1.5em

Stochastic optimization approaches for dynamic matching in crowdsourced delivery considering matching fairness and driver preference

by

Sahil Bhatt
Bachelors, Toronto Metropolitan University, 2022

A dissertation
presented to Toronto Metropolitan University
in partial fulfillment of
the requirements for the degree of
Master of Applied Science (MASc)
in the Program of
Mechanical, Industrial and Mechatronics Engineering

Toronto, Ontario, Canada, 2024

Author's Declaration For Electronic Submission Of A Dissertation

I hereby declare that I am the sole author of this dissertation. This is a true copy of the dissertation, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this dissertation to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

STOCHASTIC OPTIMIZATION APPROACHES FOR DYNAMIC MATCHING IN CROWDSOURCED DELIVERY CONSIDERING MATCHING FAIRNESS AND DRIVER PREFERENCE

Sahil Bhatt
Master of Applied Science (MASc)
Mechanical, Industrial and Mechatronics Engineering
Toronto Metropolitan University, 2024

Abstract

Crowdsourced delivery systems face significant operational challenges due to the inherent randomness in customer demand and service capacity. The randomness in service capacity is largely attributed to self-scheduling drivers, whose availability is unknown in advance. In this research, we investigate the crowdsourced delivery dynamic matching problem considering driver matching fairness. We develop a Markov Decision Process (MDP) model to sequentially select matching and heatmap decisions, which guide the repositioning of crowdsourced drivers. The MDP model consists of two interdependent processes: the selection of an optimal vector of driver pool sizes and the matching algorithm. Due to the endogenous uncertainty introduced by the selection of the driver pool sizes and the curse of dimensionality, we employ a rolling-horizon-based stochastic look-ahead policy to solve the matching problem. The matching problem is formulated as a two-stage stochastic integer program for matching and repositioning decisions. The matching algorithm's results are used to update the value function associated with the driver pool sizes. Boltzmann Exploration is then used to select a new pool size vector, which is input to the routing algorithm in an iterative process. This iterative approach aims to determine the best driver pool size to balance driver compensation standards and service level. We aim to provide insight into optimizing crowdsourced delivery while addressing the challenges posed by the inherent randomness in the system.

Acknowledgments

acknowledgements

Dedication

Contents

1 Introduction **1**

2 Literature Review **2**

2.1 Operational Challenges and Optimization 2

2.1.1 Vehicle Routing and Pickup and Delivery Problem (Static and Dynamic) . . . 2

2.1.2 Hybrid Workforce Scheduling 3

2.1.3 Self-scheduling Workforce 4

2.1.4 Additional Operational Challenges 4

2.2 Fair Task Allocation and Compensation in the Sharing Economy 5

2.2.1 Fair Task Allocation 5

2.2.2 Compensation Guarantees 5

2.3 Conclusion 6

3 Mathematical Model **8**

3.0.1 Problem Description 8

3.0.2 Driver fleet size selection problem with an embedded Markov decision process 9

3.0.3 Markov Decision Process (MDP) Model 10

4 Solution Method: Value Function Approximation based on Boltzmann Exploration **17**

4.0.1 Value Function Approximation Algorithm 18

4.0.2 Boltzmann Exploration 20

4.0.3 Parametric Cost Function Approximation for Solving MDP (3.11) 21

5 Computational Experiments **24**

5.0.1 Experimental Setup 24

5.0.2 Value Function Approximation (VFA) Algorithm Performance 25

5.0.3 Impact of Model Parameters 34

5.0.4 Driver Matching Outcomes 42

5.1 Platform Profit Analysis 45

Appendices **47**

Appendices **48**

Bibliography **48**

Acronyms **49**

List of Tables

3.1 Notation summary. UPDATE! 9

5.1 Performance Metrics with Pool Sum Term Included 25
5.2 Comparison of Constant and Variable Driver Pools for Different Weights 42
5.3 Driver utilization statistics for different weight parameters 42
5.4 Driver idle time statistics for different weight parameters 43
5.5 Empty travel distance statistics for different weight parameters 44
5.6 Platform profit statistics for different weight parameters 45
5.7 Performance Metrics with Pool Sum Term Excluded 46

List of Figures

5.1 Convergence of VFA algorithm for different weight parameter values (Expected service level) . . . 26
5.2 Convergence of VFA algorithm for different weight parameter values (Expected driver utilization) . . . 27
5.3 Impact of sample paths on expected service level (1 sample path) . . . 29
5.4 Impact of sample paths on expected driver utilization (1 sample path) . . . 30
5.5 Impact of sample paths on expected service level (1 sample paths) . . . 31
5.6 Impact of sample paths on expected service level (25 sample paths) . . . 32
5.7 Impact of quadratic penalty function on expected service level . . . 34
5.8 Impact of exponential penalty function on expected service level . . . 35
5.9 Performance of myopic policy without driver and order utility terms (Expected service level) . . . 38
5.10 Impact of driver arrival probability on expected service level . . . 40
5.11 Impact of scheduling period duration on expected service level . . . 40
5.12 Distribution of driver utilization . . . 43
5.13 Distribution of driver empty travel distance for different weight parameters . . . 44
5.14 Pool sizes . . . 45

Listings

List of Algorithms

1 Value Function Approximation Algorithm 19
2 Boltzmann Exploration 21
3 Monte-Carlo-based Cost Function Approximation for Estimating Service Level and Driver Utilization 23

Chapter 1

Introduction

With the growth of online sales, internet retailers and logistics service providers are facing challenges in meeting the rising demand in an efficient, cost-effective manner. To meet the demand, several companies are relying on a crowdsourced delivery workforce, which involves using the excess capacity of private passenger vehicles on journeys that are already taking place to support delivery operations. Some examples of companies that have integrated crowdsourced delivery in their operations include Walmart's investigation of using in-store customers to deliver goods to online customers, DHL's 'MyWays' pilot in Stockholm, and Amazon's launch of the 'Amazon Flex' service in Seattle. By utilizing existing traffic flows, crowdsourced delivery could potentially enable faster and cheaper deliveries, while also reducing the negative environmental impact of dedicated delivery vehicles.

Crowdsourced drivers are freelance workers that use their own vehicles to perform last-mile deliveries. Drivers have varying levels of time and detour flexibility. Their self-scheduling nature means that their availability can be unpredictable. To ensure all parcels are delivered on time, platforms may use third-party services to handle tasks for which no ad-hoc driver can be found. Additionally, these platforms may employ various feedback mechanisms and external regulations to ensure the reliability and trustworthiness of the ad-hoc drivers.

In our research, we devise a novel algorithm that not only addresses existing criticisms and also ensures equitable compensation for drivers. We aim to propose a robust framework that not only optimizes driver satisfaction but also enhances the efficiency and sustainability of urban delivery systems.

Chapter 2

Literature Review

2.1 Operational Challenges and Optimization

This stream focuses on the operational aspects of crowdsourced delivery, such as routing, task assignment, and resource allocation. It addresses the challenges of coordinating a large number of ad-hoc drivers and optimizing the delivery process to achieve efficiency and cost-effectiveness.

2.1.1 Vehicle Routing and Pickup and Delivery Problem (Static and Dynamic)

This subsection focuses on developing optimization models and algorithms for routing crowdsourced delivery vehicles and addressing the pickup and delivery problem, which involves determining the optimal routes for vehicles to pick up and deliver packages while minimizing costs, travel distances, and delivery times. Studies in this area consider both static and dynamic scenarios, where static problems assume that all information is known in advance, while dynamic problems account for real-time changes and uncertainties.

The work of (1) is the first to consider crowdsourced drivers within the context of last-mile delivery, considering a static setting where all information is known beforehand and obtaining the optimal solution by solving an integer program. To address large-scale instances, the authors develop a multi-start heuristic.

While this stream of literature primarily focuses on developing optimization models and algorithms for routing vehicles and assigning delivery tasks, we take a higher-level approach by addressing the strategic decision of determining the optimal worker pool sizes across different time periods.

Instead of directly solving the routing and task assignment problems, we incorporate a dynamic

matching algorithm that optimizes the driver-order assignments given the chosen worker pool sizes. This matching algorithm is executed within the larger framework of iteratively updating the value function and exploring the decision space using Boltzmann Exploration.

(2) addresses the vehicle routing problem with time windows (VRPTW) in a context where the fleet consists of both dedicated vehicles and stochastic crowd-sourced vehicles. The authors develop a two-stage stochastic programming model to handle the uncertainty in crowd-sourced vehicle availability and propose a sample average approximation method along with a branch-and-cut algorithm to solve the problem. Their approach focuses on operational-level decision-making, specifically routing and scheduling deliveries using a hybrid fleet while considering the stochastic nature of crowd-sourced vehicle availability.

In contrast, our work takes a strategic-level approach by optimizing worker pool sizes for purely crowdsourced delivery systems across different time periods. While Torres et al. address routing and scheduling challenges with a hybrid fleet, we focus exclusively on crowdsourced drivers and do not directly tackle routing problems. Instead, we employ a dynamic matching algorithm within a larger framework using Boltzmann Exploration to update value functions and explore the decision space.

(3) address the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD) while considering the total number of collected goods. The authors propose a novel mathematical model that aims to minimize both the total travel time and the maximum number of collected goods per vehicle.

In contrast to our work, which focuses on strategic worker pool size optimization for purely crowdsourced delivery systems, this paper addresses an operational-level routing problem with a fixed fleet of vehicles. While we aim to balance driver utilization and demand fulfillment across different time periods, Guo and Wang concentrates on optimizing routes for simultaneous pickup and delivery tasks within a single time frame.

2.1.2 Hybrid Workforce Scheduling

This subsection deals with the challenge of optimally coordinating and scheduling a hybrid workforce consisting of both traditional delivery personnel and crowdsourced workers. Studies in this area aim to develop models and algorithms that effectively integrate and allocate tasks between

these two types of workforces, considering factors such as availability, costs, and service level requirements.

The literature on hybrid workforce scheduling in crowdsourced delivery typically focuses on coordinating and scheduling a workforce composed of both traditional delivery personnel and crowdsourced workers. However, we exclusively focus on crowdsourced delivery systems that employ only crowdsourced drivers, as is becoming increasingly common in practice. By narrowing the scope to this specific context, we offer insights and solutions tailored to the practical challenges faced by crowdsourced delivery systems that rely solely on an ad-hoc workforce of independent drivers. This allows for a more in-depth exploration of the unique dynamics and considerations involved in managing a purely crowdsourced delivery workforce.

2.1.3 Self-scheduling Workforce

(4) examines the impact of surge pricing on driver behavior and market efficiency in ride-sharing platforms. This paper makes the following key contributions:

- Analysis of driver responses to dynamic pricing
- Evaluation of the impact of surge pricing on market efficiency
- Insights into the self-scheduling nature of gig economy workers

While Chen focuses on dynamic pricing in ride-sharing, our work addresses similar self-scheduling challenges in the context of crowdsourced delivery through worker pool size optimization.

2.1.4 Additional Operational Challenges

This subsection covers additional operational challenges and optimization problems related to crowdsourced delivery, such as order bidding mechanisms, order rejection strategies, and other aspects that may impact the efficiency and performance of the delivery system.

In our research, we do not directly address these specific aspects, but instead focus on the strategic decision of determining the optimal worker pool sizes across different time periods, considering the trade-off between driver utilization and demand fulfillment. By optimizing the worker pool sizes, we indirectly influence the ability of drivers to accept or reject orders based on the available driver capacity and orders. However, the emphasis is on finding the optimal balance

between driver utilization and service level targets, rather than developing specific order bidding or rejection mechanisms.

2.2 Fair Task Allocation and Compensation in the Sharing Economy

This stream of literature focuses on developing fair and equitable mechanisms for allocating tasks and compensating workers in the context of the sharing economy, including crowdsourced delivery services. These studies aim to address issues related to fairness, transparency, and ethical considerations in sharing economy platforms.

2.2.1 Fair Task Allocation

One key aspect explored in this stream is the development of algorithms and models that ensure fair task allocation among crowdsourced workers. These studies consider factors such as worker preferences, skills, availability, and past performance to assign tasks equitably and without bias.

Although we do not directly develop algorithms or models for fair task allocation among crowdsourced drivers, we do incorporate fairness considerations. Specifically, by iteratively updating the value function and exploring the decision space using Boltzmann Exploration, we aim to ensure that drivers are allocated tasks equitably, regardless of their arrival times or other factors. However, instead of explicitly optimizing for fairness metrics, we achieve fairness implicitly by continuously refining the value function estimates and exploring the decision space in search of solutions that balance driver utilization and demand fulfillment.

2.2.2 Compensation Guarantees

Another important aspect is the design of fair compensation mechanisms that provide workers with a fair share of the generated revenue, taking into account factors such as the complexity of tasks, effort required, and market conditions.

(5) addresses the challenge of optimizing pricing and compensation in crowd-shipping platforms. The authors develop an integrated framework that combines matching and routing models to determine optimal pricing and compensation schemes under various demand and supply scenarios. Their approach includes a routing strategy to estimate travel distances for couriers and a

matching model to assign customers to couriers while maximizing platform benefits.

In contrast to our work, which focuses on strategic worker pool size optimization for purely crowdsourced delivery systems, Le et al. concentrate on the operational aspects of pricing and compensation. While both studies aim to improve the efficiency of crowdsourced delivery systems, we approach the problem from a different perspective. Our research emphasizes balancing driver utilization and demand fulfillment through worker pool size planning, whereas Le et al. focus on optimizing pricing and compensation to attract and retain customers and couriers.

Both studies incorporate elements of matching, but in different contexts. Our work uses a dynamic matching algorithm within a larger framework employing Boltzmann Exploration, while Le et al. integrate matching directly with routing to optimize pricing and compensation. Additionally, while our research indirectly addresses compensation through worker pool size planning and utilization optimization, Le et al. explicitly design and evaluate four different pricing and compensation schemes.

Unlike studies that develop explicit compensation schemes or incentive mechanisms, we measure the driver utilization obtained through a choice of worker pool sizes, which affects the value function used in Boltzmann Exploration.

By iteratively updating the value function and selecting new driver pool sizes vectors through Boltzmann Exploration, we indirectly influences the compensation received by drivers through their achieved utilization levels. This iterative process aims to find worker pool sizes that strike a balance between driver utilization and demand fulfillment.

