Titre: Title:	Notification Timing for On-Demand Personnel Scheduling - Complexity and an Imitation Learning Approach
Auteur: Author:	Prakash Gawas
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gawas, P. (2025). Notification Timing for On-Demand Personnel Scheduling - Complexity and an Imitation Learning Approach [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/66110/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/66110/
Directeurs de recherche: Advisors:	Louis-Martin Rousseau, & Antoine Legrain
Programme: Program:	Doctorat en mathématiques de l'ingénieur

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Notification Timing for On-Demand Personnel Scheduling - Complexity and an Imitation Learning Approach

PRAKASH GAWAS
Département de mathématiques et de génie industriel

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

Mai 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Notification Timing for On-Demand Personnel Scheduling - Complexity and an Imitation Learning Approach

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

Michel GENDREAU, membre et présidente
Louis-Martin ROUSSEAU, membre et directeur de recherche
Antoine LEGRAIN, membre et codirecteur de recherche
Quentin CAPPART, membre
Tias GUNS, membre externe

DEDICATION

To my family and friends, whose unwavering support and love have been my guiding light.

To my mentors, whose wisdom and encouragement have inspired my journey.

To our beloved dog, who passed away on the very day I began my journey to Canada.

ACKNOWLEDGEMENTS

My deepest gratitude goes to my supervisors, Prof. Louis-Martin Rousseau and Prof. Antoine Legrain, for their invaluable guidance, support, and encouragement throughout my PhD journey. Their expertise and mentorship have been instrumental in shaping this research. I am particularly grateful to Prof. Louis-Martin Rousseau for helping me get started with the PhD program at Polytechnique Montreal. Special thanks to Prof. Antoine Legrain, whose meticulous attention to detail and constant drive for excellence have helped me grow immensely as a researcher. I also want to thank Guillaume Michaud and Dr. Khalid Laaziri for their technical assistance and ongoing support throughout my research.

I would like to express my heartfelt appreciation to my parents for their unwavering love, encouragement, and belief in my dreams of pursuing a PhD in Canada. I love you both. Thank you to my sister for not troubling me. I am forever grateful to my grandmother, who has been a constant source of inspiration and motivation in my life.

To my fellow PhD students and postdocs, who have become close friends over the years—thank you. Akash Sambrekar, exploring Montreal together when I first arrived was a joy. Francois Lamothe, thank you for introducing me to board games, ice skating, and climbing—activities that brought so much fun into my life. Flore Caye, thank you for your encouragement, for helping me find a dentist, and for making MTH6406 the most enjoyable course. Krunal Patel, thank you for sharing all things related to India. Qingwu Liu, thank you for organizing our badminton games and for the insightful discussions. El-Mehdi Mehiri, thank you for being my gym trainer and for your constant advice. Ruken and Ismail Sevim, thank you for being my family here in Montreal.

A special thanks to Warley Almeida, Jorge Moral, and Camille Pincon for your friendship and camaraderie—I deeply appreciate each of you. My travels were enriched by friends who hosted me and made me feel welcome in various places during my doctorate: Aashiq Shetgaonkar in the UAE, Indrajit Saha and Vivek Barsopiya in Japan, Vinay Rai and Sarvesh Gad in England, Akshay Bhave in Germany, and Benedetta Ferrari in Italy. These trips were refreshing and helped me recharge along the journey. To my roommates, thank you for maintaining a warm and welcoming atmosphere. Cooking with Aakash Sambrekar and Anamaria Pena was especially fun, and I treasure the wonderful conversations with Mairead Shaw.

I am deeply grateful to the committee members for taking the time to review my thesis and provide invaluable feedback.

Finally, I extend my heartfelt thanks to everyone who has contributed to my academic and personal growth, whether mentioned here or not. Your support has been a vital part of this journey. Thank you for being a part of it.

RÉSUMÉ

Les systèmes de service modernes s'appuient de plus en plus sur un large bassin d'employés occasionnels avec des horaires flexibles, payés à la pièce, pour répondre à la demande de travail sur appel dans des secteurs tels que le covoiturage, les livraisons et les plateformes de micro-tâches. Cependant, maintenir une prestation de service de haute qualité reste un défi, car les employés occasionnels ont souvent une implication peu fréquente et une expérience limitée. Cette étude présente un système de planification avancé, basé sur les données, qui donne la priorité aux employés occasionnels expérimentés, dans le but d'optimiser les opérations de service tout en minimisant les perturbations. Similaire aux systèmes de service sur appel traditionnels, notre approche contacte les employés par ordre d'ancienneté pour offrir des opportunités de quarts, ce qui permet aux travailleurs de choisir les quarts disponibles et même de « supplanter » les employés juniors si les quarts préférés ne sont pas disponibles. Bien que le supplantage soit autorisé, il peut entraîner une insatisfaction parmi les employés et une instabilité des horaires, créant un besoin de stratégies qui réduisent à la fois le supplantage et garantissent que les quarts sont pourvus rapidement.

Nous formalisons ce défi de planification comme un problème de synchronisation des notifications (NTP) et établissons qu'il est $\mathcal{NP}$-complet, même sous des hypothèses d'information parfaite. Pour remédier à l'incertitude des temps de réponse des employés, nous proposons un modèle stochastique en deux étapes pour la synchronisation dynamique des notifications, en développant une politique heuristique avec une structure basée sur des seuils. Cette politique est calibrée à l'aide de solutions hors ligne où toutes les incertitudes sont supposées connues, ce qui nous permet d'affiner les règles de décision pour les applications en temps réel. En optimisant le timing des notifications, notre approche réduit la probabilité de remplacements inutiles tout en garantissant l'affectation rapide des équipes.

De plus, nous appliquons une approche d'apprentissage par imitation en utilisant DAgger pour former de manière itérative un modèle prédictif qui capture la prise de décision des experts dans des scénarios d'affectation séquentielle des équipes. L'algorithme DAgger permet à la politique d'apprendre à partir de scénarios d'experts déterministes, créant ainsi un cadre de prise de décision efficace en temps réel. Cette intégration de l'optimisation et de l'apprentissage automatique s'ajuste dynamiquement aux états actuels du système, en équilibrant les notifications précoces et différées pour minimiser les perturbations du planning. Notre approche s'appuie sur un ensemble de modèles experts, notamment Hindsight, Aggregated Hindsight et un expert stochastique en deux étapes, pour informer l'algorithme

d'apprentissage avec diverses perspectives. En définissant ces multiples modèles experts, nous offrons une base flexible et personnalisable pour divers contextes opérationnels, permettant à l'algorithme d'apprentissage d'intégrer différentes perspectives d'experts pour améliorer la précision et l'adaptabilité des décisions.

Les résultats empiriques utilisant des données réelles de notre partenaire industriel démontrent la robustesse des politiques proposées, qui surpassent les méthodes heuristiques existantes en améliorant considérablement l'efficacité de la planification et la satisfaction des employés. Cette étude met en évidence le potentiel de la combinaison de la modélisation stochastique et de l'apprentissage par imitation pour relever les défis complexes de la planification induits par l'incertitude dans les systèmes de services à la demande, en apportant des méthodologies précieuses pour la prise de décision dynamique et en offrant une gamme d'options pilotées par des experts pour améliorer les performances du modèle dans diverses applications.

ABSTRACT

Modern service systems increasingly rely on a large pool of casual employees with flexible hours, paid on a piece-rate basis, to meet the demand for on-call work in sectors such as ride-sharing, deliveries, and microtask platforms. However, maintaining high-quality service delivery remains a challenge, as casual employees often have infrequent engagement and limited experience. This study introduces an advanced, data-driven scheduling system that prioritizes experienced casual employees, aiming to optimize service operations while minimizing disruptions. Similarly to traditional on-call systems, our approach contacts employees in order of seniority to offer shift opportunities, allowing flexibility for workers to choose available shifts and even "bump" junior employees if preferred shifts are unavailable. Although bumping is allowed, it can lead to dissatisfaction among employees and scheduling instability, creating a need for strategies that both reduce bumps and ensure shifts are filled promptly.

We formalize this scheduling challenge as a Notification Timing Problem (NTP) and establish that it is $\mathcal{NP}$ complete, even under the assumptions of perfect information. To address the uncertainty in employee response times, we propose a two-stage stochastic model for dynamic notification timing, developing a heuristic policy with a threshold-based structure. This policy is calibrated using offline solutions, where all uncertainties are assumed to be known, allowing us to refine decision rules for real-time applications. By optimizing notification timing, our approach reduces the probability of unnecessary replacements while ensuring the timely assignment of shifts.

Additionally, we apply an imitation learning approach using DAgger to iteratively train a predictive model that captures expert decision-making in sequential shift assignment scenarios. The DAgger algorithm enables the policy to learn from deterministic expert scenarios, creating an effective real-time decision-making framework. This integration of optimization and machine learning dynamically adjusts to current system states, balancing early and delayed notifications to minimize schedule disruptions. Our approach leverages an array of expert models, including Full-Information, Deterministic, Aggregated Deterministic, and a Two-Stage Stochastic expert, to inform the learning algorithm with diverse perspectives. By defining these multiple expert models, we offer a flexible, customizable foundation for various operational contexts, enabling the learning algorithm to incorporate different expert perspectives to improve the accuracy and adaptability of the decisions.

Empirical results using real-world data from our industry partner demonstrate the robustness

of the proposed policies, which outperform existing heuristic methods by significantly improving both scheduling efficiency and employee satisfaction. This study highlights the potential of combining stochastic modelling and imitation learning to tackle complex uncertainty-driven scheduling challenges in on-demand service systems, contributing valuable methodologies for dynamic decision-making and offering a range of expert-driven options to enhance model performance across diverse applications.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

SDP	Sequential Decision Problem
PFA	Policy Function Approximation
VFA	Value Function Approximation
ML	Machine Learning
MIP	Mixed Integer Program
OR	Operations Research
RL	Reinforcement Learning
IL	Imitation Learning
BC	Behavior Cloning
IRL	Inverse Reinforcement Learning
CO	Combiantorial Optimization
E2EL	End-to-End Learning
NTP	Notification Timing Problem
ONP	Offline Nonparametric Policy
DAgger	Dataset Aggregation
MTurk	Amazon Mechanical Turk
TSP	Traveling Salesman Problem
DQN	Deep Q Network
MDP	Markov Decision Process
CFA	Cost Function Approximation
UCT	Upper Confidence Bound for Trees
MCTS	Monte Carlo Tree Search
LfD	Learning from Demonstrations
Maxent-IRL	Maximum Entropy Inverse Reinforcement Learning
PCA	Principal Component Analysis
SEARN	Search-based Structured Prediction
SMILe	Stochastic Mixing Iterative Learning
GBDT	Gradient-Boosted Decision Trees
FIE	Full-Information Expert
DE	Deterministic Expert
2SSE	Two-Stage Stochastic Expert
ADE	Aggregated Deterministic Expert

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Machine Learning (ML), a subset of artificial intelligence, focuses on creating algorithms and statistical models that enable computers to perform tasks without explicit human instructions. These systems improve performance by learning from data, identifying patterns, and making predictions. They also encompass techniques such as supervised, unsupervised, and reinforcement learning. ML has demonstrated its transformative potential across diverse fields, from improving medical diagnostics and automating industrial processes to powering recommendation systems and optimizing resource management. ML fundamentally reshapes industries and drives innovation by offering intelligent and adaptable solutions.

Many real-world challenges involve decision-making under uncertainty, where critical parameters influencing the system are either unknown or only partially observable. Examples include planning optimal delivery routes in congested urban areas, scheduling hospital personnel in dynamic and unpredictable environments, or allocating power generation to meet fluctuating energy demands. Estimating these uncertainties accurately is inherently difficult. Traditionally, machine learning (ML) models have been utilized to predict uncertain parameters, which are subsequently incorporated into mathematical optimization frameworks to derive optimal decisions. However, such optimization problems are often computationally demanding, particularly when dealing with high-dimensional data or complex problem structures. Even with state-of-the-art solvers, finding solutions within a reasonable time frame remains a significant challenge. This has necessitated the development and application of approximation techniques to enhance the tractability of these problems.

To address these computational challenges, we explore an ML-based approach that replaces the expensive optimization process with a data-driven model capable of learning optimal decisions directly from expert demonstrations. This approach leverages imitation learning (IL), a methodology commonly used in robotics and control systems, where models are trained to replicate the behavior of an expert by observing its actions in different scenarios. By bypassing the need to solve optimization problems repeatedly, the proposed method offers a scalable and efficient alternative to solve SDPs under uncertainty.

In this work, we apply our imitation learning framework to a novel personnel scheduling problem. Decision-making involves dynamically assigning shifts to employees while accounting for constraints and uncertainties. By focusing on this operationally significant domain, we demonstrate the potential of imitation learning to streamline complex scheduling tasks, reduce computational costs, and improve decision quality in real-world applications.

This chapter is organized as follows. Section 1.1 covers the necessary background on SDP, ML, and IL. Section 1.2 describes the research objectives of the thesis. Finally, Section 1.3 presents the thesis outline.

1.1 Background

This section provides the foundational concepts required to understand the ideas and methods explored in this thesis. We first introduce Sequential Decision Problems (SDPs) and Markov Decision Processes (MDPs) Section 1.1.1, laying the groundwork for understanding decision-making under uncertainty. These frameworks are essential for modelling and solving problems that involve making a sequence of decisions over time. Section 1.1.2 offers a concise overview of Machine Learning (ML), focusing on its role in solving real-world problems and its relevance to the approaches discussed in this work. Together, these sections establish the basic theoretical and practical context necessary for the subsequent chapters.

1.1.1 Sequential Decision Problems

Sequential decision-making [Powell, 2007] is a fundamental concept in mathematics and operations research, addressing scenarios where decisions must be made in a sequence over time, often under conditions of uncertainty. This process involves determining a series of actions that optimize a particular objective, taking into account the evolving state of the system and the information available at each decision point. As actions are taken, rewards/costs are gained, which contribute to the overall objective.

In mathematics, the study of sequential decision-making encompasses a range of theories and methods, including dynamic programming and optimal control [Bertsekas, 2012], Markov decision processes [Puterman, 1990], and stochastic optimization [Birge and Louveaux, 2011]. These techniques provide frameworks for modelling and solving problems in which the outcome of each decision influences future choices and their potential outcomes. By systematically evaluating possible strategies and their consequences, sequential decision-making helps identify optimal paths to achieve the desired goals.

This area of study has wide-ranging applications, such as in inventory management, where companies must decide on stock levels over time to meet demand while minimizing costs; in robotics, where autonomous systems must navigate environments and make real-time decisions to achieve tasks; and in finance, where investors need to make sequential investment choices to maximize returns while managing risk. The mathematical foundations of sequential decision-making provide the tools and insights needed to tackle these complex and dynamic

problems effectively.

Modelling Framework

We describe a general modelling framework for SDP from [Powell, 2022]. The core elements are

- Decision Epoch (k) - A decision epoch is when a decision maker can see the current state of the system and has to make a decision. Let K be the final decision epoch. This defines whether the problem is a finite ($K < \infty$) or infinite ($K = \infty$)horizon problem. In this thesis, we will focus on finite-horizon problems.

- State variables (x_k) - The state variable contains everything we know and only what we need to know to make a decision and model our problem. This state information can include physical state variables (inventories), parameter information, contextual information (weather and prices) and beliefs in the form of probability distributions. The set of all possible states is called a state space. We will use x to denote a general state with the epoch index.

- Decision variables (a_k) - A decision variable can be binary, a discrete set, a continuous variable, or a vector of discrete and continuous variables. Decisions are subject to constraints $a_k \in A_k$.

- Exogenous information (ξ_k) - This is the information that we learn after we make a decision, which we do not know when we make a decision. Exogenous information comes from outside whatever system we are modelling.

- The transition function ($X^M(x_k, a_k, \xi_k)$) - The function consists of the equations required to update each element of the state variable. This covers all the dynamics of our system, including the updating of estimates and beliefs for SDPs.

- The objective/value function - This consists of the contribution (reward or cost) the decision maker earns each decision epoch, given by $C_k(x_k, a_k)$, where $a_k = \pi(x_k)$ is determined by a policy. Assuming that one wants to minimise costs, we can write the function as

$$V^\pi(x_0) = \min_\pi \mathbb{E}\left\{ \sum_{k=0}^{K} C_k(x_k, \pi(x_k)) | x_0 \right\} \tag{1.1}$$

where x_0 is the initial state of the problem. Note in Equation (1.1), the discount factor for future costs has been omitted primarily with a focus on finite horizon problems. The goal is to find the best policy that optimises our objective. In MDP literature, $V^{\pi}(x_k)$ is referred to as the value function (optimal cost-to-go) with respect to the policy π from the state x_k at epoch k.

