	0.16	8.44
08433 - U - 10	-	-		2.80	0.31	9.16
10433 - U - 10	-	-		6.05	0.49	12.42
12544 - U - 10	-	-		11.36	1.25	9.06
12544 - T - 5	-	-		2.23	0.43	5.24
06333 - T - 10	-	-		1.81	0.17	10.51
08433 - T - 10	-	-		2.47	0.17	14.16
10433 - T - 10	-	-		3.39	0.34	9.91
12544 - T - 10	-	-		11.54	0.79	14.68
Avg. SF (Slacks)	-	-	16.12	-	-	2.08
Avg. SF	-	-	-	-	-	6.37

Instance	TS+C (PD Var)			TS+C (PD Fix)		
	Ext. Time (s)	PDS-IP Time (s)	SF	Ext. Time (s)	PDS-IP Time (s)	SF
04333 - U - 5	0.13	0.51	0.25	0.10	0.11	0.93
06333 - U - 5	0.21	0.37	0.59	0.20	0.16	1.19
08433 - U - 5	0.29	0.44	0.66	0.28	0.58	0.48
10433 - U - 5	0.27	0.30	0.92	0.30	0.77	0.39
04333 - U - 10	0.34	0.35	0.97	0.45	0.27	1.63
04333 - T - 5	0.12	0.24	0.49	0.09	0.15	0.59
06333 - T - 5	0.24	0.28	0.84	0.18	0.22	0.82
08433 - T - 5	0.25	0.45	0.56	0.40	0.66	0.60
10433 - T - 5	0.25	0.45	0.56	0.40	0.66	0.60
04333 - T - 10	0.35	1.25	0.28	0.19	0.47	0.39
12544 - U - 5	0.52	4.61	0.11	0.83	0.77	1.08
06333 - U - 10	0.70	0.36	1.96	0.51	0.32	1.60
08433 - U - 10	0.83	0.96	0.87	0.52	0.52	1.00
10433 - U - 10	0.71	0.73	0.97	0.70	0.57	1.23
12544 - U - 10	1.65	5.78	0.29	2.11	3.39	0.62
12544 - T - 5	0.79	1.01	0.78	0.98	1.57	0.62
06333 - T - 10	0.85	0.83	1.02	0.46	0.46	0.99
08433 - T - 10	0.61	0.39	1.55	0.65	0.37	1.75
10433 - T - 10	1.28	0.97	1.33	1.25	0.57	2.19
12544 - T - 10	1.69	3.39	0.50	3.14	3.46	0.91
Avg. SF (Slacks)	-	-	0.57	-	-	0.77
Avg. SF	-	-	0.78	-	-	0.98

Table 7: PDS-IP-Bounding percentage of discarded vectors (testing)

Instances	Slacks	TS+C (L)	TS+C (PD Var)	TS+C (PD Fix)
04333 - U - 5	99.29	99.20	99.46	99.38
06333 - U - 5	99.49	99.68	99.65	99.57
08433 - U - 5	99.26	99.91	99.89	99.83
10433 - U - 5	-	99.95	99.84	99.86
04333 - U - 10	-	99.59	99.68	99.64
04333 - T - 5	99.63	99.54	99.44	99.35
06333 - T - 5	99.68	99.90	99.85	99.77
08433 - T - 5	99.55	99.91	99.84	99.72
10433 - T - 5	-	99.90	99.94	99.96
04333 - T - 10	-	99.62	99.82	99.79
12544 - U - 5	-	99.98	99.95	99.95
06333 - U - 10	-	99.86	99.89	99.85
08433 - U - 10	-	99.96	99.95	99.92
10433 - U - 10	-	99.98	99.95	99.95
12544 - U - 10	-	99.98	99.89	99.98
12544 - T - 5	-	99.98	99.94	99.94
06333 - T - 10	-	99.94	99.88	99.78
08433 - T - 10	-	99.98	99.97	99.97
10433 - T - 10	-	99.98	99.97	99.97
12544 - T - 10	-	99.99	99.99	99.97
Avg. % PDS Discarded (Slacks)	99.48	99.69	99.69	99.60
Avg. % PDS Discarded	-	99.84	99.84	99.81