Furthermore, we incorporate worker pool size planning, enabling the consideration of compensation guarantees by ensuring that sufficient driver capacity is available to meet target utilization levels.

2.3 Conclusion

From an operational standpoint, a key characteristic of a crowdsourced delivery is the self-scheduling nature of its workforce, allowing workers to independently determine their availability and frequency of work. Consequently, the platform's control over the supply of workers is indirect. To manage supply and demand dynamics effectively, platform operators often resort to dynamically

adjusting wages and prices, for example through surge pricing. This approach has been investigated in various papers examining ridesharing platforms such as Uber and Lyft (4, 6, 7).

Most similar to our work is (8) which investigates continuous approximation and value function approximation methods for scheduling a workforce consisting of professional drivers to achieve a specified service level target at minimum cost. The authors model crowdsourced drivers as random arrivals. Importantly, their work differs from our work in that it relies on a hybrid workforce of professional and crowdsourced drivers, whereas our work focuses exclusively on crowdsourced drivers.

Our research differs from existing work in two key aspects. First, we incorporate driver pool size planning into our framework to enable compensation guarantees. Second, we focus on crowdsourced delivery systems exclusively employing crowdsourced drivers, as is becoming increasingly common in practice. By narrowing our scope to this specific context, we offer insights and solutions tailored to the practical challenges faced by crowdsourced delivery systems.

Chapter 3

Mathematical Model

In this section, we first provide a general description of the problem and introduce the main notation. Then we describe the proposed mathematical formulation.

3.0.1 Problem Description

We consider a last-mile delivery network operated by a crowdsourced delivery platform, where orders and drivers arrive stochastically over a given planning horizon. Drivers do not pre-announce their work hours; however, the platform manages driver arrivals by deciding on the pool of drivers to hire at discrete periods $p \in [1, P]$. For a given pool size in period p, drivers arrive according to a binomial distribution with entry probability q_p. This implies that drivers enter the system probabilistically within the period they indicate to the platform, thus the platform does not know the exact number of drivers ahead of time.

At the operational level, the platform's objectives are twofold: (1) to meet as much customer demand as possible by achieving a targeted service level, and (2) to maximize the utilization of available drivers, ensuring their efficient assignment to delivery tasks. The latter objective is driven by criticisms of sharing economy platforms for insufficient driver compensation (REF), as well as concerns about congestion and environmental impact due to idle drivers waiting for matches (REF).

Therefore, the platform must select a pool size that balances these two objectives under uncertain order and driver arrivals. To evaluate the expected driver utilization, L, and order fulfillment, Q, resulting from a specific choice of driver pool size $\mathbf{x}$, the platform needs to solve a Markov Decision Process (MDP) that accounts for the operational decisions of the system, as well as the uncertain and dynamic arrivals of drivers and customer orders.

The platform's problem can be formulated as a *bilevel optimization* problem, where the *leader*

Table 3.1: Notation summary. UPDATE!

Decision Variables		Parameters	
x_p	integer variable indicating the number of drivers the platform accepts in period $p \in [P]$	T	planning horizon, e.g., a day
y_{tab}	binary variable which equals 1 if a driver with attributes a is matched with an order of attributes b at decision epoch t	p	period $p \in [1, P]$ indicates the discrete time intervals for pool size planning
		t	a decision epoch of the MDP model
		$L(\mathbf{x})$	Expected driver utilization given pool size vector $\mathbf{x}$
		$Q(\mathbf{x})$	Expected service level given pool size vector $\mathbf{x}$
		μ	target driver utilization level
		β	service level target
		w_s	service level target weight

determines the fleet size of drivers, denoted by x_p, for different periods $p \in \{1, \ldots, P\}$. Given a fleet size decision $\mathbf{x}$, the *follower* solves a Markov Decision Process (MDP) model that sequentially matches drivers and orders so as to maximize platform's profit and minimize the deviation from the target service level and driver utilization. The main notation used in the proposed model is defined in Table 3.1. For brevity, we denote the set of running indices $\{1, 2, \ldots, P\}$ by $[1, P]$, where P is the last period in the planning horizon.

3.0.2 Driver fleet size selection problem with an embedded Markov decision process

The platform aims to determine the optimal driver pool size vector for each time interval p throughout the planning horizon. Given the selected pool size, crowdsourced drivers join the platform probabilistically. The objective of the platform is to choose the minimum pool size that minimizes deviations from the service level and driver utilization targets, modeled as follows:

$$\begin{aligned} &\min_{\mathbf{x},\alpha} \; c(\mathbf{x}) + \alpha \\ &\text{s.t. } \alpha \geq w_s f^{serv}(\beta, Q(\mathbf{x})) + (1 - w_s) f^{util}(\mu, L(\mathbf{x})) \\ &\quad \mathbf{x} \in \mathbb{Z}^P_{\geq 0} \end{aligned} \tag{3.1}$$

Model (3.1) minimizes the sum of the cost of selecting $\mathbf{x}$ number of drivers in periods $[1, P]$,

plus the weighted sum of penalty functions $f^{util}(\cdot), f^{serv}(\cdot)$ associated with the deviation from driver utilization and service level targets, respectively, where $0 \leq w_s \leq 1$ is the weight of the service level objective. The expected utilization $L(\mathbf{x})$ and service level $Q(\mathbf{x})$ is obtained by solving an Markov decision process (MDP) given pool size $\mathbf{x}$. Thus, model (3.1) is a bi-level optimization problem, i.e., an optimization problem that contains an embedded optimization problem (**?**).

The service level penalty function $f^{serv}(\beta, Q(\mathbf{x}))$ is equal to $\zeta^{serv}(\beta - Q(\mathbf{x}))$ if $Q(\mathbf{x}) < \beta$, and 0 otherwise. Similarly, the driver utilization penalty function $f^{util}(\mu, L(\mathbf{x}))$ is equal to $\zeta^{util}(\mu - L(\mathbf{x}))$ if $L(\mathbf{x}) < \mu$, and 0 otherwise.

In model (3.1), the calculation of $L(\mathbf{x})$ and service level $Q(\mathbf{x})$ for a given $\mathbf{x}$ requires solving a large-scale MDP problem that captures the platform's operational dynamics and decisions throughout the planning horizon. Additionally, in this underlying MDP model, the random information governing the MDP transition probabilities includes endogenous uncertainty of driver arrivals. Specifically, the number of available drivers at a given decision epoch is a function of the selected pool size. We will now describe the MDP model given pool size $\mathbf{x}$.

3.0.3 Markov Decision Process (MDP) Model

Given fleet size vector $\mathbf{x}$, at the operational level, both orders and drivers arrive randomly to the crowdsourced delivery platform within a given planning horizon T (e.g., a day). The arrival of orders follows a Poisson process with a rate of λ per decision epoch t, where a decision epoch represents discrete time points within T when decisions are made. We assume that the duration of an epoch is shorter than the duration of a period for fleet size selection, i.e., $\delta(t) < \delta(p)$, where $\delta(\cdot)$ computes the duration in minutes.

When an order is announced, the platform receives information on its origin, destination, and fulfillment time window. Drivers do not pre-announce their work hours; however, for a given period p, the arrival of drivers follows a binomial distribution $\tilde{x}_p \sim \text{binomial}(x_p, q_p)$, where x_p is the fleet size at period p, determined by (3.1) and q_p is the probability of arrival to the platform, i.e., the probability that any given driver from the driver pool will log in or be available to accept matches during period p. Thus, the self-scheduling feature of drivers is captured by modeling driver arrivals as random binomial trials.

At the start of each decision epoch t, the platform assigns orders to drivers in a manner that

maximizes its profit, while considering how matching decisions impact drivers utilization. The components of the MDP are defined as follows.

3.0.3.1 State Space

The state variable S_t at decision epoch t is the minimal amount of information necessary and sufficient to make a decision. In the proposed MDP, the state variable comprises two main components: driver information and order information. The driver information includes the set of drivers in the system and their attributes, such as their availability for matching, current location and utilization up to the current time. At decision epoch t, this information is stored in a unique attribute vector a for each driver in the system, defined as:

$$a = (m_a, o_a, h_a, l_a, t_a^s, t_a^m, t_a^e)$$

where m_a is a binary indicator that equals 1 if the driver is available for matching, and 0 otherwise. o_a represents the driver's current location, and h_a indicates the number of epochs the driver has spent on-task up to time t, or up to the delivery of the current order if the driver is en-route. l_a denotes the actual utilization of the driver up to time t. The variables t_a^s, t_a^m, and t_a^e respectively represent the time the driver entered the platform, the time the driver becomes available after completing a delivery request, and the planned exit time of the driver.

We denote the set of drivers and their attributes at time t as $\mathcal{A}_t$, which is composed of available drivers $\mathcal{A}_t^{\text{avail}}$ for *available* drivers (i.e., those with $m_a = 1$), and $\mathcal{A}_t^{\text{route}}$*en-route* drivers (i.e., those with $m_a = 0$). The vector of available drivers is denoted as $R_t = (R_{ta})_{a \in \mathcal{A}_t^{\text{avail}}}$, where R_{ta} is a binary indicator that equals 1 if driver a is available at time t.

Order information describes the set of active orders, including their origin-destination (o-d) locations and time windows. Each order request is represented by a unique attribute vector b, defined as:

$$b = (o_b, d_b, [t_b^{\min}, t_b^{\max}])$$

where o_b and d_b denote the origin and destination of the order, respectively, and $[t_b^{\min}, t_b^{\max}]$ repre-

sents the time window for order fulfillment.

We denote the set of all order attributes b at time t as $\mathcal{B}_t$. The vector of orders at time t is denoted by $D_t = (D_{tb})_{b \in \mathcal{B}_t}$, where D_{tb} indicates the number of orders with attribute b at time t.

In addition to the current drivers and orders, the platform keeps track of past demand fulfillment and driver utilization from previous decision epochs. The number of matched orders and the total number of orders up to, but not including, epoch t are denoted by B_t^{matched} and B_t^{total}, respectively. Similarly, the platform tracks the utilization of all drivers who were previously available but have since exited the system. We denote the average utilization of these drivers as A_t^{util} and their total number as A_t^{total}. In the first epoch, the number of matched orders and the utilization of drivers in the system are both set to zero.

Consequently, the state of the system at epoch t is expressed as

$$S_t = (R_t, D_t, B_t^{\text{total}}, B_t^{\text{matched}}, A_t^{\text{util}}, A_t^{\text{total}}).$$

3.0.3.2 Action Space

The action space defines a time-feasible match between the set of drivers and the set of orders. At time t, the number of drivers with attribute a assigned to orders with attribute b is denoted by y_{tab}. Similarly, y_{ta0} represents the number of drivers with attribute a who are not assigned to any orders. At decision epoch t, the time required for a driver a to fulfill an order b is denoted by ι_{ab}. This time is expressed in terms of decision epochs as:

$$\iota_{ab} = \left\lceil \frac{\eta(||o_a - o_b|| + ||o_b - d_b||)}{\delta(t)} \right\rceil \tag{3.2}$$

where $||\cdot||$ is a norm that computes the distance between two nodes, η is a scalar converting the distance to a time estimate, and $\delta(t)$ is the length of a decision epoch. The set of time-feasible assignments, $\mathcal{D}_{ab}$, is defined as:

$$\mathcal{D}_{ab} = \{(a, b) \in \mathcal{A}_t^{\text{avail}} \times \mathcal{B}_t^{+} | t + \iota_{ab} \leq t_b^{\max}\}.$$

Thus, at decision epoch t, the decision vector $\mathbf{y}$ belongs to feasible set $\mathcal{Y}$ defined as

$$\mathcal{Y}_t = \left\{ \mathbf{y}_t \in \mathbb{Z}_{\geq 0}^{|\mathcal{A}_t^{\text{avail}}| \times |\mathcal{B}_t^+|} : \begin{array}{ll} \sum_{b:(a,b)\in\mathcal{D}_{ab}} y_{tab} = R_{ta}, & \forall a \in \mathcal{A}_t^{\text{avail}} \\ \sum_{a\in\mathcal{A}_t^{\text{avail}}} y_{tab} \leq D_{tb}, & \forall b \in \mathcal{B}_t \end{array} \right\}$$

3.0.3.3 Transition Function and Post-decision State

The post-decision state $S_t^y = (R_t^y, D_t^y, , B_t^{\text{total},y}, B_t^{\text{matched},y}, A_t^{\text{util},y}, A_t^{\text{total},y})$ captures the change in driver and order attributes immediately after a decision $\mathbf{y}_t$ is made. Given decision $\mathbf{y}_t$, a driver with attribute a transitions to attribute a', which is captured by the function $a^M(a, \mathbf{y}_t)$. To keep track of driver attribute transitions, we define indicator variable $\mathbb{1}(a^M(a, \mathbf{y}_t) = a')$, which equals 1 if a driver's attribute changes to a', and 0 otherwise. The post-decision resource state $R_t^y = (R_{ta}^y)_{a\in\mathcal{A}_t}$ is updated as:

$$R_{ta'}^y = \mathbb{1}(a^M(a, \mathbf{y}_t) = a')y_{tab}, \quad \forall a \in \mathcal{A}_t^{\text{avail}} \tag{3.3}$$

When a driver transitions from attribute a to a' as a result of a match, $l_{a'}$ is updated as the total utilization time after fulfilling order b, divided by the total active time in the system, i.e., $l_{a'} = \frac{h_a + \iota_{ab}}{t + \iota_{ab} - t_a^s}$. Similarly, if a driver is unmatched, $l_{a'}$ is updated as: $l_{a'} = \frac{h_a + 1}{t + 1 - t_a^s}$.

For the post-decision order vector, we first define the travel time between the origin and destination of a demand as ι_b, discretized into decision epochs, as: $\iota_b = \left\lceil \frac{\eta(||o_b - d_b||)}{\delta(t)} \right\rceil$. Any order with an expired time window is considered lost and is not moved forward to the next epoch. The set of lost orders is denoted as $\overline{\mathcal{B}}_t := \{\mathcal{B}_t : t + 1 + \iota_b > t_b^{max}\}$. For the remaining orders, order information is updated as follows:

$$D_{tb}^y = D_{tb} - \sum_{a\in\mathcal{A}_t} y_{tab}, \quad \forall b \in \mathcal{B}_t \setminus \overline{\mathcal{B}}_t \tag{3.4}$$

The number of matched orders is updated as

$$B_t^{\text{matched},y} = B_t^{\text{matched}} + \sum_{a \in \mathcal{A}_t^{\text{avail}}} y_{tab}, \tag{3.5}$$

reflecting the addition of newly fulfilled orders to the cumulative total match. The average utilization of drivers is updated upon drivers' exit, and thus is not updated in the post-decision-state, i.e., $A_t^{\text{util},y} = A_t^{\text{util}}$. Similarly, the total number of drivers and the total number of orders remains unchanged in the post-decision state.