- Policy $\pi(x_k)$ - A policy is a mapping from any state x_k to a corresponding action that can be taken in that state. A policy can be a tabular function or an algebraic function with tunable parameters θ. The optimal policy π^* is given by

$$\pi^* = \arg\min_{\pi} \mathbb{E}\left\{ \sum_{k=0}^{K} C_k(x_k, \pi(x_k)) | x_0 \right\} \tag{1.2}$$

$$\pi^* = \arg\min_{\pi} V^{\pi}(x_k) \tag{1.3}$$

  One can derive the optimal policy if the value functions are known by doing the greedy one-step search. A value function for a given policy can be derived by using dynamic programming algorithms [Puterman, 1990]. Equation (1.3) can be solved using policy iteration and value iteration algorithms [Puterman, 1990]. Alternatively, one may be only interested in the next optimal action denoted by a_k^* at some point k in the planning period. Equation (1.4) defines a_k^*. Here, $V_{k+1}(x_{k+1})$ denotes the optimal cost from the next state x_{k+1}.

$$a_k^* = \arg\min_{a_k} \left\{ C_k(x_k, a_k) + \mathbb{E}[V_{k+1}(x_{k+1})] | x_k \right\} \tag{1.4}$$

Ideally, given all states, actions, and possible outcomes, equation (1.3) would be solved offline to compute the value function. The policy obtained from this solution could then be applied online to make sequential decisions. However, the primary challenges in solving equation (1.3), even in an offline setting, are well-documented in [Powell, 2007] and are commonly referred to as the three curses of dimensionality. These challenges arise from the large state space, the extensive outcome space of the uncertainty, and the vast action space, rendering equation (1.3) intractable. Consequently, approximate methods must be employed to address this complexity.

Various classes of approximate methods for policy search are thoroughly explored in [Powell, 2022]. These methods include:

- **Policy Function Approximations (PFA):** Analytical functions that directly map

a state (encompassing all available information) to a decision.

- **Cost Function Approximations (CFA):** Policies incorporating an embedded optimization problem, where parametric adjustments are applied to the objective function or constraints.
- **Value Function Approximations (VFA):** Approaches that estimate the value function approximately rather than calculating it exactly, enabling decision-making in the current state.
- **Direct Lookahead Policies (DLA):** Strategies that optimize over a specified timeframe to account for the impact of current decisions on future outcomes, facilitating informed decision-making in the present.

The reader is encouraged to read [Powell, 2022] for details. This dissertation will focus on learning a PFA, which will be discussed in more detail in Chapter 2.

1.1.2 Machine Learning

Machine learning (ML) is a foundational discipline within the field of artificial intelligence, with the primary goal of designing algorithms that enable computational systems to solve complex problems by learning from data, observations, or experiences rather than through explicitly programmed instructions. This paradigm shift from rule-based programming to learning-based approaches has enabled ML to address tasks of increasing complexity across a diverse range of domains, from natural language processing to decision-making under uncertainty. By leveraging patterns in data, ML algorithms can generalize beyond observed examples, providing predictive and adaptive solutions.

ML algorithms are generally categorized into three primary paradigms: **Supervised Learning**, **Unsupervised Learning**, and **Reinforcement Learning**. Each paradigm addresses a unique class of problems and employs distinct methodologies. Although this thesis primarily focuses on supervised learning, it is essential to briefly outline all three paradigms to situate the discussion within the broader ML landscape. Readers seeking an in-depth treatment of these paradigms are encouraged to refer to established references such as [Hastie et al., 2001], [Bishop and Nasrabadi, 2006], and [Goodfellow et al., 2016].

In supervised learning, a dataset $\mathcal{D}$ is used to train a model (classifier) that will make predictions on unseen data. This data set includes training features and the corresponding target labels or values. The problem is termed classification if the targets are categorical and regression if the targets are numerical values. The main objective is to predict the target

label or value for data outside the training set. This is achieved by learning a mapping from inputs to targets so that the targets of the new examples can be approximated as accurately as possible. These approximations can be evaluated and optimized using error functions like the least-squares error or cross-entropy error.

The primary objective of machine learning (ML) is to develop a model that can approximate underlying patterns in data and generalize effectively to unseen data. However, achieving this goal often involves navigating the trade-off between *bias* and *variance*, which are key determinants of model performance.

Consider a highly complex model, such as a high-degree polynomial attempting to approximate a linear function with minor noise. Such a model might perfectly fit the training data by minimizing the error function and capturing every nuance, including noise. However, this can lead to *overfitting*, where the model becomes overly reliant on the training data and fails to generalize to new, unseen examples. This phenomenon is characterized by *high variance*—a model's predictions vary significantly when trained on different subsets of data.

Conversely, simpler models, such as linear functions or shallow decision trees, are often less prone to overfitting due to their limited complexity. However, these models may lack the expressive power needed to capture the intricate relationships in the data, resulting in **underfitting**. Such models exhibit *high bias*, as they oversimplify the data's underlying structure, leading to consistently inaccurate predictions.

The trade-off between bias and variance necessitates striking an appropriate balance to build a model capable of generalizing well. To achieve this, ML models incorporate *hyperparameters*, which are distinct from the parameters learned during training. Hyperparameters control aspects of the model architecture or learning process, influencing its bias and variance. For instance, in decision trees, the maximum depth of the tree is a hyperparameter; deeper trees tend to reduce bias but increase variance, while shallower trees do the opposite.

To facilitate effective training and evaluation, the available data is typically divided into three sets:

- **Training Set:** Used to learn the model's parameters by minimizing the error function.
- **Validation Set:** Used to evaluate the model's performance on unseen data and fine-tune hyperparameters to optimize generalization.
- **Test Set:** Reserved for the final evaluation to measure how well the model generalizes to completely new data.

This systematic division ensures that hyperparameter tuning does not inadvertently lead to

overfitting, providing a reliable estimate of the model's real-world performance. By carefully managing the balance between bias and variance and leveraging techniques such as hyper-parameter optimization, ML models can achieve robust and adaptive solutions for diverse tasks.

In unsupervised learning problems, there is no target to predict. The main goal is to extract information from the data by analyzing the training dataset. For example, principal component analysis (PCA) processes high-dimensional input data. It selects a fixed number of directions (k) to project the data, thereby reducing its dimensionality while retaining as much information as possible. Similarly, clustering algorithms like K-means identify underlying categories in the input data by detecting clusters. This thesis does not delve further into unsupervised algorithms; for a more detailed description, readers are referred to chapter 14 of [Goodfellow et al., 2016].

In contrast, reinforcement learning (RL) deals with decision-making problems where one or more agents interact with an environment. The agent learns to choose optimal actions in a given state to maximize cumulative rewards over time. Unlike supervised and unsupervised learning, RL does not rely on labelled training data. Instead, it employs an iterative process of *exploration* (trying different actions to discover their outcomes) and *exploitation* (choosing the best-known actions based on past experiences).

For instance, an RL agent learning to play chess might initially make random moves, gradually improving its strategy through self-play or matches against other agents by observing the outcomes (wins, losses, or draws) as rewards. Over time, the agent refines its policy—a mapping from states to actions—by maximizing the expected reward. Reinforcement learning has been applied successfully in domains such as robotics, game-playing, and autonomous systems, where sequential decision-making is crucial. For a comprehensive exploration of reinforcement learning principles and methodologies, readers are referred to [Sutton and Barto, 2018].

Imitation Learning

Imitation learning [Ho and Ermon, 2016], [Hussein et al., 2017] is a branch of supervised machine learning where an agent learns to perform tasks by mimicking the actions of an expert. Rather than learning through trial and error or from direct instructions, the agent observes and replicates behaviours demonstrated by a human or another proficient system. These demonstrations serve as training data, showing the agent the desired behaviour in various situations. This approach is inspired by the way humans and animals learn by observing others, leveraging the concept that complex tasks can be more efficiently learned

through demonstration rather than exploration, as in reinforcement learning. This approach is also referred to as Learning from Demonstrations (LfD). The agent's goal is to learn a policy, a function that maps observed states to actions, which closely matches the expert's behaviour. Imitation learning has a wide range of applications, including:

- Robotics - Teaching robots to perform tasks such as grasping objects, navigating environments, or assembling components by observing human operators.
- Autonomous Driving - Enabling self-driving cars to learn driving behaviours by imitating human drivers.
- Game Playing - Training AI agents to play complex games by watching expert players.
- Healthcare - Assisting in surgical procedures by learning from expert (surgeon) actions.

While imitation learning offers an intuitive approach to training agents, it also presents several challenges.

- High-quality, comprehensive demonstrations are necessary for effective learning.
- It is crucial to ensure that the agent can generalise from observed behaviours to new, unseen situations.
- Mistakes made during the imitation process can accumulate, leading to degraded performance over time.
- The quality of the learned policy is heavily dependent on the expertise of the demonstrator.

Research in imitation learning continues to explore methods to overcome these challenges, with a focus on improving generalisation, reducing the reliance on large amounts of demonstration data, and integrating imitation learning with other learning paradigms like reinforcement learning and self-supervised learning. Imitation learning is a promising approach to developing intelligent systems capable of performing complex tasks by leveraging the knowledge and skills of human experts. As technology advances, it is expected to play a crucial role in the deployment of autonomous systems in various real-world applications.

1.2 Research Objectives

The primary objective of this dissertation is to explore imitation learning for designing a Policy Function Approximation (PFA). Specifically, the goal is to define an analytical function

that maps a state (including all available information) to a decision, enabling a decision-maker to directly use this function to select actions based on current state information. Training the model in a supervised manner requires expert demonstrations provided by π^*. For any state x_k, the corresponding expert demonstration is $a_k = \pi^*(x_k)$. These expert demonstrations can be derived by solving (Equation (1.3)). However, as noted in Section 1.1.1, solving (Equation (1.3)) offline is computationally expensive. To address this, solutions from simpler optimization problems which are solved offline are used as a "pseudo" expert. In operations research (OR), an **offline** solution under uncertainty assumes full or partial knowledge of all relevant variables and outcomes, forming an optimal decision-making strategy.

The approach is tested on a novel Notification Timing Problem (NTP), where the decision-maker must balance two competing objectives through sequential decisions. This problem is characterized by two curses of dimensionality: a large state space and a large outcome space, though the action at each decision epoch is single-dimensional. The aim is to learn a model that predicts the action using full-information solutions as targets. Since the full-information problem is not yet defined, the complexity of the full-information problem is first examined, followed by the development of heuristic policies based on aggregated full-information solutions.

Within IL, we use behaviour cloning to learn a PFA. In standard behavior cloning, a model learns a policy by mimicking expert actions on a set of states. However, it doesn't encounter or learn from states that arise due to its own mistakes at test time. As these errors accumulate, the learned policy can deviate significantly from the expert's behavior. DAgger (Dataset Aggregation) [Ross et al., 2011] can be used to address the compounding error problem. The DAgger algorithm first starts with an initial policy, performs rollout with Expert Corrections, aggregates data, trains a new model and repeats the process till we find a good policy.

The contributions of this thesis are outlined below, highlighting the novel elements introduced to the scientific literature:

1. An introduction of a novel and significant problem related to timing job opening notifications to employees for last-minute shift assignments in Personnel Scheduling, specifically within dynamic on-demand labor platforms. To our knowledge, this is the first study to focus on optimizing the operations of a dynamic on-demand scheduling system, where employees can decide whether they want to work in response to the notifications sent. The objective of the problem is to strike a tradeoff between quick shift assignment and employee inconvenience caused by the on-demand system.

2. It is then shown that even a simple and offline version of the problem with complete

information is $\mathcal{NP} - complete$. We show a reduction from the subset sum problem by giving examples, intuition, and proof.

3. The (real) stochastic and dynamic version of the same problem is considered, showing that any dynamic policy may perform significantly worse than its offline counterpart. An operating policy called the ONP is developed to solve a two-stage stochastic programming formulation heuristically. This methodology aggregates offline solutions to form a nonparametric policy function.

4. Performance is compared against heuristic policy functions, demonstrating that the offline solution-based policy outperforms them using real-world data from our industrial partner. Results are extended by testing various employee shift preference distributions, showing that the offline policy consistently performs better.

5. With an aim to define ML algorithms based on imitating an expert based on mathematical optimization problems we define a taxonomy for the expert. This will help the reader understand the classification of the experts based on the types, usage and targets they are used to learn.

6. The NTP is then tackled by employing ML-driven operating policies learned by using the DAgger algorithm. The ML model is trained using an expert function derived from mathematical optimization, effectively emulating the expert policy without imposing the computational burden during online operations. Experiments are performed using different expert functions within the DAgger framework. The results indicate that the machine-learned model outperforms the ONP.

1.3 Outline

In Chapter 2 we present a broad literature review on various topics. Chapter 3, first gives a complete description of the dynamic Notification Timing Problem faced on an on-demand labor platform. We first consider a simpler static version of this problem under complete or full information of the uncertainty. This problem is designed to show the complexity of the problem even when have complete knowledge of the uncertainty. This problem is shown to be $\mathcal{NP} - complete$ by reducing the Subset sum problem to the NTP. We provide the reader with intuitions, examples to support the proof. We also formulate the problem as a MIP. This MIP is simple and has an elegant structure.

Chapter 4, considers the dynamic NTP which is an SDP. We approximate this problem using a two-stage stochastic formulation. This stochastic formulation is quite complex to solve in

real life. We work on a heuristic way to solve the problem. We design an algorithm where one solves a number of full information instances offline and then aggregates the solution obtained. This algorithm provides an efficient way to solve the stochastic formulation by breaking down the formulation into individual scenarios. We then simulate the policy obtained ONP on unseen test instances. The ONP is compared with the current policy in use by our industrial partner and also against the policy obtained from a tractable two-stage formulation.

In Chapter 5, we provide a critical reflection on the nature and roles of experts in imitation learning algorithms by proposing a comprehensive taxonomy. This taxonomy categorizes various attributes of experts, including the types of experts available, usage strategies, and other influential characteristics. By organizing these insights, we gain a clearer understanding of the strengths and limitations of the chosen expert within our algorithm.

Chapter 6 introduces machine learning-based policies developed through behavior cloning. To achieve this, we use solutions obtained from the Mixed Integer Program (MIP) designed in Chapter 3 as the expert to be cloned. However, this expert has inherent biases and imperfections, given that it relies on access to all future information—a perspective that does not align well with real-world constraints, making it a suboptimal candidate for direct imitation. To address this limitation, we enhance the learning process by invoking this MIP-based expert multiple times, each time under different future scenario assumptions. This approach enables us to present refined results derived from this scenario-based learning method.

1.4 Dissemination of Research

This thesis represents the contents of three research articles. Chapter 3 and Chapter 4 form the first article which is currently submitted to POMS. Chapter 5 and Chapter 6 form the basis of the third article, which I am currently working on. Then I have an accepted paper titled 'An Imitation-Based Learning Approach Using DAgger for the Casual Employee Call Timing Problem' in the International Conference on Learning and Intelligent Optimization. The proceedings of the conference are published in LNCS, volume 14990. Besides, I have presented my research at JOPT 2022, JOPT 2023, JOPT 2024, JOPT 2024, CPAIOR 2024, LION 2024.

CHAPTER 2 LITERATURE REVIEW

This chapter provides a comprehensive review of the relevant literature. Section 2.1 introduces the domain of the problem studied within the context of On-Demand Labor Platforms, setting the stage for subsequent discussions. The chapter then delves into research within the field of sequential decision making in operations research (OR), with a focus on key topics such as learning paradigms, the study of different policy classes, and contextual optimization. Section 2.2 explores the literature on reinforcement learning (RL) and supervised learning methodologies for developing solution approaches to SDP, with particular emphasis on various imitation learning (IL) algorithms. Section 2.3 reviews the use of policy function approximations (PFAs) in SDP, which form the foundation of our proposed solution approach. Finally, Section 2.4 examines the integration of machine learning (ML) and operations research (OR), highlighting their synergies in addressing complex decision-making problems.

2.1 Problem Introduction

In various service industries such as finance, restaurants, transportation, retail, and call centers, two crucial factors that influence service quality are the support provided by experienced employees and the speed of service. However, many of these systems experience fluctuating demand and have limited capacity, making it challenging to accurately determine the number of employees needed to efficiently serve customers in advance. This unpredictability can lead to either understaffing or overstaffing, both of which negatively impact customer service levels, employee satisfaction, and operating costs. To address this, companies typically plan their regular shift schedules weeks in advance to meet only the baseline demand. To manage uncertainty, they often turn to cost-effective solutions like overtime or part-time employment.

The gig economy has recently emerged as a rapidly growing business model over the past decade [Graham and Woodcock, 2019], [Donovan et al., 2016], offering an alternative approach for industries. It operates as an on-demand labor platform, where workers take on short-term tasks or freelance projects rather than traditional full-time employment. These workers, known as freelancers, casual workers, contingent workers, gig workers, or independent contractors, participate in platforms such as ride-sharing services like Uber and Lyft, food delivery platforms like GrubHub, Seamless, and InstaCart, customer contact centers like Liveops, home services like Handy, and web development and data analysis platforms like Upwork and Fiverr. Such for-profit companies operate as on-demand platforms [Allon

et al., 2023], using technology to meet short-term labor needs by connecting businesses with a flexible, crowd-sourced workforce. A key feature of these platforms is their ability to tap into a large, crowd-sourced pool of agents to manage demand spikes while controlling labor costs through variable pay instead of fixed wages. For statistics on gig workers, readers are referred to [Donovan et al., 2016]. In this work, in collaboration with our industrial partner Merinio, we focus on optimizing the operations of a system that enables service-providing companies to cover last-minute staffing needs with on-demand casual workers. This system aims to provide the most experienced employees from the pool, ensuring quality work and fostering a trusting relationship with a select group of workers.

2.1.1 On-Demand Labor Platforms

Any on-demand platform system is built around four key elements: (1) the task, which is the specific job to be completed; (2) the client, also known as the publisher or seeker, who could be an individual or a company responsible for posting tasks; (3) the worker, who has the required knowledge, skills, and abilities to complete various outsourced tasks; and (4) the platform, which serves as the intermediary between the client and the worker, providing tools to manage and organize the entire process, and sometimes even handling certain tasks on behalf of the clients. Workers begin by creating online profiles on the platform that show their skills, experience, and rates. These workers are typically freelancers or casual personnel seeking flexible work opportunities. Clients register on the platform to post task or job listings that detail the projects with which they need assistance, including scope, budget, and deadlines. These job listings can be made accessible to all relevant workers or restricted to a specific subset. Workers then apply for the positions posted by the clients.