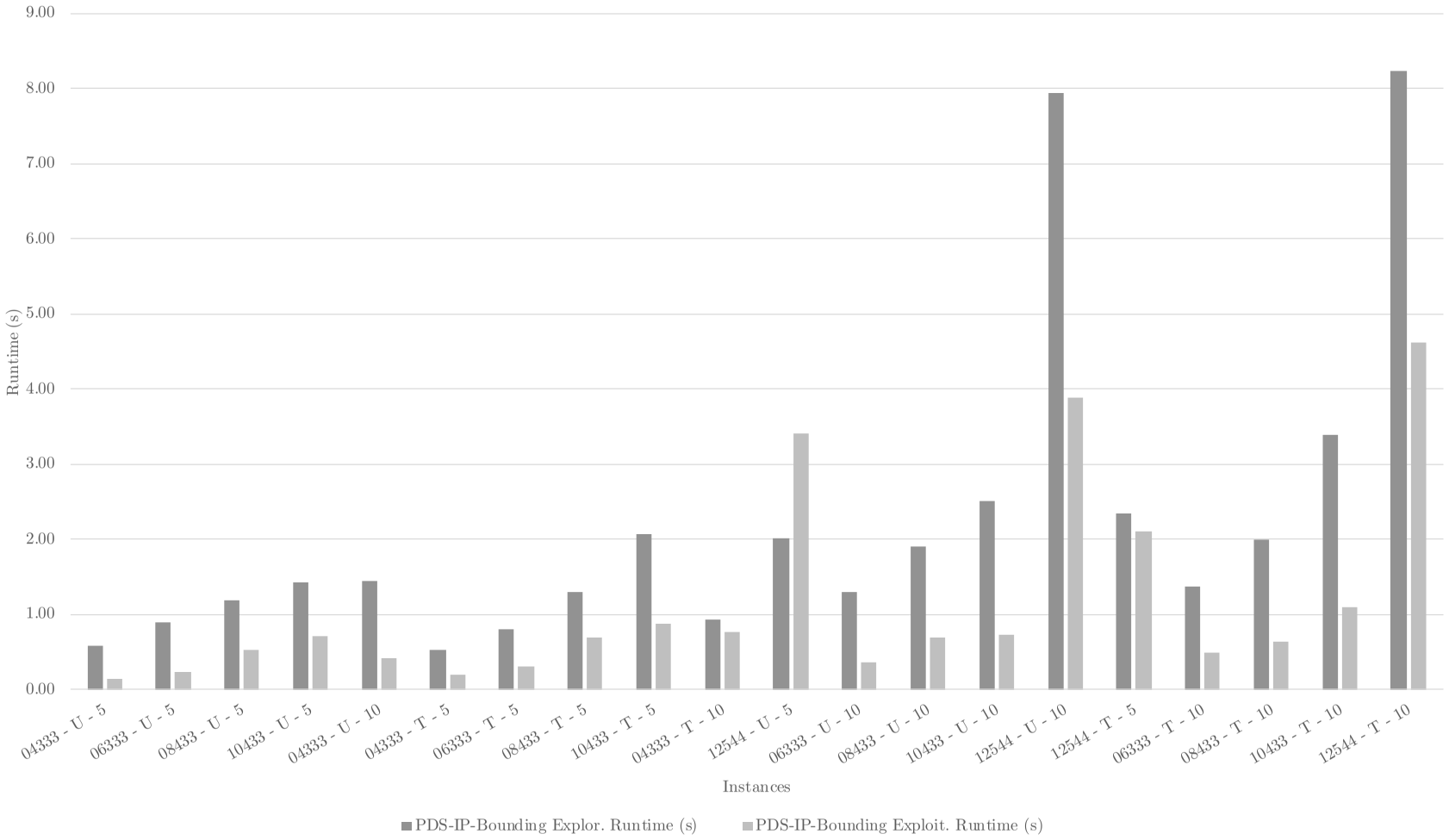


Figure 9: Comparison of exploration and exploitation subproblem runtimes

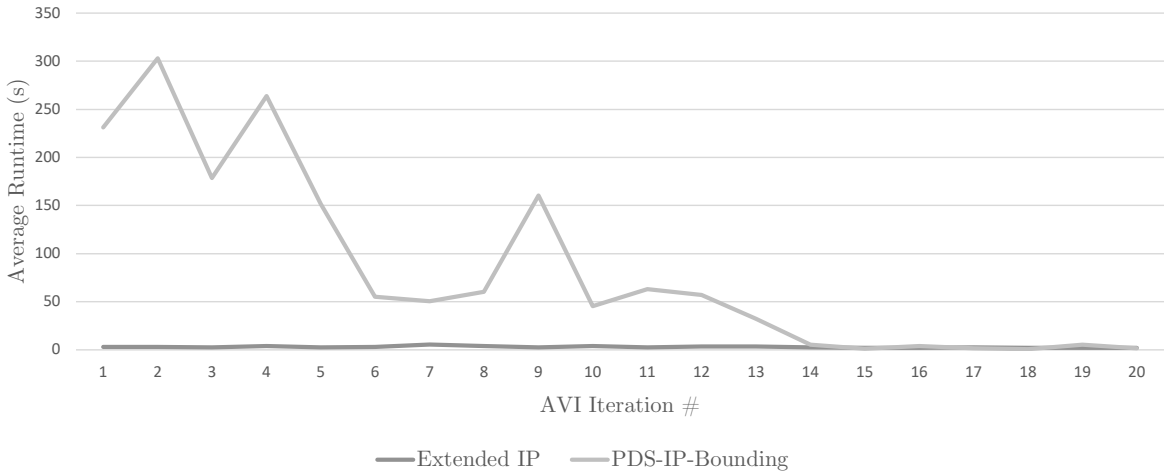


Figure 10: Training warm-up impact on runtime of PDS-IP-Bounding (12544-T-10 with TS+C (PD Fix))

Table 8: The value of exploration

Instance	Value w/ Exploration	Value w/o Exploration	Improvement Factor
04333 - U - 5	98.62	61.08	1.61
06333 - U - 5	74.45	47.67	1.56
08433 - U - 5	244.15	106.54	2.29
10433 - U - 5	197.45	61.37	3.22
04333 - U - 10	174.91	97.36	1.80
04333 - T - 5	108.91	66.24	1.64
06333 - T - 5	163.84	62.82	2.61
08433 - T - 5	228.40	100.82	2.27
10433 - T - 5	267.86	187.80	1.43
04333 - T - 10	178.52	90.67	1.97
12544 - U - 5	311.79	17.36	17.96
06333 - U - 10	162.76	32.74	4.97
08433 - U - 10	408.97	150.05	2.73
10433 - U - 10	508.77	258.90	1.97
12544 - U - 10	648.14	133.10	4.87
12544 - T - 5	340.61	50.46	6.75
06333 - T - 10	178.36	64.12	2.78
08433 - T - 10	458.35	217.45	2.11
10433 - T - 10	585.99	244.28	2.40
12544 - T - 10	863.36	280.16	3.08
Average	310.21	116.55	3.50

6 Conclusions & Future Work

In this paper, we introduced and formulated the Dynamic Freight Routing Problem (DFRP) for LTL carriers which models the daily routing of shipments from origins to destinations in the presence of demand uncertainty. We modeled this problem as a Markov Decision Process (MDP), and presented an Approximate Dynamic Programming solution approach to overcome the MDP's curses of dimensionality. Our approach relies on using a lookup table to store value function approximations, and we compared a number of aggregation approaches and demonstrate that a two-dimensional aggregation scheme is able to produce policies that are much better than a standard myopic policy. We have also introduced a framework for integrating the lookup table architecture into the IP decision epoch subproblems that need to be solved at every time period in the ADP algorithm. This framework includes: (1) the modeling of these integrated/extended IPs, (2) a solution approach which exploits their