3.0.3.4 Random Information and Pre-decision State

Let W_t denote the random information that is continuously arriving between epochs $t-1$ and t. $W_t = (W_{ta}, W_{tb})_{a \in \mathcal{A}_t, b \in \mathcal{B}_t}$, where W_{ta} denotes a driver with unique identifier of attribute a who joins or leaves the system between $t-1$ and t, and W_{tb} is the number of new orders with attribute b that arrive during this time interval. Drivers arriving in period p do so within the first t' epochs. Subsequently, a driver has a preferred exit at a random time t_a^e, uniformly distributed within the interval $[0.5\delta(p), 1.5\delta(p)]$, where $\delta(p)$ is the duration of a period. This implies that drivers remain for at least half of the duration of the period and overlap with a maximum of half of the duration of the subsequent period. A driver exits if $t_a^e \leq t+1$. In the last epoch, all remaining drivers in the system exit.

The arrival of random information transitions the system from post-decision state S_t^y to the next pre-decision state S_{t+1}, where $S_{t+1} = (R_{t+1}, D_{t+1}, , B_{t+1}^{\text{total}}, B_{t+1}^{\text{matched}}, A_{t+1}^{\text{util}}, A_{t+1}^{\text{total}})$. Driver attributes may also change as a result of exogenous information, for example due to delays in travel time. To capture these changes, we again use indicator variable $\mathbb{1}(a^M(a, W_t) = a')$, which equals 1 if a driver's attribute, a, changes to a', and 0 otherwise. Similar to the post-decision state transition function, $a^M(a, W_t) = a'$ determines the change in driver attribute due to W_t. The pre-decision resource vector, $R_{t+1} = (R_{(t+1)a})_{a \in \mathcal{A}_{t+1}}$, is computed as:

$$R_{(t+1)a'} = \mathbb{1}(a^M(a, W_t) = a') R_{ta}^x + W_{(t+1)a'}, \quad \forall a \in \mathcal{A}_{t+1} \tag{3.6}$$

The pre-decision demand vector, $D_{t+1} = (D_{(t+1)b})_{b \in \mathcal{B}_{t+1}}$, is the sum of carried forward orders D^y_{tb} and newly arrived orders $W_{(t+1)b}$, and is calculated as:

$$D_{(t+1)b} = D^y_{tb} + W_{(t+1)b}, \quad \forall b \in \mathcal{B}_{t+1} \tag{3.7}$$

The number of matched orders does not change from that in the pre-decision state ($B^{\text{matched}}_{t+1} = B^{\text{matched},y}_t$), while the total number of orders is updated by adding the new orders, i.e.,

$$B^{\text{total}}_{t+1} = B^{\text{total},y}_t + \sum_{b \in \mathcal{B}_t | W_{tb} > 0} W_{tb}. \tag{3.8}$$

Finally, the number of drivers that exit and their expected utilization of drivers are respectively updated as

$$A^{\text{total}}_{t+1} = A^{\text{total},y}_t + \sum_{a \in \mathcal{A}_t | W_{ta} < 0} W_{ta}, \tag{3.9}$$

and

$$A^{\text{util}}_{t+1} = \frac{(A^{\text{util},y}_t \times A^{\text{total},y}_t) \; + \sum\limits_{a \in \mathcal{A}_t | W_{ta} < 0} l_a}{A^{\text{total}}_{t+1}}. \tag{3.10}$$

The system evolves as follows: $(S_0, \mathbf{y}_0, S^y_0, W_1, S_1, \mathbf{y}_1, S^y_1, W_2, S_2, \ldots, S_t, \mathbf{y}_t, S^y_t, \ldots, S_T)$.

3.0.3.5 MDP Contribution Function

The contribution function of the Markov Decision Process (MDP) model is given by:

$$J_t(S_t) = \max_{\mathbf{y}_t \in \mathcal{Y}_t} \left(C_t(S_t, \mathbf{y}_t) + \gamma \sum_{S_{t+1}} \text{prob}(S_{t+1} \mid S_t, \mathbf{y}_t, \mathbf{x}) J_{t+1}(S_{t+1}) \right) \tag{3.11}$$

where $C_t(S_t, \mathbf{y}_t) \coloneqq \sum\limits_{b \in \mathcal{B}_t, a \in \mathcal{A}_t} (r(b) - c(a,b))\, y_{tab}$ represents the immediate reward of matching decision $\mathbf{y}_t$ in state S_t, with $r(b)$ being the revenue from delivering order b and $c(a,b)$ being the cost of matching driver a with order b. $\text{prob}(S_{t+1} \mid S_t, \mathbf{y}_t, \mathbf{x})$ denotes the probability of transitioning to state S_{t+1} in the next decision epoch, given the current state S_t, decision vector $\mathbf{y}_t$, and pool size

$x_{p(t)}$ chosen at period p, where $p(t)$ maps the period to decision epochs.

Equation (3.11) represents the Bellman recursion function, which determines the optimal decision $\mathbf{y}_t$ in state S_t. This function balances the immediate myopic profit with the discounted expectation of future profit, where γ is the discount factor ($0 < \gamma < 1$). Additionally, we have the boundary condition $C_T(S_T) = -w_s f^{\text{serv}}(\beta, Q(\mathbf{x})) - (1 - w_s) f^{\text{util}}(\mu, L(\mathbf{x}))$, for all $S_T \in \mathcal{S}$. The expected utilization of drivers and the expected platform service level are computed as

$$L(\mathbf{x}) = A_T^{util} \quad \text{and} \quad Q(\mathbf{x}) = \frac{B_T^{\text{matched}}}{B_T^{\text{total}}} \tag{3.12}$$

respectively, and the penalty functions $f^{serv}(\cdot), f^{util}(\cdot)$ are as defined in Section 3.0.2. The boundary condition implies that at the end of the planning horizon (e.g., a day), given fleet size vector $\mathbf{x}$, the platform incurs a penalty for deviations from the expected service level and driver utilization. These penalties are subsequently incorporated into model (3.1).

Chapter 4

Solution Method: Value Function Approximation based on Boltzmann Exploration

Fleet size problem (3.1) is a bilevel optimization problem, where for each decision vector $\mathbf{x}$, one needs to solve the large-scale MDP described in Section 3.0.3 to compute the expected driver utilization and service level. For realistic-sized problems that accurately capture the platform's dynamics, MDP problem (3.11) suffers from the curse of dimensionality in the state, action and outcome spaces, due to the large number of drivers and orders, their detailed attribute spaces and the large set of possible decisions. Additionally, the underlying MDP is a function of the selected fleet size $\mathbf{x}$, since the fleet size choice affects the realization of random driver arrival. Thus, solving (3.1) exactly becomes intractable.

In this section, we develop a value function approximation algorithm that iteratively searches the solution space of driver fleet size. The algorithm utilizes reinforcement learning, specifically Boltzmann exploration, to select a driver fleet size. For a given pool size $\mathbf{x}$, we then solve an approximation of MDP (3.11) using a Monte-Carlo-based parametric cost function approximation to obtain estimates $Q(\mathbf{x})$ and $L(\mathbf{x})$. The value function estimate from Boltzmann exploration is then updated to guide the selection of the pool size in subsequent iterations. This process continues until a predefined maximum number of iterations is reached.

In what follows, we detail the steps of the value function approximation method providing justification for each of its components.

4.0.1 Value Function Approximation Algorithm

The solution space of model (3.1) is very large. Specifically, enumerating all possible pool sizes requires evaluating $(\overline{x})^P$ possible solutions, where $\overline{x}$ is an upper bound on the maximum pool size for each period. Since fleet size problem (3.1) is a multi-period problem, similar to (8), we leverage Value Function Approximation (VFA) - an approximate dynamic programming method for solving sequential decision problems (**?**) - to efficiently navigate the solution space. VFA estimates the expected reward or value of being in a given state for each period. In a given time period p, the state S_p is the selected pool sizes of previous periods, i.e., $S_p := (x_{p'})_{p' \in \{1,\dots,p-1\}}$. The decision given state S_p determines the optimal pool size for all future periods. Thus, the value function is expressed as:

$$V(S_p) := \min_{[x_p, x_P] \in \mathbb{Z}^{P-p+1}_{\geq 0}} c(x_p, \dots, x_P) + w_s f^{serv}(\beta, Q(x_p, \dots, x_P)) + (1 - w_s) f^{util}(\mu, L(x_p, \dots, x_P))$$

The goal is to find the best fleet size by iteratively improving a value function that evaluates the quality of potential solutions. Given the large number of possible states S_p, we employ separation as a dimensionality reduction technique to better approximate the value of the state space. This approach is utilized in (8) and is detailed in (**?**). Instead of storing all previous decisions, we approximate the state variable as $V(p, x_p)$, thus previous decisions are not included explicitly in the state space, but are accounted for implicitly in the approximation (**?**). In the proposed VFA algorithm, we use Boltzmann exploration to balance exploring the solution space and exploiting known value function estimates to select the next pool size during the search process.

Initially, the value function $V(p, x_p)\ \forall p \in [1, P], \forall x_p \in X$ is set to zero, where X is the solution space that can represent upper and lower bounds on the driver pool size of a given period obtained from domain knowledge. The best solution $\mathbf{x}^*$ is initialized to a randomized pool size its value v^* is initialized to zero. For each iteration i of the algorithm, Boltzmann exploration is employed to generate a new solution $\mathbf{x}_i$, where the current value function V, the solution space X, and the iteration count i are used to probabilistically select solutions that balance exploration and exploitation. We discuss the Boltzmann exploration step in detail in Section 4.0.2.

Once a new solution $\mathbf{x}_i$ is generated, its objective value v_i is evaluated using the Monte Carlo-

based cost function approximation for estimating the expected utilization and service level in the underlying MDP. This involves generating multiple sample scenarios, and solving a series of matching problems in a rolling horizon fashion to estimate the expected service level $Q(\mathbf{x}_i)$ and driver utilization $L(\mathbf{x}_i)$, reflecting the quality of the solution $\mathbf{x}_i$. The objective value v_i is computed as:

$$v_i = \sum_{p \in [1,P]} c(\mathbf{x}_i) + w_s f^{serv}(\beta, Q(\mathbf{x}_i)) + (1 - w_s) f^{util}(\mu, L(\mathbf{x}_i)) \tag{4.1}$$

which is the objective function evaluation of (3.1) given solution $\mathbf{x}_i$. If the evaluated value v_i is better than the current best value v^*, the best value v^* is updated to v_i, and the best solution $\mathbf{x}^*$ is updated to $\mathbf{x}_i$. Subsequently, the new value function is computed as follows:

$$V_i(p, x_p) = \sum_{p' \in [p,P]} x_{p'} + (i+1) \left(w_s f^{serv}(\beta, Q(x_{i,p'})) + (1 - w_s) f^{util}(\mu, L(x_{i,p'})) \right) \tag{4.2}$$

where the second term is a penalty function that increases linearly in the iteration count i to discourage deviating from the service level and utilization targets as the iteration count increases. Given the new value function estimate, $V(p, \mathbf{x}_i)$ is updated as a weighted average of the previous and the new estimates as follows:

$$V(p, x_p) = (1 - \rho) V(p, x_p) + \rho V_i(p, x_p) \tag{4.3}$$

where ρ is a step size defined as $\rho = \frac{1}{\sqrt{N(x_p)}}$, where $N(x_p)$ is the number of times the solution x_p has been encountered in previous iterations. The steps of the algorithm are defined in Algorithm 1. The algorithm returns the best fleet size $\mathbf{x}^*$ and its associated value v^*. Next, we will describe the details of the Boltzmann exploration and the cost function approximation algorithms.

Algorithm 1 Value Function Approximation Algorithm

Initialize $V, v^* \leftarrow 0$, and the initial random best solution $\mathbf{x}^*$ $i = [1, I]$ $i = 1$ $\mathbf{x}_i \leftarrow \mathbf{x}^*$ // First Solution $\mathbf{x}_i \leftarrow$ Boltzmann Algorithm 2 // Generate New Pool Size $v_i \leftarrow$ Cost Function Approximation Algorithm 3 Given $\mathbf{x}_i$ // Evaluate Pool Size $v_i < v^*$ $v^* \leftarrow v_i$ // Update of Best Found Value $\mathbf{x}^* \leftarrow \mathbf{x}_i$ // And Best Pool Size Update $V(\cdot)$ using Equation (4.3) **return** $\mathbf{x}^*, v^*$

4.0.2 Boltzmann Exploration

At each iteration of Algorithm 1, we employ Boltzmann exploration, a reinforcement learning technique, to navigate the large decision space and select a candidate pool size $x_p,\ p \in [1, P]$. Boltzmann exploration aims to balance the exploration-exploitation trade-off when navigating a solution space by assigning a probability to each potential solution, which is calculated using an exponential function of the state's estimated value. Specifically, the probability Prob(x_p) of selecting fleet size x_p in period p is given by:

$$\text{Prob}(x_p) = \frac{e^{-V(p,x_p)/\tau}}{\sum_{x'_p \in X_p} e^{-V(x'_p)/\tau}} \tag{4.4}$$

where $V(p, x_p)$ is the estimated value (cost) of solution x_p and τ is a temperature parameter τ that controls exploration. A higher τ yields a more uniform probability distribution over the actions, thereby favoring exploration. Conversely, a lower τ results in a more peaked probability distribution, promoting the exploitation of the best-known solutions. Initially, τ is set to a high value to encourage the exploration of the decision space due to the lower confidence in the value function estimates, as the algorithm has not yet extensively explored the solution space. As the number of iterations increases, the algorithm gradually shifts towards exploitation, making higher-valued actions more likely to be chosen while ensuring that lower-valued actions still maintain a non-zero probability of selection. This contrasts with fixed probability methods such as the epsilon-greedy approach. Under certain conditions, the probabilistic approach of Boltzmann exploration can lead to more efficient exploration and potentially better convergence properties (9).