A distinctive feature of this system is the nature of employment: independent workers have the freedom to choose their work schedules and can effortlessly switch between multiple platforms to offer their services. This flexibility is a major draw for workers in the gig economy. Companies also benefit from increased labor flexibility, enabling them to hire workers with varying skill levels at different times and compensate them accordingly.

However, these online labor platforms face several domain-specific challenges [Slivkins and Vaughan, 2014], [Bhatti et al., 2020]. These challenges include task design, pricing mechanisms, client and worker evaluation, and, most crucially, task allocation or worker-client matching. Each entity involved has different objectives. Workers seek jobs that provide a steady income without compromising their work-life balance. Clients aim to receive the highest quality work within their budget constraints. Platforms focus on the short-term goal of retaining both workers and clients, while their long-term objective is to generate consistent

profit. Addressing all these challenges in a single coherent model is highly complex, which is why different platforms employ varying approaches.

Consider task allocation as an example, which will be the focus of this paper. On some platforms, like Fiverr and Upwork, workers actively search among posted projects with diverse scopes and requirements, choosing which ones to pursue by submitting applications and initial bids that propose either a fixed project price or an hourly rate. On other platforms, clients set the compensation rates, but workers still have some flexibility in choosing tasks that appeal to them. For instance, MTurk workers search among available micro-tasks, which are advertised with piecework compensation rates set by clients. On many location-based labor platforms, task assignments are determined by algorithms, with pay rates set by the platform. For example, an Uber or Lyft driver receiving a ride request has only a few seconds to decide whether to accept it, often without knowing the passenger's final destination. Similarly, on TaskRabbit, workers—known as taskers—typically have a limited window to respond to job offers, ranging from a few minutes to a few hours depending on the urgency and client preferences. Task allocation has garnered significant attention in recent years [Karachiwalla and Pinkow, 2021], [Zhen et al., 2021].

The COVID-19 pandemic in 2020 triggered widespread job losses, followed by significant labor shortages as economies began to recover [Causa et al., 2022]. Several factors contributed to these shortages, including the rigidity of traditional employment systems, which made it difficult to retain a stable workforce. To address these challenges, our industrial partner, Merinio, developed an innovative flexible scheduling system. This system introduces greater flexibility and adaptability into shift scheduling by maintaining a large pool of occasional or casual employees who are available for on-call work. Unlike traditional on-call systems, these employees are not contractually obligated to accept every assignment—they have the freedom to decline shifts or choose not to respond at all. This flexibility marks a significant departure from conventional practices.

Each employee in this pool has specific skills that qualify them for certain shifts. Management starts by determining the number of shifts needed for a particular day in the near future, and this information is entered into an electronic call system. The system then compiles a list of eligible employees based on their skills and availability, and an operating manager activates the communication process. The electronic call system manages the notification of employees about available shifts, tracks their responses, and updates the shift schedule as employees accept assignments. Workers can accept shifts that they find suitable, ensuring that labor resources are allocated efficiently while respecting employee preferences and availability. Similar platforms include Wonolo, GigSmart, and Shiftgig.

In this system, the jobs available within a service company are represented by shifts that require specific skills to be performed within a fixed time window. For instance, in an amusement park, shifts might include roles such as ticket checkers, cleaners, or waitstaff. These positions generally require low skill levels and offer nominal pay, which is already known to casual workers. The service company aims to ensure the highest quality and speed of service for its customers. Ensuring this with casual personnel can be challenging due to limited screening, inconsistent engagement, and low pay. In addition, some casual workers may face motivation and communication issues. On online platforms like Fiverr and Upwork, workers with the best reputations tend to secure jobs [Yoganarasimhan, 2013], as they are perceived to be more reliable and capable of delivering high-quality work. Conversely, workers with low skills, poor reputations, or those new to the platform often struggle to secure jobs. Reputation on these platforms is typically reflected in rating points and client reviews. In the system designed by Merinio (described in the following subsection), the key factor is the total experience of casual personnel working in the shifts. Workers with more experience are considered professional and reliable, and the system gives them a competitive advantage over less experienced personnel to maintain overall service quality. We describe the system details in Chapter 3

2.1.2 Current Research

There has been a recent line of research in on-demand labor platforms. [Taylor, 2018] explores the operations and dynamics of platforms that connect time-sensitive customers with independent service providers, such as ride-sharing services or food delivery platforms. Taylor examines two main characteristics of these platforms: customers' sensitivity to waiting times and the independence of the agents who provide services. These factors influence optimal pricing and wage-setting strategies on the platform. In [Cachon et al., 2017], authors examine the implications of surge pricing on platforms where service providers set their own schedules, like Uber and Lyft. The authors investigate several pricing models, including surge pricing, in which both consumer prices and provider wages adapt to current demand levels. The study [Allon et al., 2023] explores how gig economy workers respond to financial and behavioral incentives, which influence their work decisions on flexible platforms like ride-hailing services. By analyzing data in collaboration with a major ride-hailing platform, the authors developed an econometric model that captures the nuances of workers' labor choices, such as whether to work and for how long. In [Benjaafar et al., 2022], the authors analyze how the expansion of the gig workforce on platforms impacts worker welfare. Their study addresses the tension between flexibility for workers and the potential downsides of increased labor supply, which can drive down wages and reduce utilization rates as more workers join the

platform. [Yan et al., 2022] provides a survey for on-demand platform labor management.

2.2 Learning Paradigms

In this section, we describe three main learning paradigms for SDPs: Reinforcement Learning (RL), Inverse Reinforcement Learning (IRL), and Behaviour Cloning (BC). IRL and BC fall under the umbrella of Imitation learning (IL).

2.2.1 Reinforcement Learning

Broadly, the RL algorithms can be split into the model-based and model-free categories.

- Model-based methods concentrate on environments where transition functions are either known or can be learned, enabling algorithms to leverage this knowledge when making decisions. Monte Carlo Tree Search (MCTS) falls within this category.
- Model-free methods, on the other hand, operate independently of the availability of environment transition functions, relying solely on the agent's accumulated experience.

Furthermore, model-free methods can be categorized into two main families of RL algorithms: policy-based and value-based methods. This distinction arises from how solutions to Markov Decision Processes (MDPs) are derived. Policy-based methods approximate the policy directly, while value-based methods focus on approximating a value function that quantifies the policy's effectiveness for a given state-action pair in the environment, also called the Q value. In addition, there exist RL algorithms that combine policy-based and value-based approaches. These hybrid methods are known as actor-critic methods [Sutton and Barto, 2018]. In actor-critic methods, the critic model approximates the value function, evaluating the quality of actions taken by the actor. The actor model, in turn, approximates the policy based on this evaluation. Both the actor and critic models utilise the policy-based and value-based RL techniques mentioned earlier. This setup enables the critic to provide feedback on the actor's actions, allowing for adjustments to the learnable parameters in subsequent training steps. The reader is referred to [Sutton and Barto, 2018] for a detailed description of these methods. In reinforcement learning methods, when artificial neural networks are used to learn approximations for Q values or approximate policies, the method is referred to as Deep Reinforcement learning or Deep Q learning. The deep neural network (DQN) helps us to learn complex value/policy functions that cannot be efficiently learnt otherwise.

Both value-based and policy-based approaches do not use the model of the environment, i.e. the transition probabilities of the model, and, hence, such approaches do not plan ahead

by unrolling the environment to the next steps. However, it is possible to define an MDP for SDPs in such a way that we can use the knowledge of the environment to improve the predictions by planning several steps ahead. Monte Carlo Tree Search (MCTS) consists of 4 steps

1. **Selection** - Starting from the root node/state, the algorithm recursively selects child nodes down to a leaf node based on a selection policy, typically using the Upper Confidence Bound for Trees (UCT) method. The UCT balances exploration (choosing nodes with less information) and exploitation (choosing nodes with higher average rewards).

2. **Expansion** - When a leaf node is reached, the algorithm checks if the node represents a terminal state (end of the game/problem). If not, one or more child nodes (expanded) are created to explore possible future actions.

3. **Simulation** - A simulation or "playout" is performed from the newly expanded node. This involves randomly sampling actions to play out the rest of the game/problem to a terminal state. The outcome of this simulation provides an estimate of the value of the node.

4. **Backpropagation** - The result of the simulation is propagated back up the tree, updating the value estimates and visit counts for all nodes along the path from the expanded node to the root.

RL has been consistently used to develop algorithms to play games [Mnih et al., 2015], [Silver et al., 2016], [Schrittwieser et al., 2020]. However, in the last decade, RL methods have received much attention. [Mazyavkina et al., 2021], [Yu et al., 2021], [Farazi et al., 2021] are some articles that examine the use of RL methods for OR problems. We describe a couple of articles to demonstrate the use of RL in different contexts.

In [Kullman et al., 2022], authors study a routing problem where an operator oversees a fleet of electric vehicles for a ride-hailing service, with the primary goal of maximizing profit. This involves effectively assigning vehicles to current ride requests and strategically managing recharging and repositioning to anticipate future demands. To address the intricacies of this challenge, we employ deep reinforcement learning. This methodology revolves around developing policies that guide decision-making processes using Q-value approximations, which are learned and refined through deep neural networks. [Khalil et al., 2017] proposes a method for learning the criteria to select the next node to visit in a heuristic framework for the TSP. They utilize a graph neural network, which can handle input graphs of any size through

message-passing mechanisms. In their approach, the network processes a graph representation of the problem augmented with features indicating visited nodes. It outputs action values for each node, which are used in reinforcement learning, specifically Q-learning, to train the network. The reward used in training is the partial tour length.

2.2.2 Imitation Learning

There are generally two approaches to imitation learning: the first is to directly learn how to imitate the expert's policy (behavior cloning), and the second is to indirectly imitate the policy instead of learning the expert's reward function (inverse reinforcement learning).

Behaviour Cloning

Behaviour cloning [Torabi et al., 2018] is an ML technique where an agent learns a task by directly imitating the behaviour demonstrated by an expert. It involves training a model, typically a neural network, to mimic the actions or decisions of a human or another agent based on a set of observed examples. The process begins with collecting a dataset of input-output pairs that represent the expert's actions in various situations. The model learns to map inputs directly to corresponding outputs without explicitly understanding the underlying principles or reasoning behind the expert's decisions. Behaviour cloning is often used in scenarios where direct imitation of expert behaviour is feasible and efficient, such as in autonomous driving, robotics, and game playing, to quickly teach agents complex tasks by leveraging human expertise. However, it can be limited by the quality and diversity of the training data and may struggle in handling situations not covered by the training examples.

In the context of optimization problems, the optimal solution represents the expert decisions. In other words, the optimization solver/algorithm acts as the expert. Hence, hypothetically, given a series of optimization problems, one can easily convert the parameters of the problem into features. Then, using the optimal solutions as targets, one can fit a supervised learning model that can make predictions about optimal solutions. Given the advancement in ML models, such an approach is promising in the field of stochastic optimization and enhancing branch-and-bound algorithms. [Bengio et al., 2021] also refers to this approach as learning from demonstrations. Research articles based on this idea are described in the following paragraph.

An application of learning from an expert is found in the context of branching policies in Branch and Bound trees of Mixed Integer Linear Programs (MILPs). Choosing which variables to branch on can have a significant impact on the size of the branch-and-bound tree and

consequently affect the time taken to solve the problem. Strong branching [Applegate, 2006] stands out as an effective method in this regard. It involves conducting a one-step lookahead for each branching decision, where multiple candidate variables are tentatively branched upon. By evaluating linear programming relaxations, it estimates potential improvements in the lower bound and selects the variable that promises the greatest enhancement. Despite not exploring all variables and relying on approximate values from linear programming, this approach remains computationally demanding. [Alvarez et al., 2017] utilize a specialized form of decision tree, a classical model in supervised learning, to approximate strong branching decisions. [Khalil et al., 2016] proposes a similar approach where a linear model is dynamically learned for each instance by initially employing strong branching and eventually replacing it with its machine learning approximation. The linear approximator of strong branching, as introduced in [Marcos Alvarez et al., 2016], is actively trained. When the estimator's reliability is in question, the algorithm reverts to true strong branching, and the outcomes are used for both branching and learning. In all these branching algorithms, inputs to the machine learning model are engineered as a fixed-length vector consisting of static features describing the instance, along with dynamic features providing information about the state of the branch-and-bound process.

In the work by [Vinyals et al., 2015], a novel neural network architecture is employed to tackle the Euclidean Travelling Salesman Problem. The authors train the model using supervised learning, with pre-computed solutions to the travelling salesman problem serving as targets. Another instance is seen in the research by [Larsen et al., 2018], where a neural network is trained to predict solutions for a stochastic load planning problem with a mixed-integer linear programming formulation. The authors utilize operational solutions, specific solutions to the deterministic version of the problem, aggregating them to provide tactical solution targets for the ML model. In an approach similar to ours, [Pham et al., 2023] adopts a prediction-based approach for online dynamic radiotherapy scheduling. They propose an Integer Programming (IP) model to derive optimal offline schedules from many instances. Subsequently, a model is trained, with the offline solution serving as expert decisions in the learning process. In all the above cases, the authors use offline deterministic solutions as an expert. [Kong et al., 2022] presents an imitation approach for stochastic optimization using energy-based models (EBMs). They first build a dataset of features and decisions, where the decision represents the optimal action under complete information of the uncertainty. A negative log-likelihood (NLL) function is used to represent the loss function, essentially minimizing the energy of the optimal actions while maximizing the energy of other points. Thus, the end-to-end stochastic programming problem is translated to learning a neural network that outputs the smallest energy for the optimal actions. To avoid overfitting, they also propose a distribution-based

regularizer which augments the energy-based objective from a global training perspective.

Inverse Reinforcement Learning

IRL [Arora and Doshi, 2021] is a branch of machine learning, specifically within the broader field of RL, that focuses on inferring the reward function given by observed behaviour. While traditional reinforcement learning aims to learn a policy that maximizes cumulative rewards given a known reward function, IRL works in the opposite direction. The objective of IRL is to deduce the underlying reward structure that an observed agent (often an expert) implicitly optimises. The motivation behind IRL arises from scenarios where directly specifying the reward function is challenging or impractical. For example, in complex tasks like driving a car, playing sports, or performing surgical procedures, it is difficult to explicitly define the reward function that captures all the nuances of expert behaviour. However, we can observe the behaviour of experts and use IRL to uncover the reward function they are optimising.

The main categories of IRL methods as described by [Arora and Doshi, 2021] are based on the core approach they use for inverse learning – margin-based optimization, entropy-based optimization, Bayesian inference, classification and regression.

- **Maximum Margin Planning** - The aim is to learn a reward function that explains the demonstrated policy better than alternative policies by a margin. The methods under this category aim to address IRL's solution ambiguity by converging on a solution that maximizes some margin.

- Entropy Optimization - IRL is essentially a poorly posed problem because each policy can be optimal for many reward functions, the IRL is ambiguous and prone to imperfect behaviour [Ziebart et al., 2008]. To resolve this ambiguity, [Ziebart et al., 2008] propose the maximum entropy inverse reinforcement learning (Maxent-IRL), which chooses the distribution that does not exhibit any additional constraints beyond matching feature expectations.

- Bayesian IRL - This approach models the reward function as a probabilistic entity and uses Bayesian inference to estimate the posterior distribution over reward functions given the expert demonstrations.

- Apprenticeship Learning - This method [Abbeel and Ng, 2004] combines IRL with learning a policy that performs similarly to the expert by iteratively refining the reward function and the corresponding policy.

In terms of application, [Ziebart et al., 2008] uses their Maxent-IRL algorithm for driver route choice modelling. They modelled the problem as an MDP and used real-life data from 25 taxi drivers. [Liu et al., 2020] develops a deep inverse reinforcement learning (IRL) algorithm to capture deliverymen's preferences from historical GPS trajectories and recommend their preferred routes. The Dijkstra algorithm is used to determine an initial optimal route based on static data, and then rewards are learnt based on it. [Alsaleh and Sayed, 2020] applies the Maxent-IRL to infer cyclist preferences during their interactions with pedestrians in non-motorized shared spaces. [Pang et al., 2020] introduces Maxent-IRL to recover human travel behaviour preferences, which can be used to reproduce people's movement and transport usage.

2.2.3 Imitation Learning Algorithms

This subsection presents an overview of different Imitation Learning Algorithms.

Supervised Learning

The first approach to addressing imitation learning is through supervised learning. In this method, we use a set of training trajectories (stationary policy) generated by an expert, where each trajectory consists of a sequence of observations and corresponding actions performed by the expert. The goal of imitation learning in this context is to train a classifier/regressor that attempts to replicate the expert's actions based on the observations at each step. This is a passive approach, where the aim is to learn a target policy by passively observing complete execution trajectories. The expert only acts prior to training the ML model, which involves training a policy based on the states encountered by the expert. Additionally, it is necessary to assume that the actions within the expert's trajectories are independent and identically distributed (i.i.d.).

The primary limitation of this supervised learning approach to imitation learning is its inability to recover from mistakes. If the model deviates from the optimal trajectory at any point, it will struggle to return to the states encountered by the expert, leading to a cascade of errors. As a result, this naive algorithm fails to generalize to unseen situations. Subsequent approaches aim to address and correct this issue.