inherent structure and decomposes the problem into many small post-decision state (PDS) IPs, and (3) an adaptation of the standard ϵ-greedy algorithm to the IP setting. Our computational experiments show that there are computational advantages to be gained by prioritizing solving the most attractive PDS IPs and using dynamic bounds.

Future work might compare using other forms of value function approximations to using a lookup table approach for this class of problems. For example, linear value function approximations are commonly used in IP settings since they are easy to integrate, and fairly flexible. In addition, ReLU neural nets are becoming increasingly popular for approximating value functions and can be integrated into ADP approaches which involve IP decision epoch subproblems. How these different value function approximations compare in terms of policy quality and runtimes for this problem class is a question worth addressing. Other research directions may include using insights from this work to design efficient algorithms for solving large-scale and more realistic instances of this class of problems. As we only consider somewhat of an idealized setting, solving real-life instances may pose additional challenges and/or practical considerations that were not considered in this work (e.g. incorporating daily demand forecasts, considering detailed terminal operations such as *sorts*, adding a layer of decisions that relates to dynamic changes in trailer schedules to allow for more flexibility and cost savings, etc.).

Acknowledgement

Ahmad Baubaid would like to acknowledge financial support from the King Fahd University of Petroleum & Minerals.

References

Alumur S, Kara BY, 2008 *Network hub location problems: The state of the art. European Journal of Operational Research* 190(1):1–21, URL `http://dx.doi.org/10.1016/j.ejor.2007.06.008`.

Baubaid A, Boland N, Savelsbergh M, 2020 *The value of limited flexibility in service network designs. Transportation Science* URL `http://dx.doi.org/10.1287/trsc.2020.1009`.