For each period, we denote by X_p the set of fleet size solutions at period p. For each candidate solution $x_p \in X_p$, $u(k)$ represents the weight of the k-th potential solution x_p, calculated as $e^{-V(p,x_p)/\tau}$. This weight reflects the probability of selecting each solution based on its estimated value, with lower-cost $V(p, x_p)$ yielding higher weights. The sum of these weights, denoted as W, normalizes the probability distribution, ensuring that the total probability across all potential solutions sums to 1. A uniform random variable u is sampled from the range $[0, W]$ to introduce stochasticity in the selection process. The cumulative weight sum, u^{sum}, is then incrementally accumulated by iterating through the weights $u(k)$ until u^{sum} exceeds u. This procedure probabilis-

tically determines which solution to select, balancing exploration and exploitation as guided by the temperature parameter τ. The steps of the Boltzmann exploration algorithm are summarized in Algorithm 2.

Algorithm 2 Boltzmann Exploration

$\mathbf{x}^* \leftarrow \mathbf{0}_P$ // Initialize solution $p \in [1, P]$ $d \leftarrow \max_{x_p \in X_p} V(x_p) - \min_{x_p \in X_p} V(x_p) + \epsilon$ // Value spread $\tau \leftarrow \frac{10 \times d}{i}$ // Temperature parameter $u \leftarrow \mathbf{0}_{|X_p|}$ // Initialize weights $k \leftarrow 0$ // Initialize solution index $x_p \in X_p$ $u(k) \leftarrow e^{-V(p,x_p)/\tau}$ // Calculate weight $k \leftarrow k + 1$ $U \leftarrow \sum_{k \in |X_p|} u(k)$ // Sum of weights $u \sim \mathcal{U}(0, W)$ // Sample random variable $u^{\text{sum}} \leftarrow 0$ // Temporary weight sum $k \leftarrow 0$ // Initialize solution index $u^{\text{sum}} \leq u$ $u^{\text{sum}} \leftarrow u^{\text{sum}} + u(k)$ $k \leftarrow k + 1$ $x_p^* \leftarrow X_p(k)$ // Return element k in the solution set X_p **return** $\mathbf{x}^* = (x_p^*)_{p \in [1,P]}$

4.0.3 Parametric Cost Function Approximation for Solving MDP (3.11)

For each candidate solution $\mathbf{x}$ in the value function approximation algorithm, we need to solve MDP (3.11) to obtain the expected driver utilization $L(\mathbf{x})$ and service level $Q(\mathbf{x})$. However, for realistically sized problems, solving MDP (3.11) becomes computationally intractable due to the curse of dimensionality in the state, action, and outcome spaces. This complexity arises from the continuous location information and detailed attributes associated with both drivers and orders, which are required to accurately track the status of drivers throughout the planning horizon and compute their utilization. Furthermore, the MDP must be solved at each iteration of the value function approximation algorithm. Therefore, it is imperative to develop an efficient approximation method to solve the MDP.

To address these challenges, we propose a Monte-Carlo-based cost function approximation method that solves a series of matching problems in a rolling horizon manner over a set of discrete scenarios Ξ. Each scenario $\xi \in \Xi$ represents a sample path of the random information in the system. Then, for each scenario ξ, at decision epoch t an optimization model selects the best matching decisions given driver and order information. Based on the chosen matching decisions $\mathbf{y}$ and the arrival of new information between epochs t and $t+1$, the status of the drivers and order information is updated using Equations (3.3)-(3.7). The model is then resolved for epoch $t+1$ and the process continues until the last decision epoch T.

The matching problem is formulated as follows

$$\max_{\mathbf{y}} \sum_{a,b \in \mathcal{D}_{ab}} \pi_{ab} y_{tab} \tag{4.5a}$$

$$\sum_{b:(a,b) \in \mathcal{D}_{ab}} y_{tab} = R_{ta}, \qquad \forall a \in \mathcal{A}_t \tag{4.5b}$$

$$\sum_{a \in \mathcal{A}_t} y_{tab} \leq D_{tb}, \qquad \forall b \in \mathcal{B}_t \tag{4.5c}$$

$$y_{tab} \in \mathbb{Z}^+, \qquad \forall (a,b) \in \mathcal{D}_{ab} \tag{4.5d}$$

where π_{ab} denotes the profit of matching a driver with attributes a to an order with attributes b. In the simple myopic case, $\pi_{ab} \coloneqq r(b) - c(a,b)$ represents the revenue from fulfilling order b minus the cost of matching driver a with order b. However, the myopic policy does not account for driver utilization h_a or the proximity of orders to their delivery deadlines $t_b^{\max}$. To address this, we modify the objective function using parametric cost function approximation, which allows us to incorporate these factors so as to partially capture the effect of uncertainty when making decisions (**?**).

The parametric cost function approximation aims to adjust the cost function to prioritize certain matches. Specifically, it introduces penalties for deviations from target utilization μ and for orders approaching their delivery deadlines. To prioritize drivers with lower utilization (i.e., $h_a < \mu$), we incorporate a positive utility term when a driver's utilization falls below the target μ. Similarly, to prioritize the matching of orders nearing their deadlines, we introduce a utility term that increases as the decision epoch approaches $t_b^{\max}$.

Thus, the profit of matching driver a with order b under the parametric cost function approximation is given by:

$$\pi_{ab} = r(b) - c(a,b) + \lambda g^{util}(l_a) + (1-\lambda) g^{serv}(t, t_b^{\max})$$

where $g^{util}(l_a)$ is a penalty function that measures deviations from the target utilization, based on a driver's current utilization l_a, and $g^{serv}(t, t_b^{\max})$ is a penalty function that prioritizes the fulfillment of order b as it approaches its delivery deadline $t_b^{\max}$.

We propose the following piecewise-linear penalty functions:

$$g^{util}(l_a) = \begin{cases} \zeta(\mu - l_a) & \text{if } l_a < \mu \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

$$g^{serv}(t, t_b^{\max}) = \zeta \max\left[1 - \frac{t_b^{\max} - t}{t_b^{\min} - t_b^{\max}},\ 0\right] \tag{4.7}$$

where non-negative scalar ζ controls the magnitude of the penalty. The proposed penalty functions are motivated by their simple and intuitive interpretation. Specifically, when a driver's utilization falls below the target μ, the penalty's magnitude is a linear function of the difference between the actual utilization and the target. Similarly, for each order b, the penalty increases linearly as the delivery time window shrinks.

In Appendix XXX, we introduce alternative quadratic and exponential penalty functions to evaluate the impact of different penalty functional forms on the algorithm's performance.

The steps of the cost function approximation method are summarized in Algorithm 3.

Algorithm 3 Monte-Carlo-based Cost Function Approximation for Estimating Service Level and Driver Utilization

$\xi \in \Xi$ $t \in [1, T]$ Solve optimization model (4.5) Update the state variable: $(S_t^y | S_t, y_t)$, $(S_{t+1} | S_t^y, W_{t+1}(\xi))$ using Equations (3.3)-(3.10) **Compute** Q_ξ, L_ξ using Equations (3.12) **Return** $Q(\mathbf{x}) = \frac{\sum_{\xi \in \Xi} Q_\xi}{|\Xi|}, Q(\mathbf{x}) = \frac{\sum_{\xi \in \Xi} L_\xi}{|\Xi|}$

Chapter 5

Computational Experiments

5.0.1 Experimental Setup

We conducted experiments using both real-world instances derived from the Chicago ridehailing dataset and synthetic instances. Unless otherwise specified, all experiments use the following parameters:

- Number of iterations: 1000
- Number of periods: 16 (each period is 1 hour)
- Number of sample paths: 10
- Driver arrival probability: 0.7
- Minimum driver pool considered in Boltzmann exploration: 1
- Maximum driver pool considered in Boltzmann exploration: 250

5.0.2 Value Function Approximation (VFA) Algorithm Performance

5.0.2.1 Overall performance metrics

Table 5.1: Performance Metrics with Pool Sum Term Included

Metric	$w_s = 0$	$w_s = 0.2$	$w_s = 0.5$	$w_s = 0.6$	$w_s = 0.8$	$w_s = 1$
Sum of pool sizes	88	343	376	378	381	1078
Demand Fulfilled						
Mean	0.29	0.94	0.97	0.97	0.98	0.97
SD	0.02	0.02	0.02	0.01	0.01	0.01
CV	0.07	0.03	0.02	0.01	0.01	0.01
95% CI	(0.28, 0.29)	(0.93, 0.94)	(0.97, 0.97)	(0.97, 0.98)	(0.97, 0.98)	(0.97, 0.97)
Driver Utilization						
Mean	1.00	0.97	0.94	0.94	0.93	0.37
SD	0.00	0.07	0.10	0.09	0.09	0.31
CV	0.00	0.07	0.10	0.10	0.10	0.83
95% CI	(1, 1)	(0.97, 0.97)	(0.93, 0.94)	(0.94, 0.94)	(0.93, 0.94)	(0.37, 0.38)
Fraction Drivers Meeting Guarantee						
Mean	0.91	0.88	0.82	0.84	0.83	0.13
SD	0.29	0.28	0.27	0.27	0.27	0.04
CV	0.32	0.32	0.33	0.32	0.33	0.33
95% CI	(0.85, 0.96)	(0.83, 0.93)	(0.77, 0.87)	(0.78, 0.89)	(0.78, 0.88)	(0.12, 0.14)
Driver Empty Distance (km)						
Mean	1.74	2.27	2.50	2.53	2.57	1.92
SD	1.16	1.76	1.95	1.99	2.04	2.26
CV	0.67	0.77	0.78	0.79	0.80	1.18
95% CI	(1.73, 1.75)	(2.27, 2.28)	(2.50, 2.51)	(2.52, 2.54)	(2.56, 2.57)	(1.91, 1.93)
Platform Profit						
Mean	6075.19	14121.87	14073.55	14055.51	14030.58	14631.00
SD	357.27	365.46	347.26	359.33	365.93	327.34
CV	0.06	0.03	0.02	0.03	0.03	0.02
95% CI	(6004.30, 6146.08)	(14049, 14194)	(14005, 14142)	(13984, 14127)	(13958, 14103)	(14566, 14696)
Driver Idle Time (min)						
Mean	0.00	2.35	4.40	4.20	4.42	40.32
SD	0.00	5.01	6.86	6.46	6.60	22.42
CV	–	2.13	1.56	1.54	1.49	0.56
95% CI	(0.00, 0.00)	(2.29, 2.41)	(4.32, 4.48)	(4.13, 4.28)	(4.34, 4.50)	(40.16, 40.48)
Driver Time in System (min)						
Mean	64.45	65.83	65.65	65.70	65.92	64.85
SD	17.83	17.97	17.83	17.88	17.99	18.61
CV	0.28	0.27	0.27	0.27	0.27	0.29
95% CI	(64.00, 64.89)	(65.60, 66.06)	(65.43, 65.86)	(65.49, 65.92)	(65.71, 66.14)	(64.72, 64.98)

5.0.2.2 Convergence Analysis

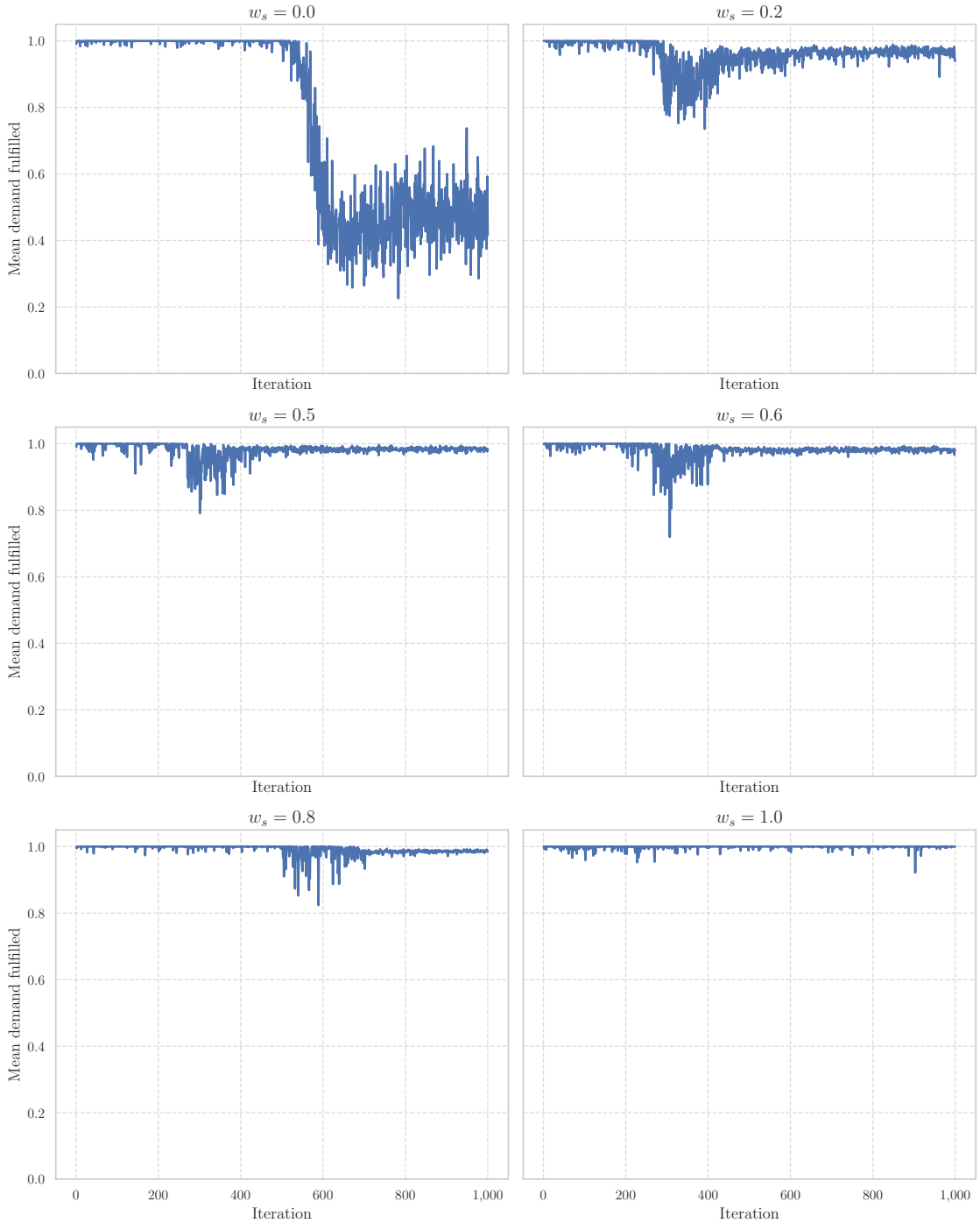


Figure 5.1: Convergence of VFA algorithm for different weight parameter values (Expected service level)