Forward Training

The forward training algorithm, introduced by [Ross and Bagnell, 2010], trains a separate policy π_k at each time step k over a total of K steps (non-stationary policy). At each step

k, the model learns a policy π_k to mimic the expert policy π^* based on the states generated by the previously learned policies $\pi_1, \ldots, \pi_{k-1}$. This iterative training process is outlined in Algorithm 1.

Algorithm 1: Forward Training Algorithm

```
Function ForwardTrain():
    Initialize π_1^0, ..., π_K^0 to query and execute π*;
    for k ∈ {1, ..., K} do
        Sample T-step trajectories by following π^{i-1};
        Get dataset D = {(s_k, π*(s_k))} of states and expert actions at step k;
        Train classifier π_k = arg min_{π∈Π} E_{s∼D}[e_π(s)];
        Set π_j^k = π_j^{k-1} for all j ≠ i;
    end
    return π_1^K, ..., π_K^K
end
```

At iteration i, the algorithm trains the policy π_k^k on the state distribution $d_k^{\pi^{k-1}}$, while all other policies remain unchanged (i.e., $\pi_j^k = \pi_j^{k-1}\ \forall j \neq k$). After K iterations, π^T no longer queries the expert, and the process is complete. One major weakness of the presented approach is that it needs to iterate over all the T periods, where the time horizon K can be quite large or even undefined.

Search-based Structured Prediction (SEARN)

The algorithm SEARN, introduced in [Daumé et al., 2009], begins by following the expert's actions at every step. It iteratively collects demonstrations and uses them to train a new policy. New episodes are compiled by taking actions based on a mixture of all previously trained policies, as well as the expert's actions. Over time, the algorithm learns to follow its mixture of policies and gradually stops relying on the expert to determine which actions to take.

In summary, this algorithm aims to learn a classifier that navigates the search space. It operates by maintaining a current policy and attempts to use it to generate new training data for learning a new policy (new classifier). When a new classifier is learned, it is interpolated with the old classifier. However, the algorithm is complex and can be overly optimistic in practice.

Stochastic Mixing Iterative Learning (SMILe)

The SMILe algorithm, also introduced by [Ross and Bagnell, 2010], was developed to address some of the limitations of the forward training algorithm. It is a stochastic mixing algorithm based on SEARN, leveraging its advantages while providing a substantially simpler implementation and requiring less demanding interaction with an expert. The algorithm trains a stochastic stationary policy over several iterations and employs a "geometric" stochastic mixing of the trained policies.

Specifically, we start with a policy π_0 that follows exactly the actions of the expert. At each iteration i, we train a policy π_i to mimic the expert under the trajectories induced by the previous policy π_{i-1}. We then incorporate the newly trained policy into the existing mix of policies using a geometric discount factor $\alpha(1-\alpha)^{i-1}$. Thus, the new policy π_i is a mix of i policies, with the probability of using the expert's action being $(1-\alpha)^i$. We can terminate after any iteration N by removing the probability of querying the expert and re-normalizing to obtain the unsupervised policy

$$\tilde{\pi}_N = \frac{1}{1-(1-\alpha)^N}\left[\pi_N - (1-\alpha)^N \pi_0\right]$$

that never queries the expert. The SMILe algorithm is described in Algorithm 2.

Algorithm 2: SMILe

Function `SMILe()`:
- Initialize $\pi_0 \leftarrow \pi^*$ to query and execute the expert;
- **for** $i = 1 \dots itr$ **do**
  - Execute π_{i-1} to get $D = \{(s, \pi^*(s))\}$;
  - Train classifier $\hat{\pi}^{*i} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim D}[e_\pi(s)]$;
  - $\pi^i = (1-\alpha)^i \pi^* + \alpha \sum_{j=1}^{i} (1-\alpha)^{j-1} \hat{\pi}^{*i}$;
- **end**
- Remove expert queries - $\pi_e^N = \frac{\pi^N - (1-\alpha)^N \pi^*}{1-(1-\alpha)^N}$;
- **return** π_e^N

end

Dataset Aggregation

In many cases, expert demonstrations will not be uniformly sampled across the entire state space and therefore, it is likely that the learned policy will perform poorly when the trajectory followed is different from those followed by the expert. This is because when a learned policy

is used in practice, it produces its own distribution of states that will be visited, which will likely not be the same as in the expert demonstrations. This distributional mismatch leads to compounding errors, which is a major challenge in imitation learning. [Ross et al., 2011] proposed the DAgger algorithm in 2010 to address these issues. DAgger is an iterative policy training algorithm that reduces the problem to online learning. At each iteration, the main classifier is re-trained in all states that the learner has ever encountered. The primary advantage of DAgger is that the expert teaches the learner how to recover from past mistakes. [Ross et al., 2011] shows that this algorithm has no regret guarantees in an online setting. A no-regret guarantee means that the learner will converge to a policy that minimizes errors against the expert, even when feedback is incrementally provided.

Given an initial training set D_0 generated by the expert policy π^*, an initial novice policy π_0 is trained. Using this initialization, DAgger iteratively collects additional training examples from a mixture of expert and novice policies. During a given episode, the combined expert-and-model system interacts with the environment under the supervision of a decision rule. The decision rule determines, at each step in time, whether to use the learned model or the expert's choice of action to interact with the environment. In the algorithm, we use observation of the system rather than the actual state. The state represents the complete, underlying status of the environment at a particular time. Observations are what the agent perceives or senses at any given moment. These observations might be partial or noisy representations of the true state. The observations (o_k) received during the episodes of an epoch, together with the corresponding actions of the expert, form a new dataset denoted as D_i. This new dataset of training examples is combined with the previous sets $D = D \cup D_i$ and the model is re-trained on D. Not here that we use the observation o_k, rather than the state. An observation is defined as partial or possibly noisy information that the agent receives about the current state. The general form of the DAgger Algorithm is presented in Algorithm 3.

By allowing the novice to act, the combined system explores parts of the state space further away from the nominal trajectories of the expert. By querying the expert in these parts of the state space, the novice can learn a more robust policy. We will now describe different decision rules in the literature.

1. **Naive DAgger** - The Naive DAgger algorithm always returns the expert's action for any given observation. A drawback of this approach is that if the learned model begins to make errors, there is no mechanism to correct them during the current episode. As a result, this can lead to the model encountering states that are either risky or irrelevant to the system, causing the initial iterations of the DAgger algorithm to be quite noisy,

Algorithm 3: DAgger : Dataset Aggregation

Function `DAgger(`$DR(.)$`):`
  Initialize π_0 to any policy in Π;
  for $i = 1 \dots itr$ **do**
    Sample T-step trajectories with $a_k = DR(o_k)$;
    Get $D_i = \{o_k, \pi^*(o_k)\}$ of states visited;
    Aggregate datasets: $D \leftarrow D \cup D_i$;
    Train π_{i+1} on D;
  end
  return best π_i on validation
end

with frequent oscillations.

Algorithm 4: NAIVE DAgger Decision Rule

Function `NaiveDAgger(`o_k`):`
  $a_k \leftarrow \pi(o_k)$;
  return a_k
end

2. **VANILLA Dagger** - In the VANILLA DAgger decision rule (Algorithm 2), the expert's action is selected with probability $\beta_i \in [0, 1]$, where i represents the DAgger epoch. If $\beta_i = \lambda\beta_{i-1}$ for some $\lambda \in (0, 1)$, the novice progressively takes more actions in each subsequent epoch. As the novice is exposed to more training examples from earlier epochs, it gains increasing autonomy to explore the state space.

   Initially, the expert has greater control over the trajectories, which helps in avoiding undesirable or unsafe states. However, the vanilla DAgger decision rule does not incorporate any similarity metric between the actions proposed by the novice and the expert. As a result, even if the novice suggests a potentially unsafe action, vanilla DAgger permits the novice to proceed with probability $(1 - \beta_i)$.

3. **SAFE DAgger** - The decision rule used by SAFE DAgger, outlined in Algorithm 6 and referred to as SafeDAgger*, permits the novice to take action if the distance between the actions is below a specified threshold τ [Zhang and Cho, 2016]. An optimal decision rule would enable the novice to act only when there is a sufficiently low probability of the system transitioning to an unsafe state. When the combined system is close to an unsafe state, the acceptable deviation from the expert's action should be smaller than when the system is further away from such states. Consequently, the single threshold τ

Algorithm 5: VANILLA DAgger Decision Rule

Function `VanillaDAgger`$(o_k, i, \beta_0, \lambda)$**:**
  $a_k \leftarrow \pi_i(o_k)$;
  $a_k^* \leftarrow \pi^*(o_k)$;
  $\beta_i \leftarrow \lambda^i \beta_0$;
  $z \sim \text{Uniform}(0, 1)$;
  if $z \leq \beta_i$ **then**
    return a_k^*;
  else
    return a_k;
  end
end

employed in SafeDAgger* can be either overly conservative when the system is distant from unsafe states or too lenient when it is close to them.

To minimize the frequency of expert queries, SAFE DAgger (Algorithm 8) approximates the SAFE DAgger* decision rule using a deep policy that assesses the likelihood of the novice policy deviating from the reference policy. Blue fonts are used to highlight the differences from the VANILLA DAgger. This algorithm introduces an additional policy, π_{safe}, referred to as the safety policy. This policy takes a partial observation of a state, $\phi(s)$, along with a primary policy π, and outputs a binary label that indicates whether the primary policy π is likely to deviate from a reference policy π^* without making a query.

The deviation of a primary policy π from a reference policy π^* is defined as

$$\epsilon(\pi, \pi^*, o_k) = \|\pi(o_k) - \pi^*(o_k)\|_2^2 \tag{2.1}$$

With this deviation defined, the optimal safety policy π_{safe}^* is expressed as

$$\pi_{\text{safe}}^*(\pi, o_k) = \begin{cases} 0, & \text{if } \epsilon(\pi, \pi^*, o_k) > \tau \\ 1, & \text{otherwise,} \end{cases} \tag{2.2}$$

where τ is a predefined threshold. At each moment, the safety policy assesses whether it is safe for the primary policy to execute its action. If it is deemed safe (i.e., $\pi_{\text{safe}}(\pi, o_k) =$

Algorithm 6: SAFE DAgger* Decision Rule

Function `SafeDAgger*`(o_k, τ)**:**
  $a_k \leftarrow \pi_i(o_k)$;
  $a_k^* \leftarrow \pi^*(o_k)$;
  if $\|a_k - a_k^*\| \leq \tau$ **then**
    return a_k;
  else
    return a_k^*;
  end
end

1), we proceed with the action suggested by the primary policy (i.e., $\pi(o_k)$). Conversely, if it is not safe (i.e., $\pi_{\text{safe}}(\pi, o_k) = 0$), we opt for the action dictated by the reference policy instead (i.e., $\pi^*(o_k)$. Algorithm 8 describes SAFE DAgger. First, the safe strategy is used instead of the naive approach to gather training examples (line 6 in Algorithm 8). Second, the subset selection process (line 7) significantly reduces the number of queries made to the reference policy. Only a limited subset of states, where the safety policy returns 0, requires labeling with reference actions. This contrasts with the original DAgger, where every collected state had to be verified against the reference policy. After updating the primary policy with D_i, which is the union of the initial training set D_0 and all the examples collected thus far, safety policy is updated. This step ensures that the safety policy accurately identifies the states that pose challenges or risks for the most recent primary policy.

Algorithm 8: SAFE DAgger

Function `SafeDAgger()`**:**
  Collect D_0 using a reference policy π^*;
  Collect D_{safe} using a reference policy π^*;
  $\pi_0 = \arg\min_\pi l_{\text{supervised}}(\pi, \pi^*, D_0)$;
  $\pi_{\text{safe},0} = \arg\min_{\pi_{\text{safe}}} l_{\text{safe}}(\pi_{\text{safe}}, \pi_0, \pi^*, D_{\text{safe}} \cup D_0)$;
  for $i = 1 \ldots N$ **do**
    Collect D' using the safety strategy with π_{i-1} and $\pi_{\text{safe},i-1}$;
    Subset Selection: $D' \leftarrow \{o_k \in D' \mid \pi_{\text{safe},i-1}(\pi_{i-1}, o_k) = 0\}$;
    $D_i \leftarrow D_{i-1} \cup D'$;
    $\pi_i = \arg\min_\pi l_{\text{supervised}}(\pi, \pi^*, D_i)$;
    $\pi_{\text{safe},i} = \arg\min_{\pi_{\text{safe}}} l_{\text{safe}}(\pi_{\text{safe}}, \pi_i, \pi^*, D_{\text{safe}} \cup D_i)$;
  end
  return best π_i and $\pi_{\text{safe},i}$;
end

4. **ENSEMBLE DAgger** - In ENSEMBLE DAgger [Menda et al., 2019], the decision rule is based on two key measures: discrepancy τ and doubt χ. The first measure, discrepancy τ, corresponds to the SAFE DAgger decision rule and is defined as a metric for the deviation of the novice from the expert (as outlined in line 4 of Algorithm 3). The second measure, doubt χ, is a variance measure that indicates the confidence of the novice or the familiarity of the current state with the training data.

Algorithm 9: ENSEMBLE DAgger* Decision Rule

1 **Function** `EnsembleDAgger*`(o_k, τ, i, χ)**:**
2 **end**
3 $a_k, \sigma^2_{a_k} \leftarrow \pi(o_k)$;
4 $a^*_k \leftarrow \pi^*(o_k)$;
5 $\hat{\tau} \leftarrow \|a_k - a^*_k\|^2$;
6 $\hat{\chi} \leftarrow \sigma^2_{a_k}$;
7 **if** $\hat{\tau} \leq \tau$ ***and*** $\hat{\chi} \leq \chi$ **then**
8 | **return** a_k;
9 **else**
10 | **return** a^*_k;
11 **end**

Ensemble DAgger leverages an ensemble method [Zhou et al., 2002], a technique that trains multiple neural networks to perform the same task and then combines their outputs into a single prediction. In this approach, uncertainty is quantified by calculating the variance, σ^2, of the predictions from the ensemble members, while the novice's action is represented as the average of these predictions. Ensemble DAgger enables the learned model to act only when its behavior is sufficiently close to that of the expert, measured by a discrepancy $\hat{\tau}$, and when it is confident in its decision, measured by a confidence metric $\hat{\chi}$. Unlike Vanilla DAgger, Ensemble DAgger only samples training data when a specific decision rule is met, with this rule being based on different metrics compared to those in Vanilla DAgger.

In addition, there are several other complex versions of the DAgger algorithm. These include HG-DAgger [Kelly et al., 2019], Lazy DAgger [Hoque et al., 2021], Dropout DAgger [Menda et al., 2017].

AggreVaTe

Previous approaches primarily focus on ensuring alignment with a demonstrator, often neglecting long-term costs, which can lead to inadequate management of unavoidable errors.

AggreVaTe, introduced in 2014 [Ross and Bagnell, 2014], enhances the DAgger algorithm by concentrating on selecting actions that minimize the expert's cost-to-go (total cost), rather than solely aiming to reduce the zero-one classification loss that comes from imitating the expert's actions. AggreVaTe can be seen as a transition from imitation learning to no-regret online learning, emphasizing regret reduction by minimizing the cost-to-go instead of merely addressing immediate classification losses.

In the initial iteration, we passively collect data by observing the expert performing the task. During each trajectory, at a randomly selected time k, we explore an action a in state x and record the expert's cost-to-go Q after executing this action. The expected future cost-to-go to take action a in state x, followed by executing the policy π for $k-1$ steps, is denoted as $Q_k^\pi(x, a)$.

Like the DAgger algorithm, AggreVaTe (Algorithm 10) gathers data through the interaction of the learner in the following manner:

- In each iteration, the current learner policy π_i is used to execute the task. The execution is paused at a randomly chosen time k, where an action a is explored in the current state x. The control is then handed over to the expert, who continues until the time horizon K.
- This procedure produces new examples of the expert's cost-to-go, represented as (x_k, a_k, Q), based on the distribution of states encountered by the current policy π_i.
- Finally, the data sets are aggregated and π_{i+1} is trained on the combined data sets.

It is crucial to recognize that methods dependent on cost-to-go estimates can be impractical, as obtaining an estimate for a single state-action pair may require executing an entire trajectory. In many scenarios, minimizing imitation loss using DAgger is more feasible, as it allows us to observe the expert's chosen action at each visited state along a trajectory, enabling the collection of K data points per trajectory rather than just one.

2.3 PFA

A policy function approximation (PFA) is an analytical function that maps a state to an action. These functions can be broadly categorised into lookup tables, parametric functions, and nonparametric functions. PFAs are the simplest and easiest class of policies to compute, although they typically require human input to specify the architecture. Given the variety of decisions encountered in everyday life, most decisions are made using simple rules that

Algorithm 10: AggreVaTe

Function `AggreVaTe(`β`):`
  Initialize $\hat{\pi}_1 \leftarrow$ any policy in Π, $D \leftarrow \emptyset$;
  for $i = 1 \dots N$ **do**
    $\pi_i \leftarrow \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$;
    for $j = 1 \dots m$ **do**
      Sample uniformly $k \in \{1, 2, \dots, K\}$;
      Start a new trajectory in some initial state drawn from the initial state distribution;
      Execute current policy π_i up to time $k - 1$;
      Execute some exploration action a_k in current state x_k at time k;
      Execute expert from time $k + 1$ to K and observe estimate of cost-to-go $\hat{Q}$ starting at time k;
    end
    Collect dataset $D_i = \{(x_k, a_k, \hat{Q})\}$;
    $D \leftarrow D \cup D_i$;
    Train $\hat{\pi}_{i+1}$ on D;
  end
  return best π_i on validation set;
end

can be described as PFAs, making them arguably the most widely used type of policy. No optimization problem needs to be solved once the policy is trained. A PFA can be parameterized by a vector θ. An example is a basic inventory policy. Whenever the inventory drops below the value $\theta^{\min}$, we order up to some upper value $\theta^{\max}$. Considering more complex functions, we define the decision rule approach as employing a parameterized mapping $\pi_\theta(x)$, e.g., linear policies [Ban and Rudin, 2019] or a neural network [Oroojlooyjadid et al., 2020].