Campbell JF, 2009 *Hub location for time definite transportation. Computers & Operations Research* 36(12):3107–3116, URL `http://dx.doi.org/10.1016/j.cor.2009.01.009`.

Campbell JF, O'Kelly ME, 2012 *Twenty-five years of hub location research. Transportation Science* 46(2):153–169, URL `http://dx.doi.org/10.1287/trsc.1120.0410`.

Cheung R, Muralidharan B, 1999 *Impact of dynamic decision making on hub-and-spoke freight transportation networks. Annals of Operations Research* 87:49–71, URL http://dx.doi.org/10.1023/A:1018909825336.

Cheung RK, Muralidharan B, 2000 *Dynamic routing for priority shipments in LTL service networks. Transportation Science* 34(1):86–98, URL http://dx.doi.org/10.1287/trsc.34.1.86.12279.

Crainic TG, 2000 *Service network design in freight transportation. European Journal of Operational Research* 122(2):272–288, URL http://dx.doi.org/10.1016/S0377-2217(99)00233-7.

Cunha CB, Silva MR, 2007 *A genetic algorithm for the problem of configuring a hub-and-spoke network for a LTL trucking company in brazil. European Journal of Operational Research* 179(3):747–758, URL http://dx.doi.org/10.1016/j.ejor.2005.03.057.

Dall'Orto LC, Crainic TG, Leal JE, Powell WB, 2006 *The single-node dynamic service scheduling and dispatching problem. European Journal of Operational Research* 170(1):1–23, URL http://dx.doi.org/10.1016/j.ejor.2004.06.016.

Erera A, Hewitt M, Savelsbergh M, Zhang Y, 2013a *Improved load plan design through integer programming based local search. Transportation Science* 47(3):412–427, URL http://dx.doi.org/10.1287/trsc.1120.0441.

Erera A, Karacık B, Savelsbergh M, 2008 *A dynamic driver management scheme for less-than-truckload carriers. Computers & Operations Research* 35(11):3397–3411, URL http://dx.doi.org/10.1016/j.cor.2007.01.019.

Erera AL, Hewitt M, Savelsbergh MWP, Zhang Y, 2013b *Creating schedules and computing operating costs for LTL load plans. Computers & Operations Research* 40(3):691–702, URL http://dx.doi.org/10.1016/j.cor.2011.10.001.

Hejazi B, Haghani A, 2007 *Dynamic decision making for less-than-truckload trucking operations. Transportation Research Record: Journal of the Transportation Research Board* 2032(1):17–25, URL http://dx.doi.org/10.3141/2032-03.

Jarrah AI, Johnson E, Neubert LC, 2009 *Large-scale, less-than-truckload service network design. Operations Research* 57(3):609–625, URL http://dx.doi.org/10.1287/opre.1080.0587.

Kleywegt AJ, Papastavrou JD, 1998 *Acceptance and dispatching policies for a distribution problem. Transportation Science* 32(2):127–141, URL http://dx.doi.org/10.1287/trsc.32.2.127.

Kokonendji CC, Zocchi SS, 2010 *Extensions of discrete triangular distributions and boundary bias in kernel estimation for discrete functions. Statistics & Probability Letters* 80(21):1655–1662, URL http://dx.doi.org/10.1016/j.spl.2010.07.008.

Lin CC, Lee SC, 2018 *Hub network design problem with profit optimization for time-definite LTL freight transportation. Transportation Research Part E: Logistics and Transportation Review* 114:104–120, URL http://dx.doi.org/10.1016/j.tre.2018.03.007.

Lindsey K, Erera A, Savelsbergh M, 2016 *Improved integer programming-based neighborhood search for less-than-truckload load plan design. Transportation Science* 50(4):1360–1379, URL http://dx.doi.org/10.1287/trsc.2016.0700.

Pillac V, Gendreau M, Guéret C, Medaglia AL, 2013 *A review of dynamic vehicle routing problems. European Journal of Operational Research* 225(1):1–11, URL http://dx.doi.org/10.1016/j.ejor.2012.08.015.

Powell WB, 1986 *A local improvement heuristic for the design of less-than-truckload motor carrier networks. Transportation Science* 20(4):246–257, URL http://dx.doi.org/10.1287/trsc.20.4.246.

Powell WB, 2003 *Dynamic models of transportation operations. Handbooks in Operations Research and Management Science*, volume 11, 677–756 (Elsevier), ISBN 978-0-444-51328-1.

Powell WB, 2011 *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley Series in Probability and Statistics (Hoboken, N.J: Wiley), 2nd ed edition, ISBN 978-0-470-60445-8.