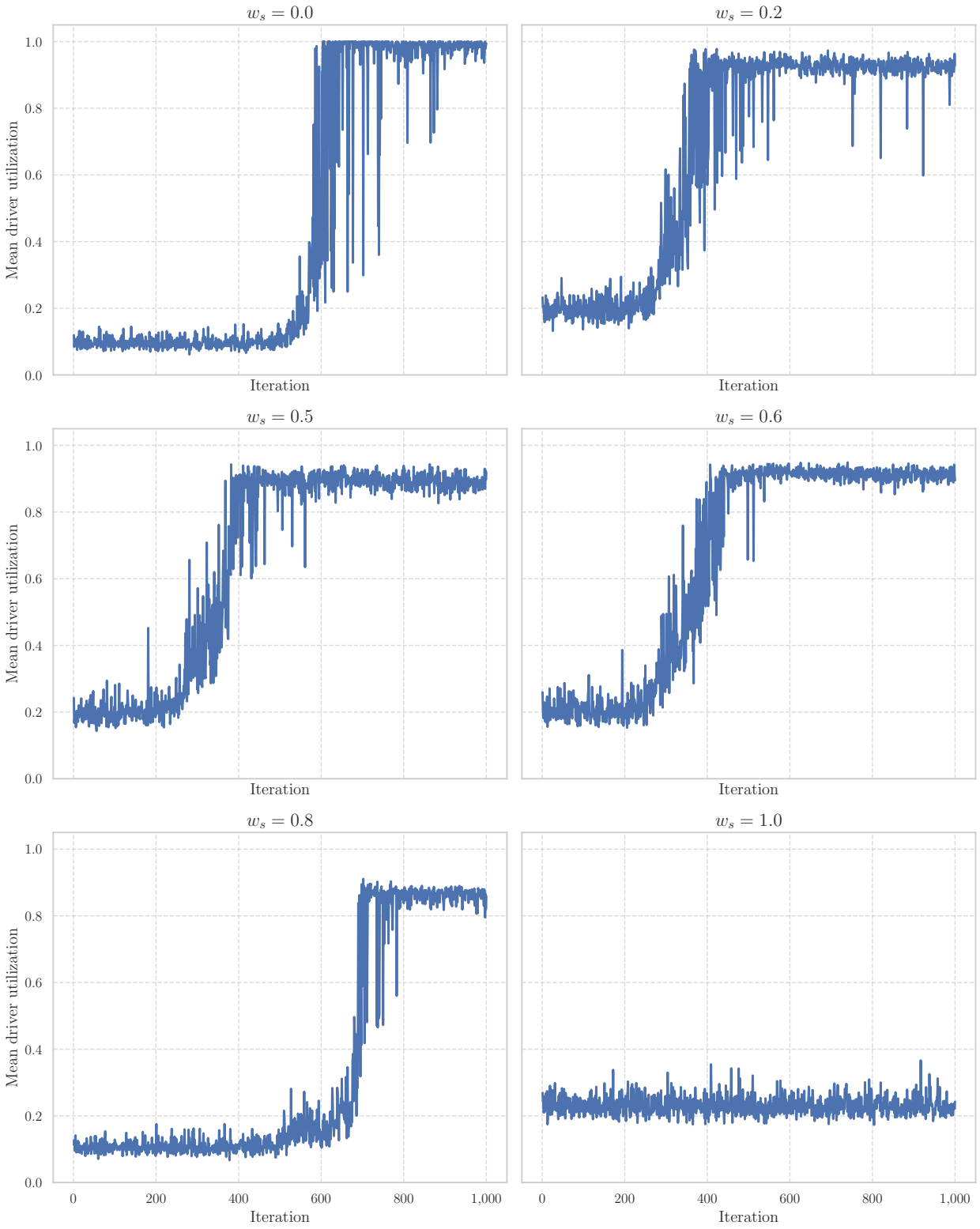


Figure 5.2: Convergence of VFA algorithm for different weight parameter values (Expected driver utilization)

Figures 5.1 and 5.2 demonstrate the convergence of the VFA algorithm. The algorithm shows consistent convergence patterns across different weight parameters. This signifies that the algorithm is robust to variations in parameters.

5.0.2.3 Impact of Sample Paths

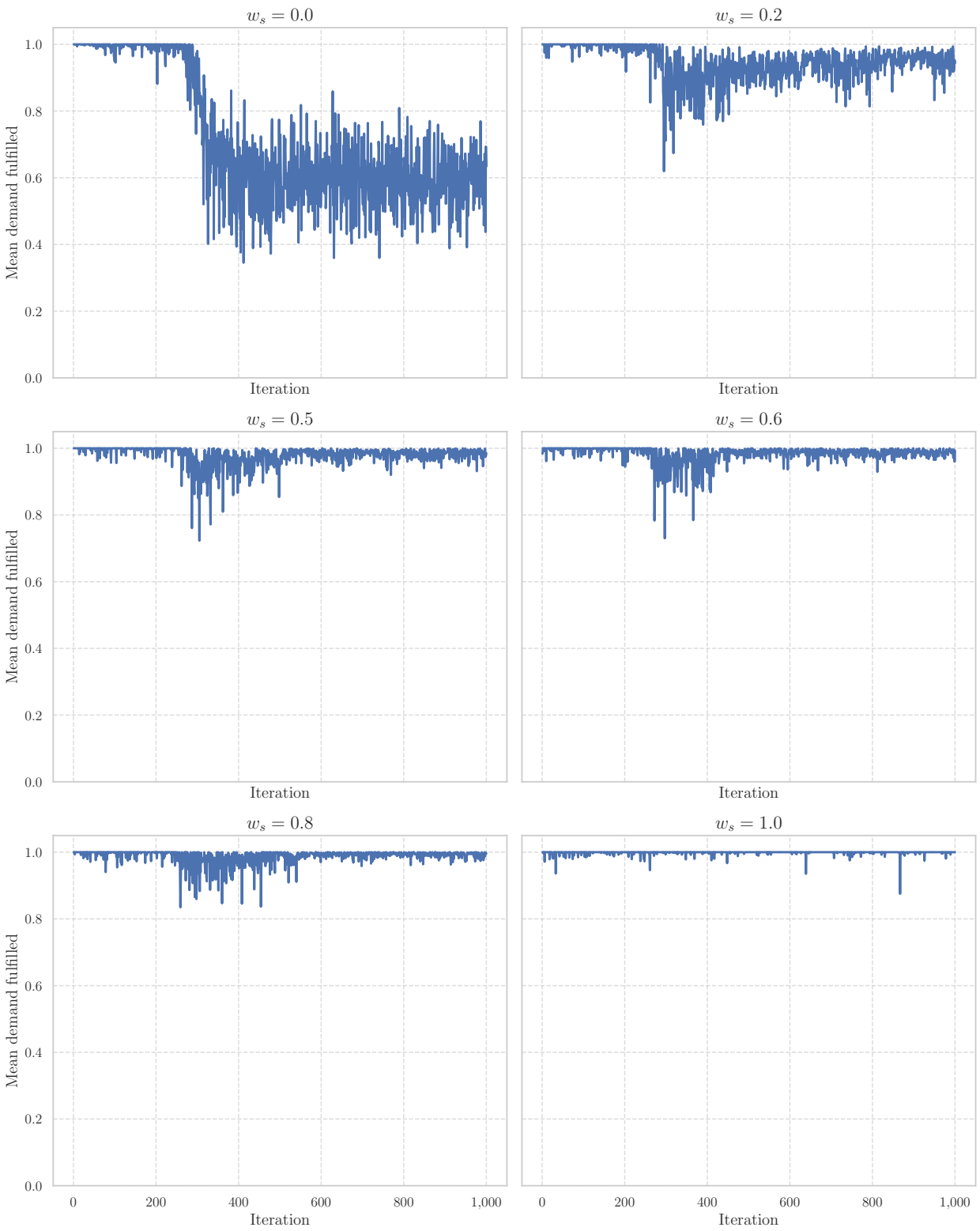


Figure 5.3: Impact of sample paths on expected service level (1 sample path)

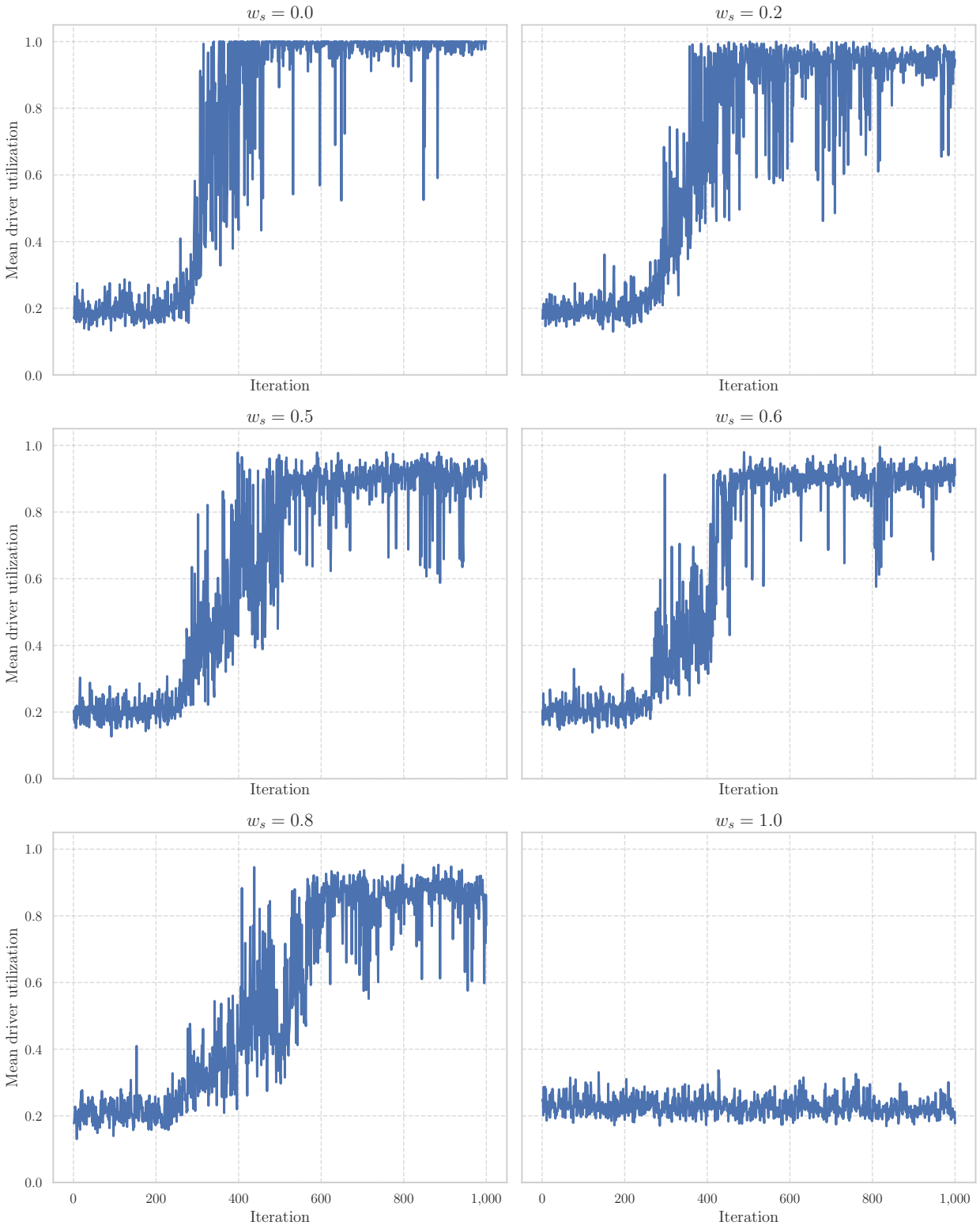


Figure 5.4: Impact of sample paths on expected driver utilization (1 sample path)