The problem of finding a good PFA is termed a policy search. This usually involves solving the following problem.

$$\min_{\theta=(f\in\mathcal{F},\theta\in\Theta^f)} \mathbb{E}\Big\{C(x_k, \pi(x_k|\theta))\Big\} \tag{2.3}$$

$f \in \mathcal{F}$ represents the search over a class of functions, including lookup tables, parametric and non-parametric functions, and any associated parameters θ in the parameter space defined by Θ^f.

Next, we list down the different types of policy functions.

1. **Lookup Tables** - A lookup table policy is a function in which, for a particular discrete state x_k, we return a discrete action $a_k = \pi(x_k)$. This means that we have one

parameter (an action) for each state. Lookup tables are very easy to understand and are commonly used in real life.

- Inventory Management - Lookup tables are employed to store optimal reorder quantities, safety stock levels, and lead times based on historical demand patterns and supplier performance metrics. They facilitate quick decision-making regarding inventory replenishment and minimize stockouts.
- Call Routing - Call-in centres use specific rules to govern how a call should be routed. A classic example is routing customers with a different first language to an advisor who speaks that language or directing an at-risk customer to a retention specialist.
- Customer Segmentation - An e-commerce platform uses lookup tables to categorize customers into segments (e.g., loyal, occasional buyers) based on purchase frequency and monetary value. This data guides targeted promotions and customer retention efforts.
- Games - Engines like Stockfish or AlphaZero use transposition tables to store previously analyzed board positions and their best moves, enabling rapid decision-making and improving overall performance in competitive chess matches.
- Clinical Decision Systems (CDS) - Lookup tables are integral to CDSS, where they store medical guidelines, diagnostic criteria, treatment protocols, and evidence-based recommendations. This assists healthcare professionals in making accurate diagnoses and treatment decisions. In a hospital setting, a CDS uses lookup tables to match patient symptoms and lab results with known disease patterns and treatment protocols, providing clinicians with actionable insights and reducing diagnostic errors.

But in practice, lookup tables can be very hard to optimize since there is a value (the action) for each state. In business contexts, lookup table policies are commonly referred to as business rules, although often parameterized. These rules are typically not optimised using formal methods; this chapter will outline how to achieve optimization.

2. **Parametric PFA** - The simplest form of parametric approximations is linear decision rules (LDRs). Linear decision rules emerged in the early development of stochastic optimization [Garstka and Wets, 1974] and then in robust optimization [Ben-Tal and Nemirovski, 1999], [Bertsimas and Thiele, 2006] in inventory management. A linear

decision rule policy might be of the form

$$\pi(x_k|\theta) = \theta_0 + \theta_1\phi_1(x_k) + \theta_2\phi_2(x_k)$$

Here ϕ_1, ϕ_2 represent feature functions that capture essential state information. For example, [Ban and Rudin, 2019] uses LDRs to study two variants of the newsvendor problem.

Although linear decision rules (LDRs) are easier to fit, they can be limited in scope. Kernel Hilbert Space (RKHS)-based decision rules are more flexible than linear ones, as highlighted [Bertsimas and Koduri, 2022].

Recently, there has been significant interest in non-linear policies, particularly those utilizing deep neural networks (DNN) [Goodfellow et al., 2016]. A neural network is a computational model inspired by the way biological neural networks in the human brain process information. These models are designed to recognize patterns, classify data, and make decisions based on the input they receive. The main advantage of neural networks is their ability to approximate virtually any functional form, eliminating the need to specify the form beforehand. Neural networks have been used for decades in deterministic engineering control problems, where decisions often involve a three-dimensional force on a device. Despite their high-dimensional architecture, neural networks are prone to overfitting, making them struggle with noisy data, which is common when simulating policies. Additionally, neural networks require very large training datasets, often necessitating millions of policy simulations. These methods have been rigorously studied in reinforcement learning [Arulkumaran et al., 2017], particularly within a class of algorithms known as policy gradient methods. They find applications in robotics and strategy games such as Go, chess, and various video games.

The use of NN also allows us to generate a stochastic policy. For example, in a vehicle routing problem, one can generate a probability distribution on the next node to visit. This type of policy is referred to as the Boltzmann policy [Powell, 2022]. [Bello et al., 2016] develop a neural combinatorial optimization framework that uses RL to optimize a policy modelled by a DNN. Using several classical combinatorial optimization problems, such as the Travelling Salesman Problem (TSP) and the knapsack problem, they show the effectiveness and generality of their architecture. [Kool et al., 2018], [Nazari et al., 2018] are some examples of DNN used to obtain a PFA in vehicle routing problems. The DNN directly generates a probability distribution over possible actions. Other applications of Neural Networks for learning PFA include the job-shop scheduling problem [Zhang et al., 2020], and the resource management problem [Mao et al.,

2016].

3. **Non-Parametric Policies** - Often, the policy that needs to be learned is too complex to be accurately represented by low-order polynomials. Conversely, higher-order polynomials may encounter challenges such as overfitting. In such scenarios, one approach is to partition the state space into distinct regions and learn a policy separately for each region. An illustration of this can be found in [Gombolay et al., 2018], where the authors address a job-shop scheduling problem. They begin by predicting task priorities based on the current state and select the most crucial task accordingly. Subsequently, they employ another model to determine whether to execute this prioritized task.

2.4 Integration of ML and CO

Current research areas in the intersection of Combinatorial Optimization (CO) and ML can be categorized into two main directions: ML-augmented CO and End-to-End CO learning. The former focuses on using ML to aid the decisions performed within an optimization algorithm used to solve CO problems. In the context of combinatorial optimization, these are broadly categorized into methods that learn to guide the search decisions in branch and bound solvers and methods that guide the application of primal heuristics within branch and bound. Some examples are in [Khalil et al., 2016], [Gasse et al., 2019], [Tang et al., 2020]. These papers are generally based on one of the learning paradigms of Section 2.2. The focus of this section is the combination of ML and CO techniques to form integrated models which predict solutions to optimization problems under uncertainty, also referred to as End-to-End Learning (E2EL).

Sequential Learning and Optimization

Traditionally, optimization problems were solved more in a sequential manner, where ML models were trained to predict a conditional distribution for the uncertain parameters given the covariates, and then an associated optimization problem was solved to obtain optimal actions. This represents a sequential approach to tackle the stochastic optimization problem in two stages. The key idea is to train a model to generate accurate parameter predictions $\hat{\boldsymbol{\xi}}$ with respect to ground-truth values $\boldsymbol{\xi}$. The predictive model is built to maximize its predictive power, and the predictions are used as parameters of a preceding decision model. In the second stage, decisions are made based on the model's predictions, and the costs associated with decisions are realized. ML models are trained and evaluated by common loss functions such as MSE. The main drawback is that the model does not take into account

how our current predictions affect the downstream optimization problem. Alternatively, approaches in this category are also referred to as *predict-then-optimize* [Elmachtoub and Grigas, 2022] or *prescriptive optimization* [Bertsimas and Kallus, 2020] or *prediction-focused learning* [Mandi et al., 2024].

An example of this two-step process can be found [Ferreira et al., 2016]. They tackle the challenge of pricing and predicting demand for products that an online retailer has never sold before, which account for the majority of sales and revenue. They use machine learning techniques to predict future demand for new products and then solve the subsequent multi-product price optimization problem to determine accurate prices. Motivated by the delivery operations, [Liu et al., 2021] studies a last-mile delivery problem and discusses a framework that integrates travel-time predictors with order-assignment optimization for drivers. [Bertsimas and Kallus, 2020] assigns weights to the observations of the uncertain parameters in the historical data. In their approach, they estimate the objective of an instance by applying the same weights generated by the ML model to the corresponding objective functions of those samples. [Mukhopadhyay and Vorobeychik, 2017] applies this process for EMS systems by first predicting emergency incidents and, subsequently, solving a CO problem to determine the optimal ambulance waiting locations and ambulance allocation. Additional applications can be observed in areas such as Retail Inventory Management [Ferreira et al., 2016], Healthcare Resource allocation [Chan et al., 2012], and Logistics [Chu et al., 2023].

End-to-End Learning

E2EL represents an approach where an ML model (g_θ) is trained to optimize a loss function that gauges the quality of the resultant decisions. Numerous recent studies have used this approach, also known as the integrated /smart *predict-and-optimize*, integrated learning and optimization, (task-based) end-to-end learning/forecasting/optimization, decision-aware learning, contextual optimization, etc. Generating the predicted parameters $\hat{\boldsymbol{\xi}}$ serves as an intermediary step in the integrated approach, with the primary focus of training not on the accuracy of $\hat{\boldsymbol{\xi}}$. This method can influence the estimation process towards a solution with a higher Mean Squared Error (MSE) or any distance metric, yet still results in a nearly optimal decision. A special loss function is used to focus the impact of prediction on downstream costs and is referred to as a task loss. [Kotary et al., 2021], [Mandi et al., 2024], [Sadana et al., 2024] provide extensive surveys of this approach.

[Sadana et al., 2024] divides E2EL as follows:

- Conditional distribution-based model: This aims to learn a conditional distribution

model that will directly capture a task-based objective within the context of stochastic programming. [Donti et al., 2017] appears to be the first to study the integration of ML and CO. They model the distribution of the uncertain parameters using parametric distributions (exponential and normal). The results obtained outperformed sequential learning and optimization.

- Regret minimization task: A recent line of work defines regret as the difference between the full-information optimal objective value and the objective value realized by the prescriptive decision. Given the current state x, a predictive model g_θ and the corresponding decision a, we define the regret (loss function $\mathcal{L}$) using [Sadana et al., 2024] as

$$Regret(\mathbf{a}, \boldsymbol{\xi}) = C(\mathbf{x}, \mathbf{a}, \boldsymbol{\xi}) - C(\mathbf{x}, \mathbf{a^o}, \boldsymbol{\xi}) \tag{2.4}$$

  The above measures the suboptimality of decision a given the realization of ξ against the optimal offline decision a^o, which acts as the target. Training these end-to-end models involves the introduction of external CO solvers into the training loop of an ML model, often a DNN. As pointed out in [Kotary et al., 2021], combinatorial problems with discrete state spaces do not offer useful gradients; viewed as a function, the argmin of a discrete problem is piecewise constant. The challenge of forming useful approximations to $\frac{\delta \mathcal{L}}{\delta \xi}$ is central in this context and must be addressed to perform backpropagation.

  [Baty et al., 2024] and [Greif et al., 2024] apply a regret minimization approach to dynamic vehicle routing and dynamic inventory routing problems, respectively. Similarly, [Rautenstrauß and Schiffer, 2025] develops models for Emergency Medical Service (EMS) systems aimed at minimizing response times to satisfy legal requirements and ensure timely care for patients.

  In these pipelines, a machine learning (ML) model is trained to parameterize a combinatorial optimization (CO) problem. The solution to the parameterized CO problem yields a feasible solution to the original problem. The objective is to train the ML model to produce parameterizations that minimize the deviation between the predicted and optimal solutions, essentially framing the task in an imitation learning setting.

  Both [Greif et al., 2024] and [Rautenstrauß and Schiffer, 2025] employ the DAgger algorithm to construct the dataset used to imitate an offline policy. To measure non-optimality, they use a perturbed Fenchel-Young loss, which quantifies the discrepancy between the predicted and optimal solutions. This loss function facilitates end-to-end training via stochastic gradient descent.

  [Liu et al., 2020] uses a DNN to model the routing behavior of users in a transporta-

tion network and learn the parameters by minimizing the mismatch between the flow prescribed by the variational inequality and the observed flow.

- Optimal action imitation: In this approach, one learns a model g_θ that produces prescriptive action a that is as close as possible to the optimal offline action and, therefore, closer to the regret objective. The loss function for such a problem can be defined as

$$Loss(\mathbf{a}, \boldsymbol{\xi}) = \ell(\mathbf{a}, \mathbf{a^o}) \tag{2.5}$$

  where $\ell(\mathbf{a}, \mathbf{a^o})$ is distance function. This distance function can be as simple as Mean Squared Error (MSE) or Binary Cross-Entropy (BCE). When we employ regret as the distance metric, the approach aligns with regret minimization techniques. In this case, the model g_θ learns to imitate the optimal hindsight action, aiming to minimize the difference (or "regret") between the chosen action and the expert's optimal action in hindsight. We refer the reader to [Hussein et al., 2017] for a comprehensive review of imitation learning methods and applications.

  In this setup, we refer to the target action as the *expert* action, representing an ideal decision or choice based on available information. By minimizing the distance to the expert's action, the model effectively learns from the expert's behavior, making this a typical imitation learning problem, as discussed in Section 2.2.

CHAPTER 3 NOTIFICATION TIMING PROBLEM - DESCRIPTION AND COMPLEXITY

In this Chapter, we first describe the on-demand system in detail in Section 3.1. Then, we delve into a simplified offline version of the Notification Timing Problem (NTP) and demonstrate that it is $\mathcal{NP} - complete$. The offline version represents where we have perfect information on employee response delays. Section 3.2 introduces some essential notation and assumptions for the offline problem. Following that, we will present the decision version of the problem and establish some preliminary concepts. Section 3.5.2, defines a reduction from the Subset-Sum problem, a well-known $\mathcal{NP} - complete$ problem. To provide a better understanding of why this reduction works, Section 3.5.3 will offer some intuitive insights. Finally, in 3.5.4, we outline the structure of the optimal schedule for the deterministic version of our problem NTP obtained through the reduction and provide a proof demonstrating that this deterministic version of the problem is also $\mathcal{NP} - complete$.

3.1 Problem Description

The client company initially assesses the number of on-demand shifts required for a specific upcoming day based on a precise forecast of future demand. This required number of shifts is then input into an electronic system overseen by the platform. The system generates a list of eligible employees according to their skills and availability, and an operations manager initiates the communication process on the client's behalf. The electronic system notifies workers about available shifts, monitors their responses, and updates the shift schedule as employees accept the assignments.

Notifications are sent to employees on their phones in the order of seniority, which is determined by experience and reputation. The number of notifications sent at any given time follows a preset operating policy. Employees have the option to take their time before responding to available shifts, as they might be busy, need to consider their options or wish to consult with family. If employees do not respond at all, it indicates their unavailability for that day. The interval between the system's notification and the employee's response is referred to as the "response delay". Once an employee accepts a shift, that shift is marked as occupied. As more employees respond and claim shifts, overall shift occupancy increases. Employees generally have different preferences that influence their choices.

The system also allows senior employees to select shifts that junior employees have already

claimed, provided they respond within a specified time frame known as the "cutoff". This results in a schedule change where the originally scheduled employee is "bumped". Senior employees have this privilege because they are the most experienced and productive. The cutoff period begins for an employee from the moment a notification is sent to that employee. If an employee responds after the cutoff has passed, they can only choose from unoccupied shifts, thereby losing their seniority advantage. The system encourages prompt shift selection by rewarding employees who consistently respond quickly with a higher seniority level, giving them access to a wider range of shifts. Employees who are bumped are promptly notified and can choose from the remaining available shifts at that time. This also means that they can bump some junior employees who hold their next preferred shift, and thus, one response can initiate multiple bumps. The scheduling of shifts is finalized either when all employees have responded or when the planning period ends.

The uncertainty in the system arises from the fact that the employee response delay is not known in advance. If too many employees are contacted early in the scheduling horizon, this can lead to multiple bumps, especially if some senior employees take longer to make their decisions. Frequent disruptions or bumps can cause frustration among junior employees and may even lead them to leave the system, which is particularly problematic given the transient nature of the workforce. Employee retention is, therefore, crucial. Conversely, if too few employees are contacted in an effort to avoid bumps, there may not be enough time for responses within the planning horizon, potentially resulting in unassigned shifts.

This creates a challenge for the system in determining how many notifications are to be sent at any given time, considering the current state of the system. From a management perspective, it is important to establish a notification policy that minimizes bumps while ensuring all shifts are filled. However, these two goals often conflict, adding complexity to the problem. For example, to minimize bumps, an ideal policy might be not to send any notifications, but this would leave shifts unfilled. Conversely, if the priority is to ensure all shifts are covered, notifying everyone at the outset might seem optimal, but it could lead to excessive bumps.

We refer to this problem of determining the optimal timing for contacting employees as the Notification Timing Problem (NTP). The decision involves finding a notifying policy that balances these conflicting objectives. Specifically, at any given time, the policy should assess the current state of the system and determine when to contact the next employee. The NTP is, therefore, an online sequential decision problem with uncertainty, where each decision incurs a cost in terms of either bumps or unassigned shifts.