Powell WB, Jaillet P, Odoni A, 1995 *Stochastic and dynamic networks and routing. Handbooks in Operations Research and Management Science*, volume 8 of *Network Routing*, 141–295 (Elsevier), URL http://dx.doi.org/10.1016/S0927-0507(05)80107-0.

Powell WB, Koskosidis IA, 1992 *Shipment routing algorithms with tree constraints. Transportation Science* 26(3):230–245, URL http://dx.doi.org/10.1287/trsc.26.3.230.

Powell WB, Shapiro JA, Simão HP, 2002 *An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. Transportation Science* 36(2):231–249, URL http://dx.doi.org/10.1287/trsc.36.2.231.561.

Powell WB, Sheffi Y, 1983 *The load planning problem of motor carriers: Problem description and a proposed solution approach. Transportation Research Part A: General* 17(6):471–480, URL http://dx.doi.org/10.1016/0191-2607(83)90167-X.

Powell WB, Sheffi Y, 1989 *OR practice—design and implementation of an interactive optimization system for network design in the motor carrier industry. Operations Research* 37(1):12–29, URL http://dx.doi.org/10.1287/opre.37.1.12.

Powell WB, Simao HP, Bouzaiene-Ayari B, 2012 *Approximate dynamic programming in transportation and logistics: A unified framework. EURO Journal on Transportation and Logistics* 1(3):237–284, URL http://dx.doi.org/10.1007/s13676-012-0015-8.

Puterman ML, 2005 *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics (Hoboken, NJ: Wiley-Interscience), ISBN 978-0-471-72782-8.

Ridouane Y, Herszterg I, Boland N, Erera A, 2020 *Near real-time loadplan adjustments for less-than-truckload carriers. Optimization Online* .

Riis JO, 1965 *Discounted markov programming in a periodic process. Operations Research* 13(6):920–929, URL http://dx.doi.org/10.1287/opre.13.6.920.

Ritzinger U, Puchinger J, Hartl RF, 2016 *A survey on dynamic and stochastic vehicle routing problems. International Journal of Production Research* 54(1):215–231, URL http://dx.doi.org/10.1080/00207543.2015.1043403.

Shi N, Cheung RK, Xu H, Lai KK, 2011 *An adaptive routing strategy for freight transportation networks. Journal of the Operational Research Society* 62(4):799–805, URL http://dx.doi.org/10.1057/jors.2010.7.

Simao HP, Powell WB, 2019 *Decomposition methods for dynamic load planning and driver management in LTL trucking. 7th INFORMS Transportation Science and Logistics Society Workshop* (Vienna).

Topaloglu H, Powell WB, 2006 *Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. INFORMS Journal on Computing* 18(1):31–42, URL http://dx.doi.org/10.1287/ijoc.1040.0079.

Toriello A, Nemhauser G, Savelsbergh M, 2010 *Decomposing inventory routing problems with approximate value functions. Naval Research Logistics (NRL)* 57(8):718–727, URL http://dx.doi.org/10.1002/nav.20433.

Ulmer MW, Mattfeld DC, Köster F, 2017 *Budgeting time for dynamic vehicle routing with stochastic customer requests. Transportation Science* 52(1):20–37, URL http://dx.doi.org/10.1287/trsc.2016.0719.

Ulmer MW, Thomas BW, 2019 *Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. European Journal of Operational Research* 285(1):183–195, URL http://dx.doi.org/10.1016/j.ejor.2019.04.029.

van Heeswijk W, Mes M, Schutten M, 2015 *An approximate dynamic programming approach to urban freight distribution with batch arrivals*. Corman F, Voß S, Negenborn RR, eds., *Computational Logistics*, volume 9335 of *Lecture Notes in Computer Science*, 61–75 (Cham: Springer International Publishing), ISBN 978-3-319-24264-4, URL http://dx.doi.org/10.1007/978-3-319-24264-4_5.

van Heeswijk WJA, Mes MRK, Schutten JMJ, 2019 *The delivery dispatching problem with time windows for urban consolidation centers. Transportation Science* 53(1):203–221, URL http://dx.doi.org/10.1287/trsc.2017.0773.

Veugen LMM, van der Wal J, Wessels J, 1983 *The numerical exploitation of periodicity in markov decision processes. Operations-Research-Spektrum* 5(2):97–103, URL http://dx.doi.org/10.1007/BF01720020.

Wieberneit N, 2008 *Service network design for freight transportation: A review. OR Spectrum* 30(1):77–112, URL http://dx.doi.org/10.1007/s00291-007-0079-2.