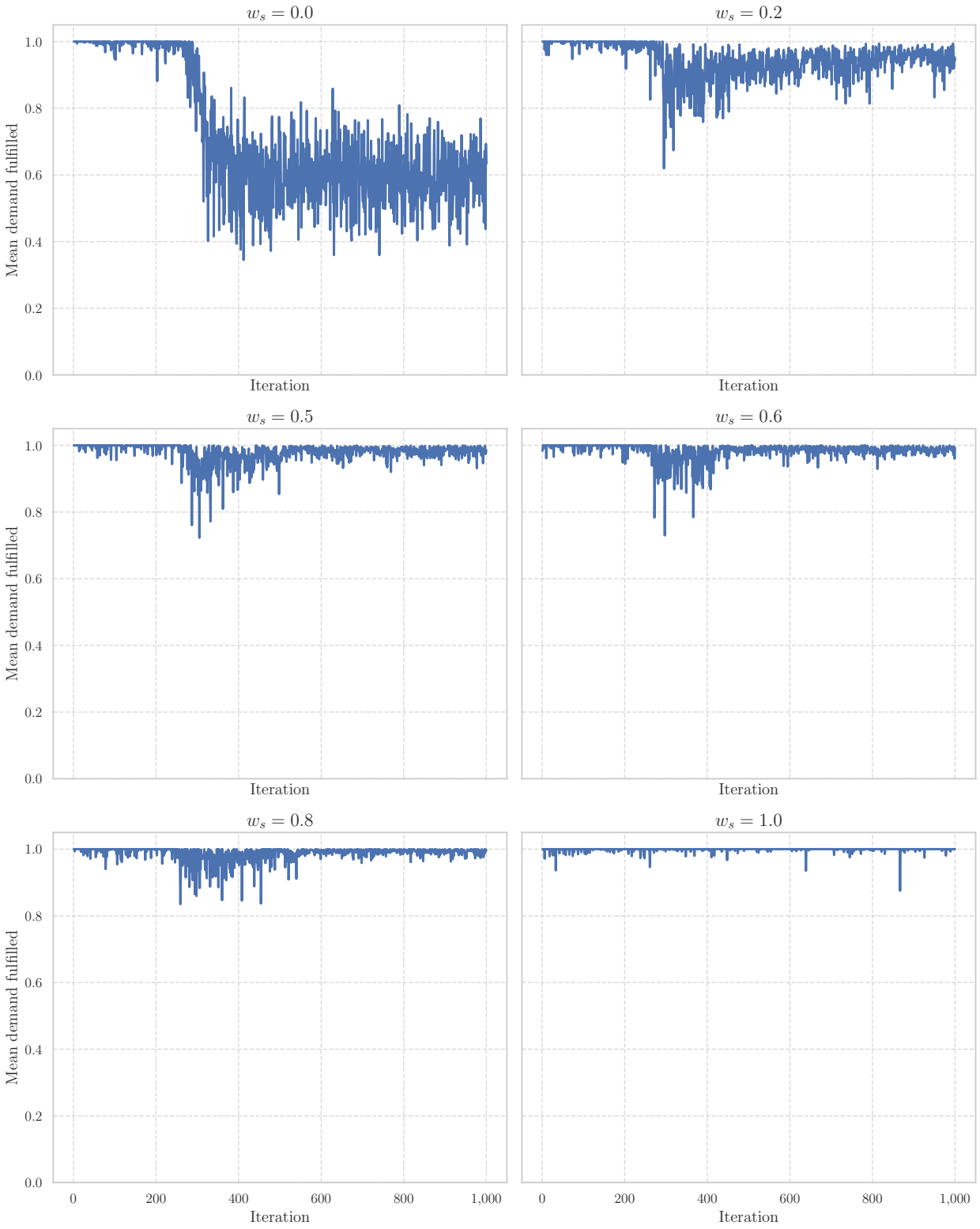


Figure 5.5: Impact of sample paths on expected service level (1 sample paths)

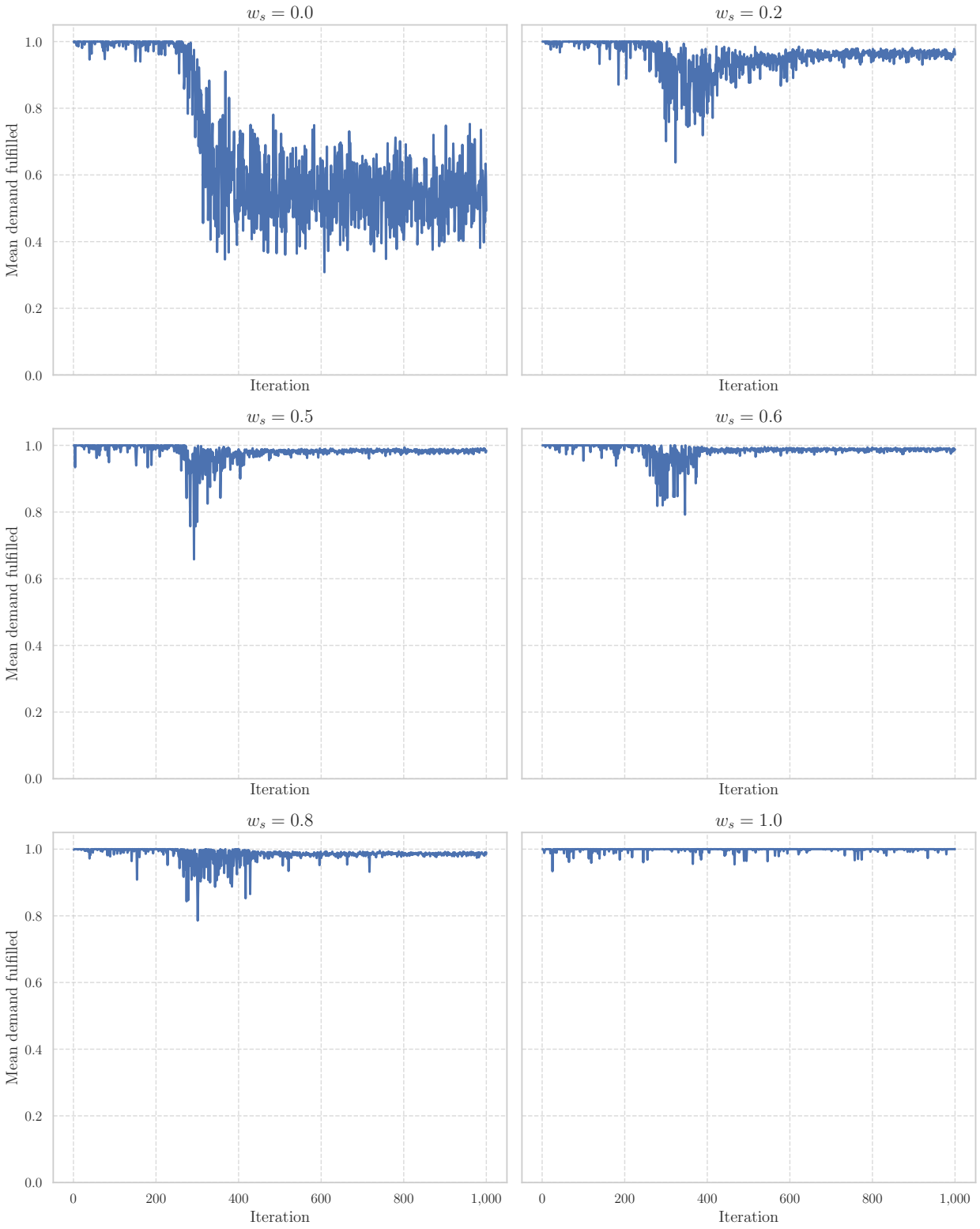


Figure 5.6: Impact of sample paths on expected service level (25 sample paths)

Figures 5.5 and 5.6 illustrate the impact of varying the number of sample paths on the convergence of the expected service level. Increasing the number of sample paths generally leads to smoother convergence.

5.0.3 Impact of Model Parameters

5.0.3.1 Effect of Penalty Functions

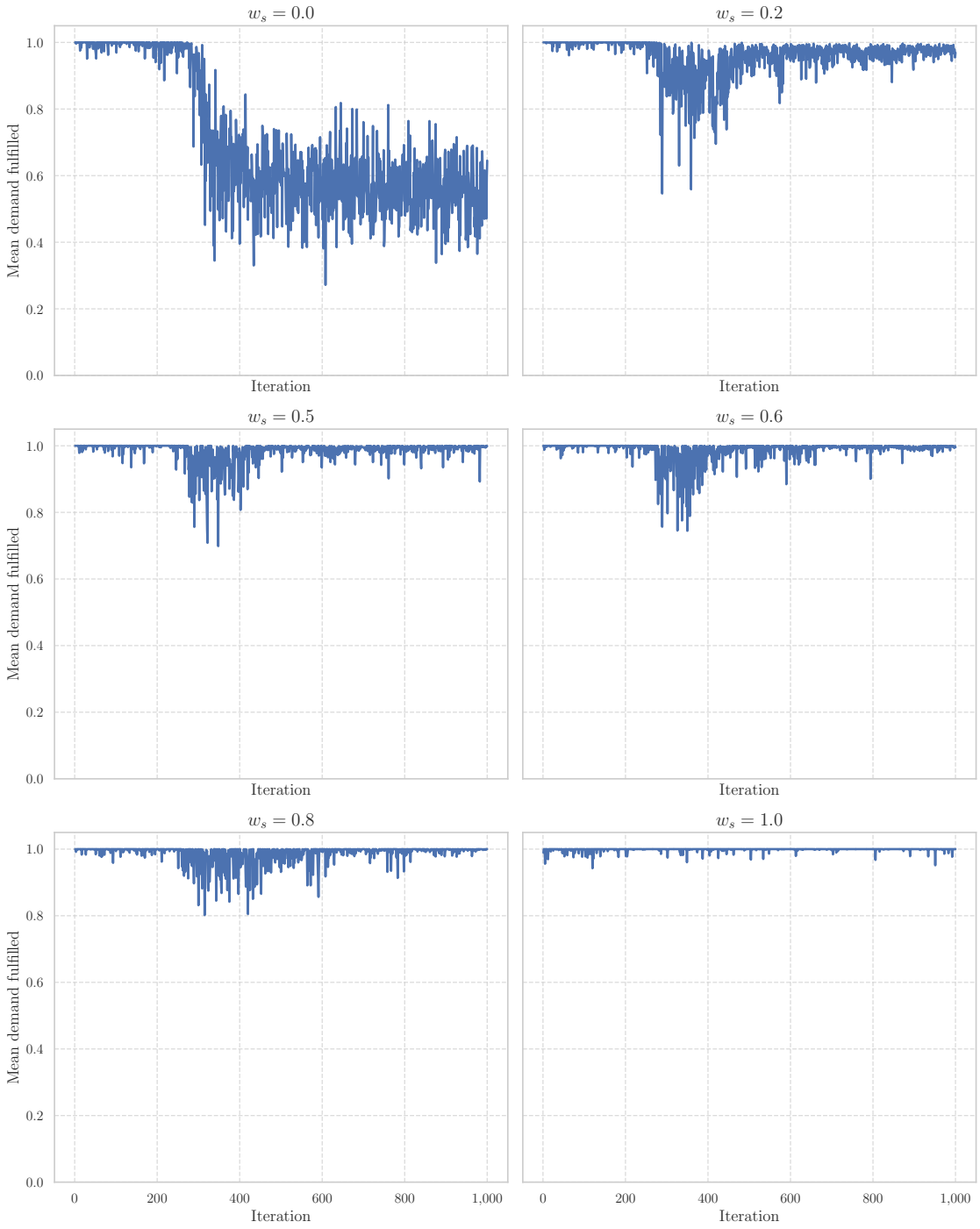


Figure 5.7: Impact of quadratic penalty function on expected service level

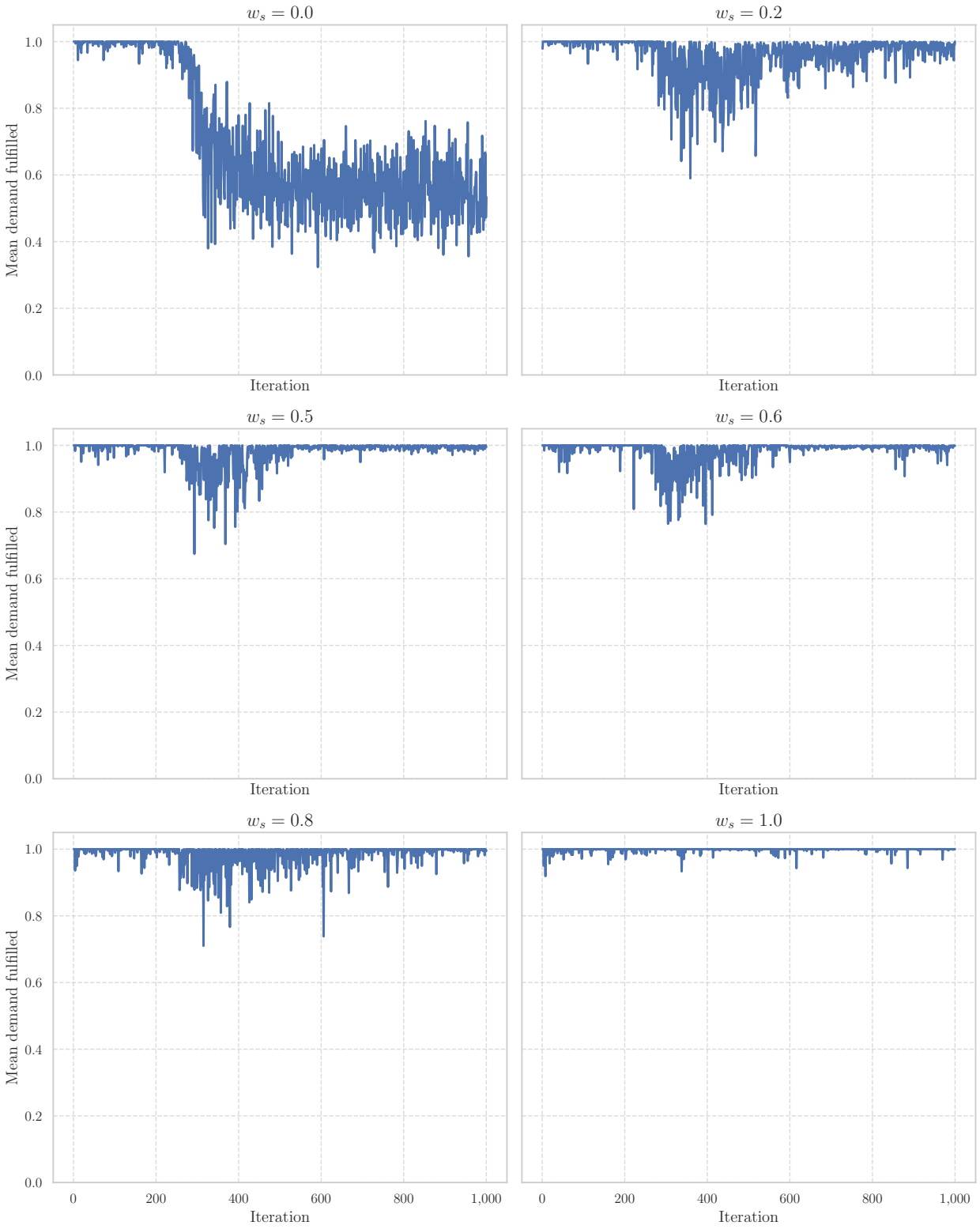


Figure 5.8: Impact of exponential penalty function on expected service level

Figures 5.7 and 5.8 demonstrate the impact of different penalty functions on the expected service level. The choice of penalty function can significantly influence the convergence behaviour and performance of the VFA algorithm.