3.2 Offline NTP

We define a simplified offline/static version of the NTP with the following assumptions and simplifications. This problem is slightly different from the actual problem faced online, but it demonstrates that even under simple assumptions, it is not easy to solve. The offline problem means all response delays and shift preferences for each employee are known from the outset of the planning process. All employee response times are assumed to be shorter than the planning horizon. While, in reality, response times might exceed the horizon, this offline version can exclude such employees from consideration as they will not be able to choose any shifts even if they are included. The response delays are discretized to the nearest integer. Table 3.1 defines the notations used for the problem.

Table 3.1 Model, Parameters, and Variables of the NTP

Sets			
$\mathcal{E}$			set of employees in the order of seniority with 1 being most senior
$\mathcal{L}$			set of shifts
$\mathcal{X}_i = \{l_i^1, l_i^2, \ldots, l_i^M\}$			ordered set of employee preferences, $l_i^1 \in \mathcal{L}$
Parameters			
M			number of employees
L			number of shifts
$\mathbf{r} = [r_i]_{i \in \mathcal{E}}$			response delays for all employees
H			planning horizon
Variables			
s_i	$i \in \mathcal{E}$	$s_i \geq 0$	time of start of communication for employee i
e_i	$i \in \mathcal{E}$	$e_i \geq 0$	time of response for employee i
y_{ij}	$i, j \in \mathcal{E}$	$y_{ij} \in \{0, 1\}$	1 if i bumps j else 0
$b_i = \sum_{j \in \mathcal{E}: i<j} y_{ij}$	$i \in \mathcal{E}$	$b_i \geq 0$	number bumps caused by employee i

The planning horizon is discretized into units, and notifications are sent at these discrete time steps only. The planning horizon H is much greater than the number of employees available after discretization. We assume that there is no cutoff time. Senior employees can respond and bump junior employees at any point in the planning process. We consider that there are the same number of shifts ($L = M$) to fill as employees, and all employees are eligible or can work all these shifts. In actual online instances, we generally have a larger number of employees than there are shifts. Some of these employees may not want to work and may not respond. On the other hand, the total number of employees who want to work can be more than the total available shifts. Since all this information is known, one can easily filter employees in such cases to ensure $L = M$. The number of shifts vacant for the online instances will depend on how many employees can respond within the horizon. Given

that we know the response delay in advance for the offline instances, we can easily compute the number of shifts that will remain vacant beforehand. Thus, this cost is constant, and the objective is just to minimize total bumps, given the knowledge of the vacancies in the offline formulation.

An employee is referred by his seniority number $i \in \mathcal{E} = \{1, 2, \ldots, M\}$. For any two employees $i, j \in \mathcal{E}$, $i < j$ means employee i is more senior than employee j, and therefore employee i has to be compulsorily notified before or simultaneously as employee j. Let $L_i \in \mathcal{X}_i$ refer to the next available shift for employee i at the current time t. This means that at $t = 0$, $L_i = l_i^1$, where l_i^1 is the most preferred shift in $\mathcal{X}_i = \{l_i^1, l_i^2, \ldots, l_i^M\}$. Also, let I_l refer to the employee that occupies shift $l \in \mathcal{L}$ at time t. Similarly, at $t = 0$, $I_l = \phi$, indicating that the shift is unoccupied at the beginning. When employee i responds, then:

1. he will always select the first available and preferred shift L_i. If this shift L_i is currently unoccupied ($I_{L_i} = \phi$) then that shift is directly assigned to employee i, that is $I_{L_i} = i$. That particular shift is then not available for selection for any employee j such that $i < j$. L_i is then updated to the next available shift in the preference order for i. This shift L_i may be occupied by a junior employee or may be unoccupied.

2. if L_i is currently occupied by employee j_1 such that $i < j_1$, that is $I_{L_i} = j_1$, then a bump occurs. This results in a change of personnel, and we update $I_{L_i} = i$ and $I_{L_{j_1}} = j_1$. We repeat the steps 1 and 2 for j_1. This may result in another bump if L_{j_1} is allocated to someone else since j_1 needs to be assigned his preferred shift.

3. At any point in the simulation, if there is no shift available for some junior employee j, then $L^i = \phi$, referring to a null shift. This will happen only when senior employees to j have selected all available shifts.

Hence, by design of the system, a response by an employee i may result in a chain of bumps. Let $\mathcal{B}_i$ refer to a chain of employees in the order of seniority who are bumped as a result of i responding at time t. We define the total bumps by employee i at time t as the total schedule changes due to a response, which in this case is $|\mathcal{B}_i| = b_i$. We can construct the bump chain $\mathcal{B}_i$ as follows, given i has responded at time t. A pseudo-code Algorithm 3 to compute the chain is also given in the appendix.

$$\mathcal{B}_i = \{j_1, \ldots, j_k : I_{L_i} = j_1, I_{L_{j_k}} = \phi, I_{L_{j_o}} = j_{o+1}, j_o < j_{o+1}, 1 \leq o < k\} \tag{3.1}$$

In this case, j_1 is directly bumped by i, while others are indirectly bumped in a chain due to the need to reassign j_1, then j_2 and so on. The shift L_{j_k} for the last employee j_k in the set was either unoccupied or refers to a null shift wherein the j_k can no longer work. Due to discretization, it may be possible that we will have multiple responses at the same time. In such a case, we calculate $\mathcal{B}_i$ for each employee who responded in the order of seniority. This ensures that there is no bump between those responding in the same period. This implies that if two senior employees respond at the same time, a junior employee could be counted as being bumped twice, even though there is only one schedule change for that employee at that moment. This reflects real-time operations, where the system is updated continuously. The scenario that two senior employees respond at the exact same time is impossible due to the real-time updates. The response of one of the senior employees will be recorded earlier than the other and hence the junior employee will end up getting bumped twice by the senior ones.

3.3 Employee Preferences

Next, we talk about employee preferences, which are difficult to model. Our industrial partner does not store any data on employees. This data is difficult to collect as asking employees to rank their preferences every time is not feasible. Even if this data is available, preferences are expected to change day by day for all employees. Further, learning a model over such a dataset would further induce errors. Hence, in the absence of the preference data, our aim in solving the problem is to obtain a schedule to send notifications to minimize all potential instances of bumps. A potential instance of a bump is when a senior employee responds after a junior employee. It is equivalent to saying we want to minimize the total count where a senior employee responds after a junior employee. We then show that the minimum **potential** instances of bumps are a tight upper bound on the total bumps given any set of employee preferences. Given an employee i has responded at time e_i, the number of potential bumps p_i by employee i equals $|\mathcal{P}_i|$ where

$$\mathcal{P}_i = \{j : j \in \mathcal{E}, e_i > e_j, i < j\} \tag{3.2}$$

Let $S = [(s_i, e_i)]_{i \in \mathcal{E}}$ denote a notification schedule. S is a feasible schedule for the instance if all the employees are notified in the order of seniority ($s_i \leq s_{i+1}$) and all of them respond within the horizon ($e_i \leq H$). Also, let $B_S = \sum_{i \in \mathcal{E}} b_i$ denote the total bumps suffered by the employees when following the notification schedule S, where each employee i induces

$b_i = |\mathcal{B}_i| = \sum_{j \in \mathcal{E}} y_{ij}$ bumps. The variable b_i will be used later in the paper. Nevertheless, we define it here. Then the following theorem holds.

Theorem 1. *Given an instance $I = \{H, M, \mathbf{r}\}$, let $S^*_{\mathcal{X}}$ denote an optimal schedule with preferences $\mathcal{X}$ such that $\mathcal{X} = \{\mathcal{X}_i\}_{i \in \mathcal{E}}$. Also, let S^*_p be the schedule that minimizes the total potential bumps for this instance, and S^*_I be the schedule that minimizes the total bumps with identical preferences. Then, $B_{S^*_{\mathcal{X}}} \leq B_{S^*_p} = B_{S^*_I}$.*

Corollary 1. *Minimizing the total potential bumps is equivalent to minimizing total bumps under the same employee preferences.*

The proof of Theorem 1 is provided in the Appendix. Consequently, the NTP with the same preferences and equal employees and shifts represents the worst-case scenario regarding bumps. Diverse employee preferences only serve to decrease the occurrence of bumps. Henceforth, NTP will specifically denote the problem where one minimizes the total potential bumps. All variables related to bumps, b_i, y_{ij}, B_S will now refer to potential bumps. Any reference to a bump from this point on will correspond to a potential bump unless mentioned otherwise. In the further sections, we show that this problem is hard, and hence the problem of minimizing bumps under some preferences is also hard.

3.4 MIP Model - Static/Offline NTP

The MIP formulation for the simplified NTP is given by equations (3.3) - (3.7). This formulation only decides when to notify employees. It does not explicitly allow shifts to employees. To have a feasible schedule, we simply need to ensure that all employees respond within the time horizon H. Note that in this formulation, we merely minimize the total number of bumps in the final schedule. The vacancy cost is not included as it is easy to know precisely how many shifts one can schedule using the offline information indicated earlier.

$$\boldsymbol{MIP_{NTP}} := \quad \min \quad \sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{E}: i < j} y_{ij} \tag{3.3}$$

$$\begin{aligned}
\textbf{s.t.} \quad & s_i \leq s_j && \forall i, j \in \mathcal{E}, i < j && (3.4)\\
& e_i \leq H && \forall i \in \mathcal{E} && (3.5)\\
& e_i = s_i + r_i && \forall i \in \mathcal{E} && (3.6)\\
& e_i - e_j \leq (r_i - r_j) y_{ij} && \forall i, j \in \mathcal{E}, i < j, r_i > r_j && (3.7)\\
& y_{ij} \in \{0, 1\} && \forall i, j \in \mathcal{E}, i < j && (3.8)\\
& s_i, e_i \geq 0 && \forall i \in \mathcal{E} && (3.9)
\end{aligned}$$

The objective function minimizes total bumps. (3.4) ensures the feasibility of the schedule so that employees are notified according to seniority. (3.5) makes sure that all employees respond before the horizon. The start and end times should match the response delay through (3.6). Finally, (3.7) counts each potential bump instance in the schedule. A potential bump occurs when any employee i responds after any junior employee j that is $e_i > e_j$. This equation represents the $Big-M$ constraint. $(r_i - r_j)$ is a valid value of $Big-M$ because first $i < j$ and the seniority constraint must be respected between i and j, that is, $s_i \leq s_j$. Then, the bound of $(r_i - r_j)$ is only met when $s_i = s_j$, when i, j are called at once. In addition, we only include the constraint when $i < j$ and $r_i > r_j$. When $r_i \leq r_j$, i cannot bump j due to the seniority constraint as i would respond earlier than j. This constraint also counts bumps as they would be in real-time when there are multiple responses simultaneously. Note that in this formulation, we do not explicitly track potential bumps. To count the potential total bumps an employee suffers, one just needs to count how many senior employees have responded after that employee. This also implies that employees who get bumped are allocated the next shift in the identical preference order.

3.5 Complexity

This section shows that the static/offline NTP is $\mathcal{NP}-complete$ by reducing from the Subset Sum problem. We first define some preliminaries required for the proof. Then we give the reduction explaining the intuition and giving an example.

3.5.1 Preliminaries

Let NTP(I) denote the NTP defined by instance $I = \{H, M, \mathbf{r}\}$. Also, let $\mathcal{F}_{NTP}(I)$ denote the set of all feasible solutions for NTP(I). Given any schedule S, the *makespan*, denoted by $C(S)$, is defined as the time required to schedule all shifts.

$$C(S) = \max_{i \in \mathcal{E}} \{e_i\}$$

The makespan is feasible if $C(S) \leq H$. Consider any two employees $i, j \in \mathcal{E}$ where $i < j$. A No Bump Schedule (NBS) is any schedule S such that $B(S) = 0$, i.e., there cannot be two employees i, j, and $i < j$ such that $e_i > e_j$. Note that this schedule will not have potential bumps or realized bumps. Any NBS, if feasible for a given instance, is also optimal. Let C_0^* denote the minimum makespan of an NBS such that $B(S) = 0$.

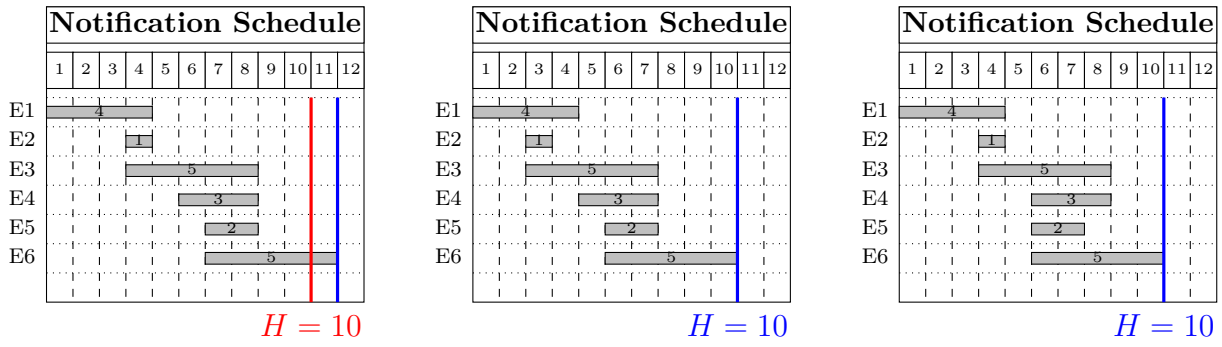
The following proposition gives an expression to calculate C_0^* for the NTP.

Proposition 1. *The minimum makespan of a NBS is* $C_0^* = r_1 + \sum_{i=1}^{M-1} (r_{i+1} - r_i)^+$.

The following example illustrates the problem.

Example: Consider an instance I with $M = 6$ employees, a planning horizon $H = 10$, and response times $\mathbf{r} = [4, 1, 5, 3, 2, 5]$. Figure 3.1(a) presents an NBS for this instance, with a makespan of 11, indicated by the blue line. For clarity, we refer to each employee as E_i. An NBS is feasible for NTP(I) if the condition $H \geq C_0^*$ is satisfied. However, since $H = 10$, it is evident that an NBS is not feasible in this case. Let $\mathcal{O}_{NTP}(I) \subset \mathcal{F}_{NTP}(I)$ represent the set of optimal schedules that minimize potential bumps for instance I. Multiple optimal solutions can exist for the same instance: Figure 3.1(b) and (c) show two such schedules, $S_1, S_2 \in \mathcal{O}_{NTP}(I)$, respectively. Introducing potential bumps into the schedule can reduce the total time needed to complete all shifts within the planning horizon. This is demonstrated in Figure 3.1(a) and (b). In Figure 3.1(a), employee E_2 is notified at time 3, while in Figure 3.1(b), E_2 is notified one unit earlier, at time 2. Notifying E_2 earlier introduces a bump for that employee, but it allows the system to notify all subsequent employees one unit earlier, speeding up the process. Figure 3.1(c) illustrates a similar situation for employee E_5.

This example highlights why introducing bumps is sometimes necessary to complete a schedule within the given horizon.



(a) NBS: $\mathbf{C_0^* = 11}$ (b) $\mathbf{S_1^*}$: $\mathbf{B(S_1^*) = 1}$, $\mathbf{C(S_1^*) = 10}$ (c) $\mathbf{S_2^*}$: $\mathbf{B(S_2^*) = 1}$, $\mathbf{C(S_2^*) = 10}$

Figure 3.1 Comparison of notification schedules under different conditions: (a) NBS, (b) S_1^*, and (c) S_2^*.

3.5.2 Reduction

The decision version of the NTP is as follows:

INSTANCE : Finite vector $\mathbf{r}$ with duration $r_i \in Z^+$, for each $i \in \mathcal{E} = \{1, .., M\}$, a target horizon H and a target number of potential bumps B^* .
QUESTION : Does a feasible schedule S exist with at most B^* potential bumps for a given instance?

To prove that the NTP is $\mathcal{NP} - complete$, we reduce from the Subset-Sum problem.

The SUBSET-SUM problem is defined as follows using [Karp, 1972], [Gary and Johnson, 1979].

INSTANCE : Finite set A, size $a_j \in Z^+$, for each $j \in A$ and a target positive integer W.
QUESTION : Is there a subset $A^* \subseteq A$ such that the sum of sizes in A^* is exactly W i.e. $\sum_{j \in A^*} a_j = W$?

Given $A = \{1, \ldots, N\}$, the sizes $[a_j]_{j \in A}$, and a target W, let $NTP\Big([a_j]_{j \in \mathcal{A}}, W\Big)$ denote the reduced instance. Let the total number of employees be $M = N + \sum_{j \in A} a_j + 1$. The letter i is used to denote the seniority level of an employee. We define different sets of employees and their seniority as described in Table 3.2. These three sets represent the critical set, the stable set and the specific last employee. A critical employee is the one who will only cause bumps. A stable set employee is one that will only suffer a bump. There is one critical employee i_k for each index $k \in A$. Also, any two critical employees i_k, i_{k+1} are separated by a block of stable set employees $\mathcal{E}_k^S$ of size a_k. There are a total of $|A|$ blocks of stable employees. Finally, to complete we have a singleton set for the most junior employee.