Even when $w_s = 1$, the target utilization of 0.8 is still achieved. There are several reasons for this. First, driver utilization and service level are inherently correlated; a high service level can only be achieved with sufficient driver utilization. When $w_s = 1$, while the algorithm prioritizes service level and neglects driver utilization, preference will still be given to a pool with a lower sum due to the $sum(pool_sizes[p+1:])$ term, as long as the service level is sufficient. In other words, it is often possible for a smaller driver pool to achieve high service level in expectation. The algorithm is focused on choosing the driver pool that probabilistically minimizes the deviation from the service level target. This can be done in a manner that does not compromise driver utilization. In theory, if all drivers were to enter with a probability of 1, then there would be a pool that has exactly as many drivers as are needed to fulfil the service level target. For example, if the probability of entering is 1, after the algorithm terminates we could simply assign the pools at each period that represent the number of drivers required to fulfil all the orders or achieve a service level greater than or equal to the target and all the drivers would have a utilization close to 1.

Even when the binomial driver entering probability is lower than 1, a similar result can still be achieved although the pool sizes would be larger to account for the stochastic nature of driver arrivals. The stochasticity in driver arrivals means that in expectation a larger number of drivers enter the platform and have lower utilization on average.

Another factor that explains this behaviour is that the value function approximation algorithm does not converge to higher pool sizes as long as the service level target is achieved. For example, if $w_s = 1$ and two pool sizes achieve a similar service level, the algorithm will prefer the lower pool sizes because of the $sum(pool\_sizes[p+1:])$ term, which leads to higher driver utilization. As long as the achieved service level is higher than 0.95, the associated penalty term is 0 and within this constraint the algorithm will choose the pool with the smallest possible sum. This can be viewed as a soft constraint since the model achieves these targets through penalty terms. The penalty of 250 associated with deviations from the driver utilization and service level targets may in some cases be outweighed by the increase in service level achieved by increasing the pool sizes, so the algorithm may give preference to a pool that does not quite meet the service level target, even when $w_s = 1$.

This is consistent with the fact that the same trend is not true of the service level. When w_s = 0, the service level significantly drops because the algorithm prefers smaller pool sizes to not only improve driver utilization but also lower the $sum(pool_sizes[p+1:])$ term. There is no incentive to increase the pool sizes to meet demand. And even if higher pool sizes could lead to a similar driver utilization, the $sum(pool\_sizes[p+1:])$ term will still cause convergence to lower pool sizes.

This is also consistent with the fact that when w_s = 0 the mean driver utilization is nearly 1 (0.99...) but when w_s = 1 the fraction demand fulfilled is not as close to 1 (0.96 or 0.97). The $sum(pool_sizes[p+1:])$ term lowers the number of available drivers and as a result demand fulfillment. As a result, when w_s = 0 the inclusion of the summation term drives the algorithm towards smaller pool sizes irrespective of the impact of the service level. A smaller pool size will be preferred not only because the deviation from mean driver utilization is zero, but also because the summation term is lower. In other words, even if two pool sizes achieve zero deviation from the utilization target, the pool size with smaller sum will achieve a lower objective value. But when w_s = 0 the inclusion of the summation term drives the algorithm towards smaller pool sizes that still meet the target service level or possibly even exhibit a slight deviation, since that effect may be outweighed by the increase of the $sum(pool_sizes[p+1:])$ term necessary to minimize the deviation.

5.0.3.2 Myopic Policy without Driver and Order Utility Terms

Figure 5.9 shows the performance of the myopic policy without driver and order utility terms. This analysis helps in understanding the impact of including these terms in the VFA algorithm.

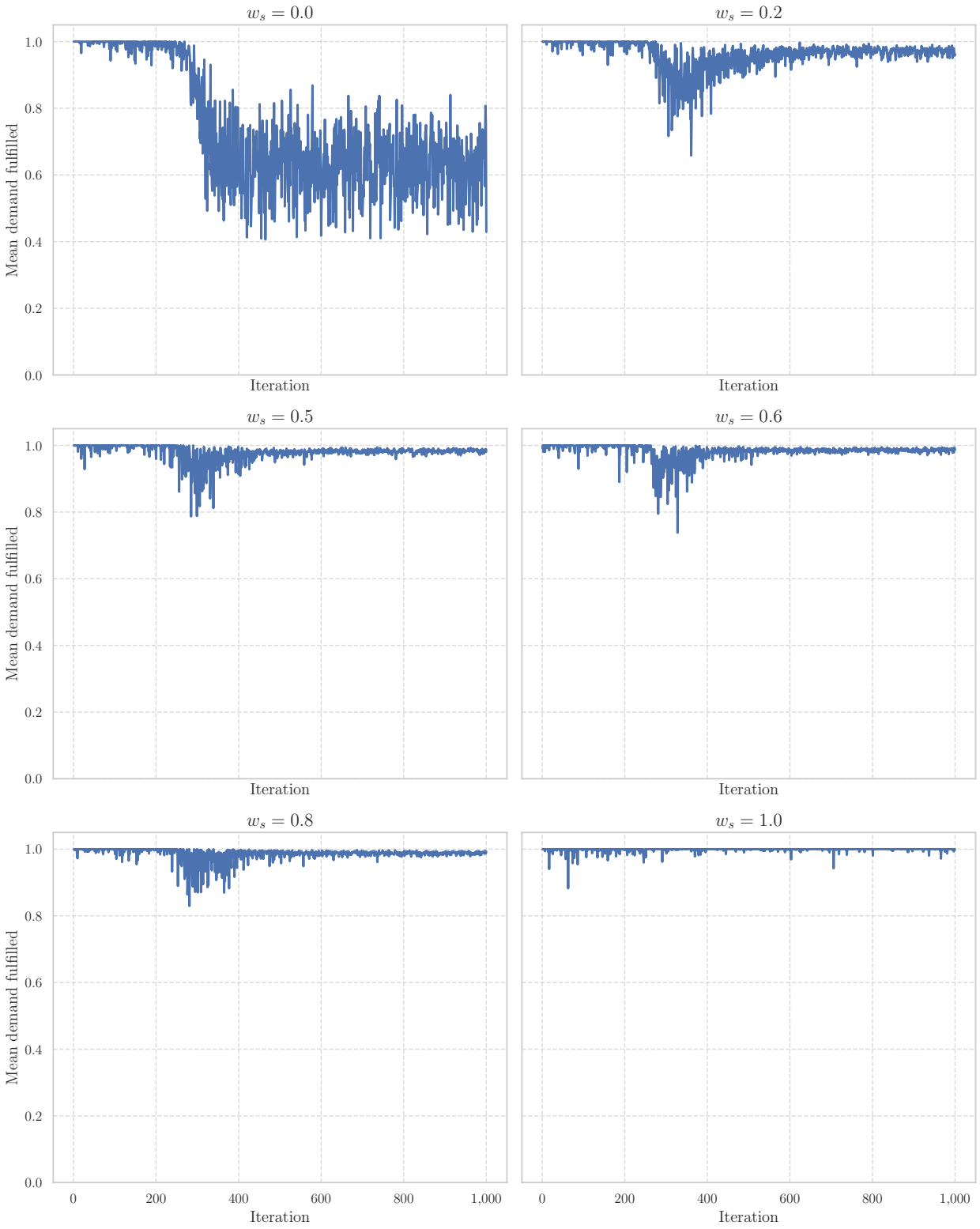


Figure 5.9: Performance of myopic policy without driver and order utility terms (Expected service level)

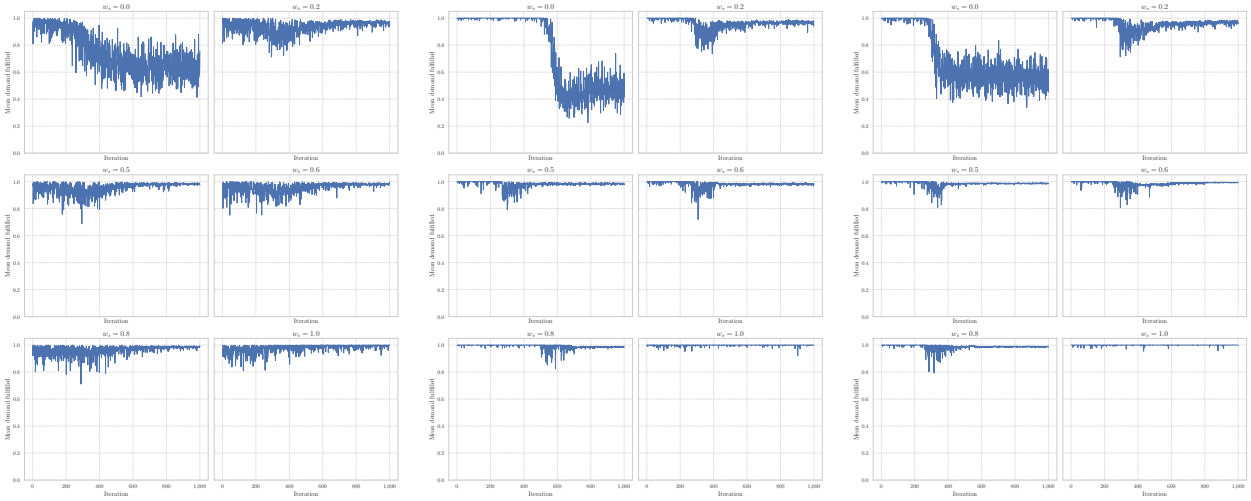
Even with driver and order utility terms excluded, both demand fulfilled and driver utilization still converge to values close to 1. The optimization algorithm has a platform profit term that still indicates that it is more profitable to perform a match as compared to a null assignment and the only question remains which drivers are most profitable to match, irrespective of driver or order fulfilment priorities. As a result, the platform profit is expected to increase significantly. The mean driver utilization will probably remain unaffected because a high number of matches is correlated with high platform profit and also driver utilization and demand fulfilment. However, it is likely that there will be significant variation in driver utilization as the optimization algorithm neglects driver utilization targets, thereby implying a lower matching fairness and a lower fraction of drivers meeting the guarantee. At the same time, it is also expected that the variation in both demand fulfilment and driver utilization across periods will exhibit significantly greater variation and be unstable.

However, despite the exclusion of the utility terms, the value function approximation algorithm still considers the demand fulfilment and driver utilization, applying a significant penalty to pools that lead to deviation from the targets. This conflicts with the optimization algorithm and is a more indirect method to achieve the targets. The optimization algorithm does not directly consider the driver utilization or service level targets and instead focuses on maximizing platform profit. However, the value function approximation algorithm completely ignores the objective value of the optimization algorithm (the net platform profit) and chooses the pool with the smallest sum that minimizes the deviation from the targets. As a result, the driver utilization and demand fulfilment targets are still expected to be met, although they will probably exhibit greater variability, but at the same time the best pool will achieve a higher platform profit.

Another question worth exploring is the effect of excluding the terms on the convergence of pool sizes.

5.0.3.3 Effect of Driver Arrival Probability

Figures 5.10 illustrates the impact of driver arrival probability on the expected service level. The analysis includes the case of compliant drivers (arrival probability = 1.0).

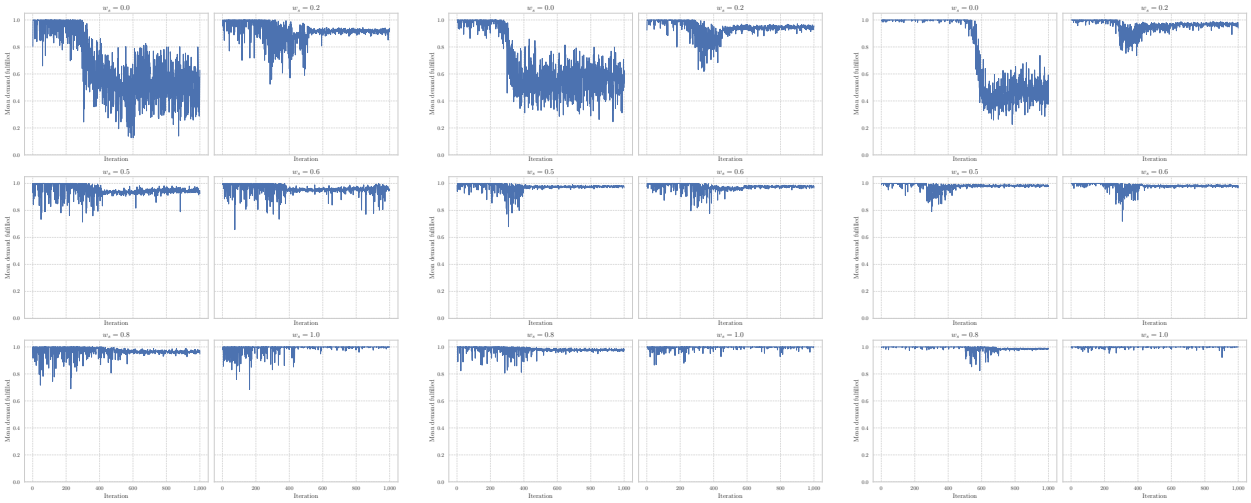


Arrival probability: 0.2 Arrival probability: 0.7 Arrival probability: 1.0

Figure 5.10: Impact of driver arrival probability on expected service level

5.0.3.4 Effect of Scheduling Period Duration

Figure 5.11 presents the relationship between the objective value and the weight parameter for varying scheduling period durations.



4-hour periods 2-hour periods 1-hour periods

Figure 5.11: Impact of scheduling period duration on expected service level

As the number of scheduling periods decreases, each period is of a greater length and drivers stay in the system for longer periods of time. This means that while it is more likely for a significant number of drivers to be available they may be busy and not available to fulfil orders, reducing

the service level, since drivers arrive with lower frequency and it is less probable that there are idle drivers. However, the algorithm iteratively optimizes the pool sizes to account for this, so that the pool sizes are larger to ensure that a sufficient number of drivers is available. At the same time, it is also important that this should not come at the expense of driver utilization. The Boltzmann exploration algorithm iteratively optimizes the pool sizes taking into account both of these competing but correlated factors.

5.0.3.5 Experimental Results: Constant Pool Size

The above plots approximate a convex function except for the case $w_s = 0$, which has a global minimum at a pool size of 1. The other plots have a global minimum above which they rise, and the gradient steadily decreases beyond the global minimum as the weight decreases. The form of the objective value metric is:

$$v_i = \sum_{j=1}^{n} p[j] + \text{penalty} \cdot \sum_{p=0}^{\text{periods}-1} \Big[w_s \cdot \max(Q - \text{Q}_p, 0) + w_d \cdot \max(W - \text{L}_p, 0) \Big]$$

This achieves a global minimum for the smallest sum of pool sizes that in expectation minimizes the sum of the deviations from the service level and driver utilization targets over each period. Once the global minimum is reached, increases in the pool size do not justify the marginal decrease in deviations, which causes a steady increase in the objective value. The gradient decreases as the weight increases because higher weights place more emphasis on minimizing the deviation from the service level target, while the deviation from the driver utilization target is negligible even for smaller weights because the service level is strongly correlated with driver utilization; it is not possible to achieve a high service level unless drivers are sufficiently utilized.

The variable pool sizes achieve a much better balance between driver utilization and service level; for example, in the extreme cases not necessarily achieving a value of 1.

Table 5.2: Comparison of Constant and Variable Driver Pools for Different Weights

w_s	Type of Driver Pool	Objective Value	Mean Demand Fulfilled	Mean Driver Utilization	Sum of Pool Sizes
0	Constant	176.0	0.548	1.000	176
	Variable	127.0	0.399	0.996	127
0.2	Constant	359.3	0.883	0.990	304
	Variable	355.4	0.928	0.969	331
0.5	Constant	405.6	0.928	0.967	336
	Variable	389.9	0.948	0.949	354
0.6	Constant	829.6	0.704	1.000	240
	Variable	411.1	0.951	0.941	361
0.8	Constant	433.6	0.976	0.888	400
	Variable	428.1	0.956	0.926	368
1	Constant	2560.0	1.000	0.191	2560
	Variable	1144.0	0.999	0.380	1144

5.0.4 Driver Matching Outcomes

5.0.4.1 Distribution of Driver Utilization

Table 5.3: Driver utilization statistics for different weight parameters

Weight	Mean	Variance	Min	Max
0.0	1.0	0.0	1.0	1.0
0.2	0.9661	0.0048	0.5	1.0
0.5	0.9375	0.0088	0.45	1.0
0.6	0.937	0.0085	0.45	1.0
0.8	0.9349	0.009	0.45	1.0
1.0	0.3725	0.0968	0.0	1.0

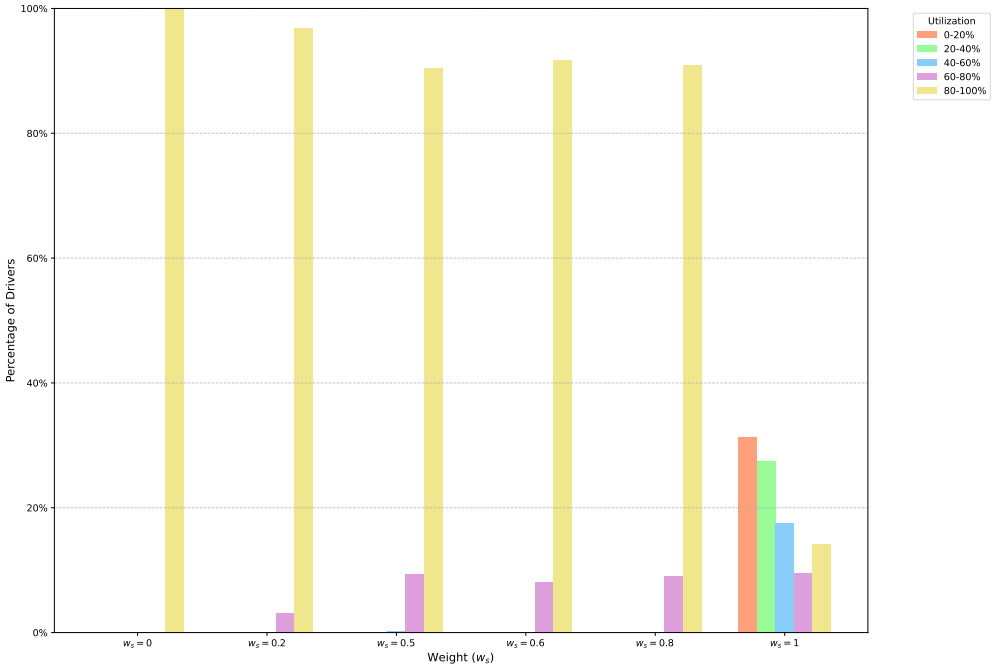


Figure 5.12: Distribution of driver utilization

5.0.4.2 Distribution of Driver Idle Time

Table 5.4: Driver idle time statistics for different weight parameters

Weight	Mean	Variance	Min	Max
0.0	0.0	0.0	0.0	0.0
0.2	2.3432	24.8219	0.0	40.5
0.5	4.2704	45.4725	0.0	42.75
0.6	4.2569	42.3705	0.0	40.5
0.8	4.3785	44.0226	0.0	42.8418
1.0	40.3708	503.1077	0.0	105.0

5.0.4.3 Distribution of Driver Empty Travel Distance

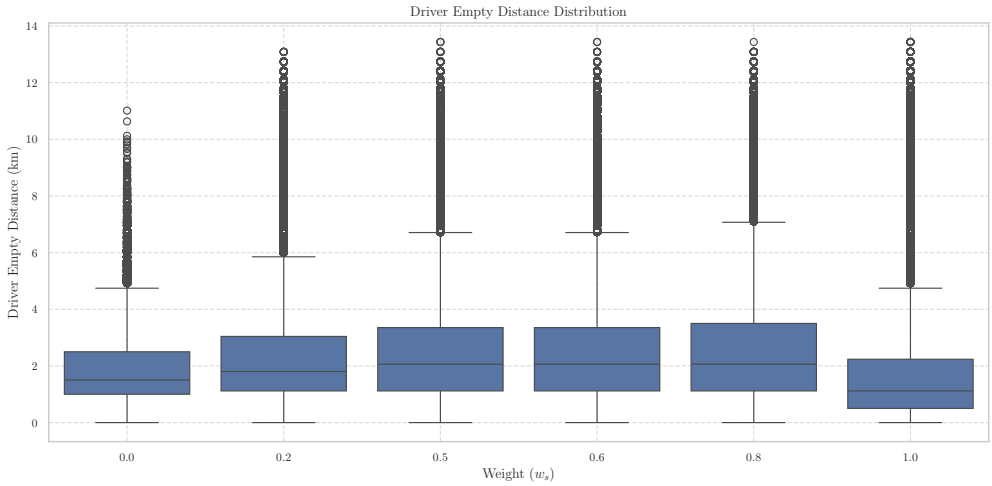


Figure 5.13: Distribution of driver empty travel distance for different weight parameters

Table 5.5: Empty travel distance statistics for different weight parameters

Weight	Mean	Variance	Min	Max
0.0	1.741	1.3477	0.0	11.0114
0.2	2.2745	3.0976	0.0	13.0863
0.5	2.5073	3.8054	0.0	13.435
0.6	2.5321	3.9522	0.0	13.435
0.8	2.5678	4.1977	0.0	13.435
1.0	1.9249	5.1202	0.0	13.435

5.1 Platform Profit Analysis

Table 5.6: Platform profit statistics for different weight parameters

Weight	Mean	Variance	Min	Max
0.0	6072.9188	133519.1239	5057.1917	7149.781
0.2	14132.2088	112420.3109	13106.834	15256.0096
0.5	14070.0998	118771.2211	13044.1627	15021.9166
0.6	14060.8725	123444.7525	12955.8172	14892.4352
0.8	14019.6012	134026.3156	13025.5303	14896.5387
1.0	14629.219	108126.3923	13505.6589	15372.9489

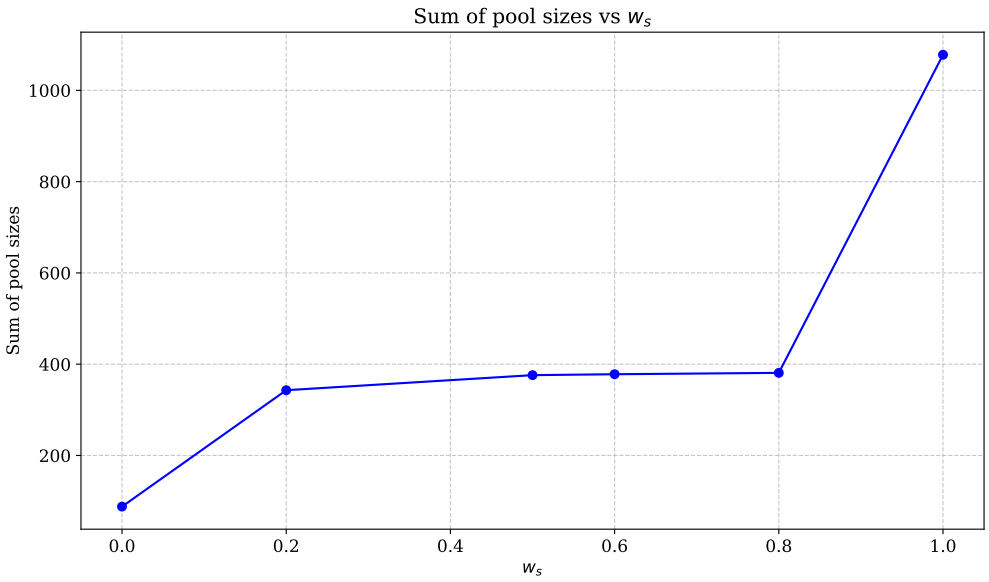


Figure 5.14: Pool sizes

5.1.0.1 Overall performance metrics for pool sum excluded

We performed an analysis with the pool summation term excluded. The results are summarized in the table below.

Table 5.7: Performance Metrics with Pool Sum Term Excluded

Metric	$w_s = 0$	$w_s = 0.2$	$w_s = 0.5$	$w_s = 0.6$	$w_s = 0.8$	$w_s = 1$
Sum of pool sizes	168	378	399	407	414	2043
Demand Fulfilled						
Mean	0.52	0.98	0.98	0.99	0.99	1.00
SD	0.03	0.01	0.01	0.01	0.01	0.00
CV	0.05	0.01	0.01	0.01	0.01	0.00
95% CI	(0.52, 0.53)	(0.98, 0.99)	(0.98, 0.99)	(0.99, 0.99)	(0.99, 0.99)	(1, 1)
Driver Utilization						
Mean	0.99	0.92	0.91	0.89	0.89	0.22
SD	0.05	0.10	0.11	0.12	0.11	0.27
CV	0.05	0.11	0.12	0.13	0.13	1.21
95% CI	(0.99, 0.99)	(0.92, 0.92)	(0.91, 0.91)	(0.89, 0.89)	(0.89, 0.89)	(0.22, 0.23)
Fraction Drivers Meeting Guarantee						
Mean	0.90	0.81	0.77	0.72	0.74	0.07
SD	0.29	0.26	0.26	0.24	0.25	0.02
CV	0.32	0.33	0.34	0.34	0.34	0.33
95% CI	(0.85, 0.95)	(0.76, 0.86)	(0.72, 0.82)	(0.68, 0.77)	(0.69, 0.78)	(0.06, 0.07)
Driver Empty Distance (km)						
Mean	1.91	2.31	2.58	2.62	2.71	1.65
SD	1.31	1.72	1.99	2.04	2.16	2.36
CV	0.69	0.75	0.77	0.78	0.80	1.44
95% CI	(1.90, 1.92)	(2.30, 2.31)	(2.57, 2.59)	(2.61, 2.63)	(2.70, 2.72)	(1.64, 1.66)
Platform Profit						
Mean	9294.89	14412.19	14044.79	14045.48	13920.13	15236.40
SD	403.91	342.58	334.65	359.94	375.27	352.59
CV	0.04	0.02	0.02	0.03	0.03	0.02
95% CI	(9215, 9375)	(14344, 14480)	(13978, 14111)	(13974, 14117)	(13846, 13995)	(15166, 15306)
Driver Idle Time (min)						
Mean	0.95	5.61	6.33	7.33	7.71	49.60
SD	3.37	7.12	7.82	8.35	8.20	22.18
CV	3.53	1.27	1.24	1.14	1.06	0.45
95% CI	(0.89, 1.01)	(5.53, 5.70)	(6.24, 6.42)	(7.23, 7.43)	(7.62, 7.81)	(49.49, 49.72)
Driver Time in System (min)						
Mean	65.45	65.56	66.05	65.91	66.00	64.07
SD	17.76	17.66	17.81	17.92	17.97	18.21
CV	0.27	0.27	0.27	0.27	0.27	0.28
95% CI	(65.13, 65.77)	(65.35, 65.78)	(65.84, 66.26)	(65.70, 66.11)	(65.79, 66.20)	(63.98, 64.16)

Appendices

Bibliography

[1] C. Archetti, M. Savelsbergh, and M. G. Speranza, "The vehicle routing problem with occasional drivers," *European Journal of Operational Research*, vol. 254, no. 2, pp. 472–480, 2016.

[2] L. Torres, M. Gendreau, and W. Rei, "Vehicle routing with stochastic supply of crowd vehicles and time windows," *Transportation Science*, vol. 56, no. 4, pp. 949–969, 2022.

[3] Q. Guo and N. Wang, "The vehicle routing problem with simultaneous pickup and delivery considering the total number of collected goods," *Mathematics*, vol. 11, no. 3, p. 729, 2023.

[4] M. K. Chen, "Dynamic pricing in a labor market: Surge pricing and flexible work on the uber platform," *Proceedings of the 2016 ACM Conference on Economics and Computation*, pp. 455–455, 2016.

[5] T. V. Le, A. Stathopoulos, T. Van Woensel, and S. V. Ukkusuri, "Designing pricing and compensation schemes by integrating matching and routing models for crowd-shipping systems," *Transportation Research Part B: Methodological*, vol. 146, pp. 1–21, 2021.

[6] G. P. Cachon, K. M. Daniels, and R. Lobel, "Surge pricing and its spatial supply response," *Management Science*, vol. 63, no. 2, pp. 429–445, 2017.

[7] C. S. Tang, "On the surge pricing of uber and other sharing economy platforms," *Operations Research*, vol. 64, no. 6, pp. 1287–1299, 2016.

[8] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and M. Hennig, "Workforce scheduling in the era of crowdsourced delivery," *Transportation Science*, vol. 54, no. 4, pp. 1113–1133, 2020.

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

Acronyms

AP Application Provider. *Glossary:* AP