Table 3.2 Employee sets for the reduction

Sets		
$\mathcal{E}$	set of all employees in order of seniority, $= \{1, ...M\}$	$\lvert\mathcal{E}\rvert = M$
$\mathcal{E}^C$	set of seniority indices of critical employees, $= \{i_k : i_k = k + \sum_{j=1}^{k-1} a_j, k \in A\}$	$\lvert\mathcal{E}^C\rvert = N$
$\mathcal{E}_k^S$	k^{th} block of stable employees, $= \{i : i_k < i \leq i_k + a_k\}$	$\lvert\mathcal{E}_k^S\rvert = a_k$
$\mathcal{E}^S$	set of seniority indices of stable employees, $= \cup_{k \in A} \mathcal{E}_k^S$	$\lvert\mathcal{E}^S\rvert = \sum_{k \in A} a_k$
$\mathcal{E}^M$	singleton set for the last employee, $= \{M\}$	$\lvert\mathcal{E}^M\rvert = 1$

The response delay for each set of employees is given by Equation 3.10. For any employee i_k in the critical set, the response delay is simply the sum of all sizes till index k. For the stable employees in the k^{th} block, the response delay is the same as the critical employee i_{k-1}. For

the last employee, it is simply the sum of all sizes.

$$r_i = \begin{cases} \sum_{j=1}^{k} a_j & \text{if } i = i_k \in \mathcal{E}^C, k \in A, \\ r_{i_{k-1}} & \text{if } i \in \mathcal{E}^S, i_k < i < i_{k+1}, k \in A, \\ \sum_{j \in A} a_j & \text{if } i \in \mathcal{E}^M; \end{cases} \tag{3.10}$$

where $r_{i_0} = 0$.

The total planning horizon is set to $H = C_0^* - W$ where $C_0^* = 2\sum_{j \in A} a_j$ using Proposition 1. It is assumed that $W \leq \sum_{j \in A} a_j$, else it is polynomial to check that the subset sum is infeasible when $W > \sum_{j \in A} a_j$. One can easily verify that the above instance has size $r_i \in \mathbb{N}$, for each $i \in \mathcal{E}$ and a positive target integer $H \in \mathbb{N}$ and is achieved in polynomial operations. The following example illustrates the reduction.

Example: Consider a Subset-Sum instance with $A = \{1, 2, 3\}$, sizes $a_1 = 1, a_2 = 4, a_3 = 7$ and with $W = 5$. As a consequence $|\mathcal{E}^C| = 3, |\mathcal{E}^S| = 12, |\mathcal{E}^M| = 1, M = 16$. The seniority indexes in the critical, stable, and last employee sets are defined below.

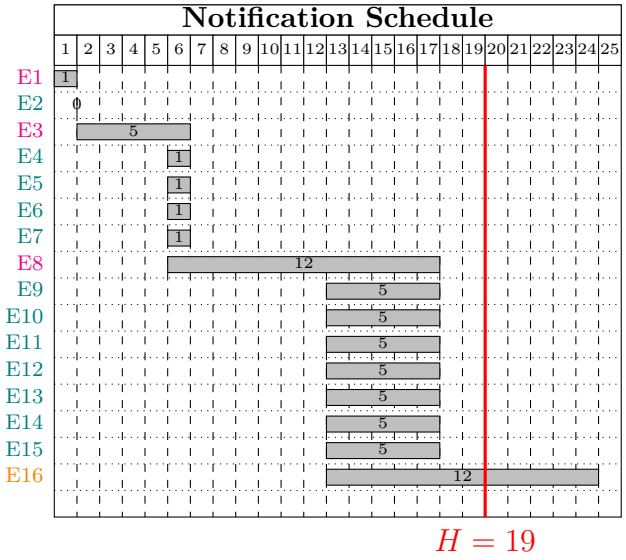
- $\mathcal{E}^C = \{1, 3, 8\}$
- $\mathcal{E}^S = \{2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15\}$
- $\mathcal{E}^M = \{16\}$

The response delay vector $R = [1, 0, 5, 1, 1, 1, 1, 12, 5, 5, 5, 5, 5, 5, 5, 12]$. The horizon $H = C_0^* - W = 19$, where $C_0^* = 24$. Color codes are used to denote employees from their respective set on the x-axis using the following convention: $\mathcal{E}^C$, $\mathcal{E}^S$, $\mathcal{E}^M$. The NBS and the optimal schedule S^* are in Figure 3.2 (a) and (b).

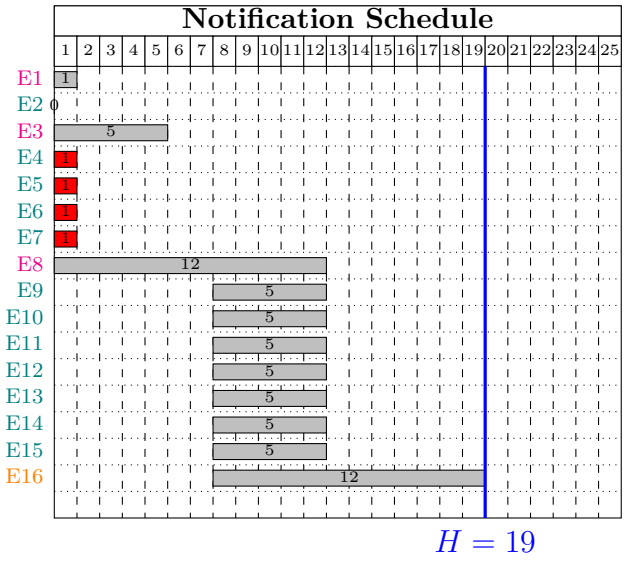
Let us define a special type of Notification schedule that will help us establish the proof.

Definition 1. *Block Schedule $S(A')$: Given $A' \subseteq A$, a block schedule $S(A') = [(s_i, e_i)]_{i \in \mathcal{E}}$ for $NTP\Big([a_j]_{j \in \mathcal{A}}, W\Big)$ is constructed as follows:*

$$s_i = \begin{cases} 0 & \textit{if } i = 1, \\ s_{i_k} + r_{i_k} - r_i & i = i_k + 1, \forall k \in A \backslash A', \\ s_{i-1} & \textit{else ;} \end{cases} \tag{3.11}$$



(a) NBS with $\mathbf{C_0^*} = 24$.



(b) $\mathbf{S}^*$: $\mathbf{B}(\mathbf{S}^*) = \mathbf{5}$ and $\mathbf{C}(\mathbf{S}^*) = 19$.

Figure 3.2 Comparison of notification schedules: (a) NBS schedule and (b) Optimal schedule.

Alternately it means that all stable set employees corresponding to $k \in A'$ are potentially bumped by the respective critical employee. One can easily verify that $\forall k \in A, r_{i_k} - r_i = r_{i_k} - r_{i_{k-1}} = a_k$ when $i = i_k + 1$ in Equation 3.11. Figure 3.2 (b) is an example of the block schedule $S(A')$ with $A' = \{1, 2\}$.

3.5.3 Intuition

The crux of the reduction lies in the way the response delays are defined. They ensure the following.

- There are no bumps between any two stable set employees because for any $i, i' \in \mathcal{E}^S$ such that $i < i'$ we have $r_i \leq r_{i'}$.
- There are no bumps between any two critical set employees, as we have $r_{i_k} \leq r_{i_{k'}}$ for any $i_k, i_{k'} \in \mathcal{E}^C$ such that $i_k < i_{k'}$.
- Any critical employee i_k can either potentially bump only a whole block of stable set employees $\mathcal{E}_k^S$ to create time savings or bump none. Alternately this means that the critical employee i_k can introduce a total of a_k potential bumps in the schedule. As a consequence, the total potential bumps $B(S)$ will be a sum of these sizes. The aim is now to introduce bumps so that $B(S)$ equals to W. For example, in Figure 3.2 (a), one can see that E3 corresponding to index 2, can only bump E4, E5, E6, and E7 corresponding to the size $a_2 = 4$. E3 cannot bump any employee junior to E7. It is also ensured that no stable employee bumps any other employee from the entire set.
- A block schedule is always feasible and optimal. The block schedule $S(A)$, which notifies everyone at the start, is always feasible. Furthermore, when the total potential bumps $B(S) = \sum_{k \in A'} b_{i_k}$, where $A' \subset A$, the block schedule $S(A')$ gives the minimum makespan and hence is also optimal.
- Any schedule S such that $B(S) < W$ is infeasible. If a critical employee i_k, potentially bumps a block of stable set employee $\mathcal{E}_k^S$, then it also creates maximum time savings of a_k. Note that a block schedule will create time savings of exactly a_k units for a potential bump by critical employee i_k. Thus, starting from the NBS schedule that has a makespan of C_0^*, we set $H = C_0^* - W$ to make certain that any feasible solution has at least W potential bumps.

The goal is to find a feasible block schedule S, such that $B(S) = W$. When this happens, all the indexes $\{k_1, k_2, \ldots, k_Z\}$ such that critical employee I_{k_u}, causes all the potential bumps, $\forall u \in \{1, \ldots, Z\}$ are part of the subset, and we would have a YES answer to Subset-Sum problem.

3.5.4 Proof

Our main result in this section establishes the computational complexity of the NTP. The proof of all propositions, corollaries, lemmas, and theorems is given in the appendix. Let $\mathcal{F}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$ and $\mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$ denote the feasible and optimal set of solutions for $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$.

Proposition 2. *The block schedule $S(A')$ has the following properties:*

1. $\forall k \in A, s_{i_k} = s_{i_k-1}$.
2. $\forall i \in \mathcal{E}_k^S, \forall k \in A$, $e_i = e_{i_k}$ *or* $s_i = s_{i_k}$.
3. $s_{i_{k+1}} = s_{i_k} + \mathbf{1}_{k\notin A'} a_k$, *where* $\mathbf{1}$ *denotes indicator variable.*
4. $B(S(A')) = \sum\limits_{j\in A'} a_j$
5. $C(S(A')) = C_0^* - \sum\limits_{j\in A'} a_j$.

Properties 1 and 2 mean that the block schedule respects the seniority constraint of the problem. Given the start of any critical employee i_k, property 3 allows us to find the start time of the next critical employee i_{k+1}. Property 4 gives an equation to compute the total potential bumps suffered by the block schedule. Property 5 establishes the total makespan of the block schedule.

Proposition 3. *For any $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, $C = e_M$.*

Proposition 4. $\forall k \in A$, $\forall l \in A$ *employee i_k cannot bump employee i_l.*

Proposition 5. $\forall i \in \mathcal{E}^{\mathcal{S}}$, $\forall i' \in \mathcal{E}$ *employee i cannot bump employee i'.*

Proposition 6. $\forall k \in A$, $\forall i \in \mathcal{E} : i \geq i_{k+1}$, *employee i_k cannot bump employee i.*

Corollary 2. $\forall k \in A$, $\forall i \in \mathcal{E}$ *if employee i_k bumps employee i, then $i \in \mathcal{E}_k^S$.*

Proposition 3 defines the makespan of the schedule. Since the last employee i_M has the maximum response delay, it is simple to show that the end of its interaction e_M, marks the end of the schedule for the $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$. Propositions 4, 5, and 6 establish that it is only the critical set of employees who cause the bumps. Moreover, any critical employee i_k can only bump the immediately following a_k employees in the order of seniority, all of whom belong to the stable set from Corollary 2.

Lemma 1. $\forall S^* \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, $\forall k \in A, b_{i_k} = a_k$ *or* 0.

Corollary 3. $\forall S \in \mathcal{F}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, *the total bumps* $B(S) = \sum_{k\in A} b_{i_k}$.

Lemma 2. $S(A) \in \mathcal{F}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$.

Lemma 1 proves that any critical employee i_k will bump the immediately following a_k junior employees or bump no one. As a consequence of Lemma 1, Corollary 3 establishes the total bumps in the schedule. Next, we show that the block schedule $S(A)$ is always feasible for the $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$ in Lemma 2. Hence, the reduced problem is always feasible.

Lemma 3. $\exists A' \subseteq A$ *such that* $S(A') \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$.

Corollary 4. $\forall S^* \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, *the optimal objective value* $B(S^*) \geq W$.

The final piece of the proof comes from Lemma 3. It means that given any horizon W, there exists some subset A' and a corresponding block schedule $S(A')$ that is optimal. This is a very important result as it allows only concentrating on a solution space that comprises block schedules.

Theorem 2. *For* $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, $B(S^*) = W \Leftrightarrow \exists A^* \subseteq A$ *such that* $\sum_{j\in A^*} a_j = W$.

Theorem 3. $NTP(I) \in$ NP.

Theorem 4. $NTP(I)$ *is* NP-complete.

Theorem 2 proves that for any $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, the number of bumps in the optimal schedule $B(S^*)$ is always equal to the target W defined by the Subset-Sum if and only if the Subset-Sum problem is feasible. Additionally, it is easy to show that $NTP(I)$ is in the class NP in Theorem 3. Hence, the offline problem is hard from Theorem 4. The hardness stems from the fact that one needs to count how many bumps an employee can cause.

3.6 Comparing offline and online solutions

Now consider again the online version of the problem where the response delays are unknown while all other assumptions of the NTP are valid. Assume we have an online algorithm for the problem that observes the system at each discrete time epoch in the horizon and decides when to notify the next employee. One would be naturally interested in evaluating such an algorithm. In particular, it is common to use competitive ratios [Borodin and El-Yaniv, 2005] that measure how good an online algorithm is compared with the optimal result of the offline

model where all the information is provided. The competitive ratio of an online algorithm for an optimization problem is simply the approximation ratio achieved by the algorithm, that is, the worst-case ratio between the cost of the solution found by the algorithm and the cost of an optimal solution. The following theorem holds.

Theorem 5. *Given any online algorithm, there exists an instance I in which the algorithm suffers the maximum potential bumps or has vacant shifts. The offline counterpart will have zero potential bumps and zero vacancies.*

As a consequence of Theorem 5, the performance of an online algorithm can be arbitrarily poor. This is a significant result as it suggests the problem remains hard even online.

3.7 Conclusion

Traditional on-demand personnel scheduling systems have many issues, mainly relating to employee flexibility and ease of operation for the management. These systems have hardly received any attention in the operations research community to optimize their operations. This paper presents a novel, flexible, dynamic, semi-automatic on-demand personnel scheduling system to tackle these issues. One aspect of the flexibility in the system allows senior employees to replace/bump junior employees already scheduled to work a shift. These bumps are undesirable, and the management would like an operating policy to reduce them as much as possible while ensuring all shifts are filled. In real cases, employees have preferences depending on their personal schedules and their likes. Since employee preferences are difficult to capture, we consider first showing that having the same preferences represents the worst-case problem and is equivalent to a problem where we want to minimize all potential possibilities of bumps. A potential possibility represents all cases where any senior employee responds after a junior employee. Our main contribution is to show that this problem of minimizing potential bumps while ensuring all shifts are scheduled is $\mathcal{NP} - complete$ even if all uncertainty is known beforehand in the offline problem. We do this by reducing the SUBSET-SUM problem. Finally, we also show that any algorithm for the online version can be arbitrarily bad when compared to the offline problem.

CHAPTER 4 DYNAMIC NOTIFICATION TIMING PROBLEM

In the real world, employee response delays are stochastic and can vary widely among employees. However, we assume that the distribution of the response delays is available. The uncertainty creates an additional layer of difficulty, as there is a chance that employees may not respond at all. This can happen if some employee has a response delay greater than the horizon. In such cases, waiting for employees to respond to avoid bumps may lead to vacant shifts at the end of the planning horizon. We describe the dynamic stochastic version of the NTP called as DNTP in Section 4.1 and also give a two-stage stochastic formulation for the offline problem. In Section 4.1.2, we design approximate policy functions for DNTP.

4.1 Problem Description - DNTP

All data parameters of the deterministic problem (NTP) remain applicable to the stochastic version DNTP, with some additional considerations. In the stochastic version, we introduce a shorter cutoff time for bumps, denoted by D. This cutoff time is assumed to be known to all employees, and they are expected to respond within this timeframe. Specifically, if an employee is busy when they receive a notification, they will still react within the cutoff period to ensure they can continue to exercise their ability to bump junior employees.

Moreover, the stochastic model assumes that the number of available employees exceeds the number of shifts to be filled, creating a buffer in case some employees fail to respond in time. In this context, employee response delays, denoted as r_i, are modeled as independent and identically distributed (i.i.d.) random variables. These delays introduce an element of uncertainty, as the exact time at which each employee will respond is not deterministic. Similar to the deterministic case, all employees remain eligible for all shifts, but there is an operational limit on the maximum number of notifications that can be sent simultaneously, which is constrained by resource limitations such as communication capacity and workforce management protocols.

In the offline version of the problem (where all parameters are known in advance), it is relatively straightforward to determine the number of unfilled shifts. Specifically, any employee whose response time is shorter than the planning horizon can potentially fill a shift. Vacant shifts occur only when fewer employees respond within the horizon than the number of available shifts. In this case, the notification schedule can be optimized to minimize bumps while ensuring the shift-vacancy constraint is satisfied.

However, in the stochastic case, the optimization becomes more complex. Since the number of vacant shifts depends on the varying response times across different scenarios, a unified notification schedule must be developed that works effectively across all possible scenarios. Unlike the deterministic scenario, the number of vacant shifts cannot be predetermined, as the delays in employee responses will vary according to the probability distributions modeled in the stochastic framework. This necessitates a more robust approach to scheduling that accounts for the uncertainty inherent in the system.

We propose a two-stage stochastic model: in the first stage, we determine the notification schedule for each employee; in the second stage, bumps are counted based on the realization of employee response. Let Ω denote the set of outcomes for all the response delays. We use ω to denote a scenario and the corresponding response delay as $\mathbf{r}_\omega$. The MIP formulation for this problem is given by MIP_{DNTP-S} in equations (4.1) - (4.14). Table 4.1 defines the additional variables and parameters.

Table 4.1 Parameters and Variables of the DNTP

Parameters	
M	number of employees
L	number of shifts
D	cutoff time to bump a junior employee
G	reward for filling a shift
W	maximum number of concurrent notifications per time unit
Variables	
$y_{ij\omega} \in \{0,1\}$	1 if employee $i \in \mathcal{E}$ bumps employee $j \in \mathcal{E}$ in scenario ω else 0
$\bar{z}_{i\omega} \in \{0,1\}$	1 if employee $i \in \mathcal{E}$ responds within the horizon and gets a shift in scenario ω else 0
$\hat{z}_{i\omega} \in \{0,1\}$	1 if employee $i \in \mathcal{E}$ responds within the horizon and does not get a shift in scenario ω else 0
$\theta_\omega \in Z^+$	number of shifts vacant in scenario ω

$$\boldsymbol{MIP_{DNTP-S}} := \quad \min \quad \frac{1}{|\Omega|} \sum_{\omega \in \Omega} Q(s, \mathbf{r}_\omega) \tag{4.1}$$

$$\text{s.t.} \qquad s_i \leq s_j \qquad \forall i, j \in \mathcal{E}, i < j \tag{4.2}$$

$$s_i + 1 \leq s_{i+W} \qquad \forall i, i + W \in \mathcal{E} \tag{4.3}$$

$$s_i \in Z^+ \qquad \forall i \in \mathcal{E} \tag{4.4}$$

$$\tag{4.5}$$

$$\text{where} \quad Q(s, \mathbf{r}_\omega) := \quad \min \quad \left\{ G\theta_\omega + \sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{E}: i<j} y_{ij\omega} \right\} \tag{4.6}$$

$$\text{s.t.} \quad s_i + r_{i\omega} \geq (H+1)(1 - \bar{z}_{i\omega} - \hat{z}_{i\omega}) \qquad \forall i \in \mathcal{E} \qquad (4.7)$$

$$s_i + r_{i\omega} \leq H + r_{i\omega}(1 - \bar{z}_{i\omega} - \hat{z}_{i\omega}) \qquad \forall i \in \mathcal{E} \qquad (4.8)$$

$$s_i - s_j + \delta_{ij\omega} \leq \delta_{ij\omega} y_{ij\omega} + (H + r_{i\omega})(1 - \bar{z}_{i\omega} + \hat{z}_{i\omega}) \qquad \forall i, j \in \mathcal{E}, i < j,$$

$$r_{j\omega} \leq r_{i\omega} \leq D \qquad (4.9)$$

$$\sum_{i \in \mathcal{E}} \bar{z}_{i\omega} + \theta_\omega \geq L \qquad (4.10)$$

$$\sum_{j<i} \hat{z}_{j\omega} \leq i(1 - \bar{z}_{i\omega}) \qquad \forall i \in \mathcal{E} \qquad (4.11)$$

$$y_{ij\omega} \in \{0, 1\} \qquad \forall i, j \in \mathcal{E}, i < j \qquad (4.12)$$

$$\bar{z}_{i\omega}, \hat{z}_{i\omega} \in \{0, 1\} \qquad \forall i \in \mathcal{E} \qquad (4.13)$$

$$\theta_\omega \geq 0 \qquad (4.14)$$

We use a weighted objective function that aims to minimize both bumps and shift vacancies. That is, instead of enforcing a strict shift-vacancy constraint as in MIP_{NTP}, we introduce a high penalty, G, for any unfilled shifts. In the offline version of this problem (NTP), it is straightforward to determine the number of unfilled shifts. Any employee with a response time shorter than the planning horizon can fill a shift, meaning vacant shifts occur only when fewer employees respond within the horizon than available shifts. Thus, the notification schedule can be optimized to minimize bumps while satisfying the shift-vacancy constraint. However, in the stochastic case, since we need a unified notification schedule that works across all possible scenarios, the number of vacant shifts cannot be predetermined due to varying response times across scenarios. Given the first-stage decisions s_i, the number of employees responding back will also vary in each scenario. In some scenarios, all employees may need to be notified to fill the shifts, while in others, all shifts may be scheduled before notifying every employee. Thus, each scenario will have a different number of vacancies and bumps.

To correctly count the bumps, we introduce two new variables, $\bar{z}_{i\omega}$ and $\hat{z}_{i\omega}$, to track whether employee i has responded and whether they are assigned a shift, respectively for each realized scenario of $\mathbf{r}_\omega$. Constraints (4.2) ensure that seniority is respected, while constraints (4.3) enforce the operational limit of notifying a maximum of W employees per epoch. The second-stage constraints are provided in equations (4.7) - (4.13). Constraints (4.7) and (4.8) determine when employee i responds. Constraints (4.9) track potential bumps, and constraints (4.10) count vacant shifts in each scenario. Additionally, constraints (4.11) ensures that if employee i responds and is assigned a shift (i.e., $\bar{z}_{i\omega} = 1$), then all senior employees who responded earlier will also be assigned a shift (i.e., $\hat{z}_{i\omega} = 0$). Constraints (4.10), (4.11)

together ensure that $\bar{z}_{i\omega}$, $\hat{z}_{i\omega}$ are assigned appropriate values if an employee gets a shift or not. We introduce a large constant G to ensure complete shift assignment. Note that G is used only to prioritize shift assignments and does not reflect the actual cost of leaving a shift vacant. Estimating the true cost of a vacant shift is challenging, as management may resort to alternative means to fill the shift as a last resort.

This formulation is complex and computationally expensive to solve. We also provide a complete information offline formulation for a single scenario ω, denoted as MIP_{NTP2}, $(4.15) - (4.13)$. This formulation is a simplified version of MIP_{DNTP-S} requiring only one of the variables $\bar{z}_{i\omega}$, $\hat{z}_{i\omega}$ to track responses.

Variables	
z_i	binary variable equal to 1 if employee $i \in \mathcal{E}$ responds within the horizon else 0
θ	number of shifts vacant

Table 4.2 Parameters, and Variables of the DNTP2

$$MIP_{NTP2} := \quad \textbf{min} \quad G\theta + \sum_{i\in\mathcal{E}} \sum_{j\in\mathcal{E}:i<j} y_{ij} \tag{4.15}$$

Constraints

$$s_i \leq s_j \qquad \forall i, j \in \mathcal{E}, i < j \tag{4.16}$$

$$s_i + r_i \geq (H+1)(1 - z_i) \qquad \forall i \in \mathcal{E} \tag{4.17}$$

$$s_i + r_i \leq H + r_i(1 - z_i) \qquad \forall i \in \mathcal{E} \tag{4.18}$$

$$s_i - s_j + \delta_{ij} \leq \delta_{ij} y_{ij} + (H + r_i)(1 - z_i) \qquad \forall i, j \in \mathcal{E}, i < j, r_j \leq r_i \leq D \tag{4.19}$$

$$\sum_{i\in\mathcal{E}} z_i + \theta \geq L \tag{4.20}$$

$$s_i + 1 \leq s_{i+W} \qquad \forall i, i + W \in \mathcal{E} \tag{4.21}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{E}, i < j \tag{4.22}$$

$$s_i \in \mathbb{R}^+ \qquad \forall i, \in \mathcal{E} \tag{4.23}$$

$$z_i \in \{0, 1\} \qquad \forall i \in \mathcal{E} \tag{4.24}$$

The above formulation is similar to MIP_{DNTP-S}. However, since we are only concerned with one scenario at a time, we need only one single additional variable z to track when an employee has responded. The following constraints are added to the formulation.

$$s_{i+1} - s_i \leq H(1 - z_i) \qquad \forall i \in \mathcal{E} - \{M\}, r_{i+1} \geq r_i \tag{4.25}$$

$$y_{ij} \leq z_i \qquad \forall i, j \in \mathcal{E}, i < j \tag{4.26}$$

$$y_{ij} \leq z_j \qquad \forall i, j \in \mathcal{E}, i < j \tag{4.27}$$

(4.25) specifically refers to case when $r_{i+1} \geq r_i$. Since the next employee, $i+1$, has a greater response time, there is no incentive to wait to notify $i+1$ unless that employee is to be pushed out of the horizon. Hence (4.25) will ensure that this employee is either notified at the same time or responds after the end of the horizon. (4.26) and (4.27) ensures that employee i can only bump employee j if both of them respond within the time horizon.

4.1.1 Sequential Decision Model

We now formally present the model for DNTP by introducing the key elements of sequential decision-making.

Decision Epochs: Let $\mathcal{K} = \{0, 1, \ldots, H\}$ be the set that contains all decision moments within the planning horizon H for some list of shifts denoted by L. A decision epoch occurs at fixed time intervals of 1 unit.

State Variables: In any period, $x_k = (x_k^n, x_k^s, x_k^r)$ gives the state of the system. Here x_k^n is the number of notifications sent until epoch k, and x_k^s is a list of indicators for each employee with a value of 1 if that employee has responded and 0 otherwise. $x_k^r = [x_k^{ri}]_{i \in \mathcal{E}}$ keeps track of the bump cutoff and is used to know if an employee can bump others or not.

Decision Variable: For every decision moment k in the planning horizon, the policy decides the number of eligible employees $a_k \geq 0$ to send notifications next, given the state x_k.

Exogenous Information: In DNTP, at every epoch, the responses from employees represent the exogenous information. Let $\xi_k = \{\bar{x}_k^s\}$, which is an array of indicators for each employee with a value of 1 if an employee has responded in that epoch.

Transition Function: The transition function, $X^M(.)$ describes the transition from state x_k to x_{k+1} such that $x_{k+1} = X^M(x_k, a_k, \xi_k)$. More specifically,

- $x_{k+1}^n = x_k^n + a_k$.
- $x_{k+1}^s = x_k^s + \bar{x}_k^s$.
- $x_{k+1}^r = [x_{k+1}^{ri}]_{i \in \mathcal{E}}$ such that

$$x_{k+1}^{ri} = \begin{cases} 0, & \text{if } \bar{x}_k^{si} = 1, \\ D-1, & \text{if } x_k^n < i < x_k^n + a_k, \\ \max(x_k^{ri} - 1, 0), & \text{else.} \end{cases}$$

Reward Function: We have two objectives that we can optimize for the problem. Let $C_k(x_k, a_k)$ be the cost in each epoch and is given by $C_k(x_k, a_k) = \sum_{i \in \mathcal{E}: \bar{x}_k^{si}=1} b_k^i$, where b_k^i is the bumps caused by employee i. Also, we face terminal cost if the shifts go vacant, as given by $C_H(x_H) = G * \min(L - \sum_{i \in \mathcal{E}} x_k^{si}, 0)$. This represents the loss of opportunity when we compare it to the objective of the single scenario offline formulation in the E-Companion. However, when choosing actual policies, the objective function that our industrial partner wants to optimize is discussed in Section 4.2.

4.1.2 Designing Policy Function Approximations

To tackle the complexity of the DNTP, we focus on designing approximate policies using Policy Function Approximations (PFA) as outlined by [Powell, 2022]. PFA is a policy search approach that is especially valuable when a straightforward, easy-to-implement policy is desirable. These policies provide a direct mapping from states to actions using an analytical function, thereby eliminating the need to solve complex optimization problems in each decision stage. Examples of PFAs include lookup tables, linear decision rules, monotone threshold policies, and nonlinear models. The central challenge is to develop a policy structure that is well-suited to the specific problem context, balancing simplicity with performance.

For the DNTP, we approximate optimal policies by employing a monotone threshold-based structure. A policy follows a monotone threshold approach when it selects an action a_k only up to a defined threshold Λ_k at decision epoch k. Within this framework, the action a_k is directly influenced by the residual $(\Lambda_k - \lambda_k)$, where λ_k is the current value of a target feature or basis in the state x_k, and Λ_k is the optimal threshold for that feature. A typical example of such a policy is the classic (s, S) inventory policy, where restocking occurs only when inventory drops below s, and replenishment brings it up to S. In this case, the total inventory level serves as the feature driving the threshold. Thus, our task centers on designing effective features and determining their associated optimal thresholds Λ_k. However, finding these optimal thresholds is challenging. We adopt an approximate method to estimate appropriate threshold values to address this problem, optimizing two distinct policies for this problem. This approach allows us to balance computational feasibility with policy performance efficiently.

- **Notify η and Wait w epochs** (NAW(η, w)) - A straightforward two-parameter static policy that sends η notifications every w epochs. The parameters η and w can be adjusted for optimal performance. This state-independent approach does not consider any information from the state and consistently notifies individuals at fixed intervals. Since the parameters remain unchanged over time, we can utilize a simple brute-force search to determine the optimal estimates by minimizing the average cost across all simulation runs. Our industrial partner currently employs this policy.

- **Offline Notification Policy** (ONP) - ONP is a dynamic policy in which the threshold parameters vary over time. This heuristic policy depends solely on the elapsed time since the initiation of the notification process rather than the current state of the system. The target feature utilized is the cumulative number of notifications issued at each decision epoch. Next, we will discuss the algorithm employed to estimate these thresholds.

Algorithm 11 presents pseudo-code for the function that defines estimates for thresholds Λ_k for the ONP. Here, Λ_k represents the cumulative number of notifications that should have been sent by epoch k. The instance parameters, the set of offline instances Ω, and an aggregator function q are provided as inputs. Algorithm 11 first generates and solves a single scenario $\omega \in \Omega$ using MIP_{NTP2}. The solution obtained is then used to compute the feature value λ_k^ω e.g., the number of people notified, for each decision epoch k of scenario ω. We calculate these features for all instances in the scenario set. The aggregator function q is then applied to these feature values from the exact offline solutions, generating approximate thresholds $\bar{\Lambda}_k = q([\lambda_k^\omega]_{\omega \in \{1,\ldots,|\Omega|\}})$ across all instances for each decision epoch k. The function outputs estimates of the threshold values for each decision epoch, represented as $\bar{\Lambda} = [\bar{\Lambda}_k]_{k \in \mathcal{K}}$.

Algorithm 11: An algorithm to estimate the threshold for all features

```
Function Compile(M, L, H, D, Ω, q):
    for ω ∈ {1, ..., |Ω|} do
        Solve instance with realization r_ω using MIP_NTP2;
        Compute the value of each feature λ^ω = [λ_k^ω]_{k∈K};
    end
    for k ∈ K do
        Λ̄_k = q([λ_k^ω]_{ω∈{1,...,|Ω|}});
    end
    return Λ̄
end
```

In any decision epoch k, the decision $a = \lfloor \bar{\Lambda}_k - \lambda_k^\omega \rceil$ is the residual between the calculated target number of notifications $\bar{\Lambda}_k$ to be sent from offline solutions and its value in the current state, λ_k^ω for scenario ω. $\lfloor . \rceil$ denotes the nearest integer after rounding.

Algorithm 11 that designs the ONP offers a practical and computationally efficient heuristic solution to the stochastic problem MIP_{DNTP-S}. Given a scenario set Ω, one would use MIP_{DNTP-S} to obtain a near-optimal policy. However, even for a small number of scenarios, this formulation explodes in size. Algorithm 11 simplifies this complex formulation by decomposing the stochastic problem into smaller, manageable deterministic subproblems. Specifically, Algorithm 11 breaks down various stochastic scenarios into individual deterministic instances and leverages the relative simplicity of solving deterministic notification timing problems rather than directly addressing the full stochastic model. After deriving optimal or near-optimal solutions for each deterministic scenario, the results are aggregated to create a unified notification schedule that smooths out the uncertainty across the different instances considered similar to what the stochastic model will aim to do. Moreover, Algorithm 11 also allows the user to develop different threshold policies for different target features.

The primary goal of this approach is to develop a simple yet dynamic policy with time-varying threshold parameters. Various other threshold policies can be designed for different feature functions. In the literature, offline solutions have often been used to guide the design of effective online policies. For example, [Pham et al., 2023] proposed a prediction-based approach for online dynamic radiotherapy scheduling, where a regression model is trained to map patient arrival patterns to optimal waiting times, leveraging offline solutions with full knowledge of future arrivals. Similarly, [De Filippo et al., 2021] tackled multi-stage optimization under uncertainty by integrating a two-stage offline strategy with an online greedy heuristic, achieving notable improvements in solution quality by enhancing offline/online integration. [Kong et al., 2022] introduced a machine learning model that directly predicts the optimal action using offline solutions, using an energy-based parameterization of the model. For a comprehensive review of methods using offline full information solutions, we refer the reader to [Sadana et al., 2024], [Mandi et al., 2024].

4.2 Experiments

We conducted two sets of experiments to assess our approach. In the first set of experiments, we compare different policies with the weighted objective of potential bumps and vacant shifts. The second experiment aimed to evaluate our proposed methodology when preferences are different, representing the case of actual bumps. All experiments and models were implemented in Python, and we employed GUROBI 10.0 to solve the offline optimization

problem instances. We set a maximum solving time of 4 minutes for each instance. GUROBI typically found the optimal solution quickly, often within seconds for all instances, though occasionally, it required more time to confirm optimality.

4.2.1 Scenarios and Evaluation

We conducted experiments with a fixed planning horizon of 6 hours ($H = 6$ hours) and a constant number of available shifts ($L = 50$). A group of $M = 150$ employees with the same preferences can fill these shifts. In these experiments, we explored two distinct cutoff times: 2 hours and 3 hours. It is generally assumed that employees are aware of this cutoff time, and we anticipate that employees who wish to select a shift will respond before this time limit is reached.

Each experiment was defined by a tuple (M, L, H, D), encompassing the number of employees, available shifts, the planning duration, the probability distribution for employee response delays, and the cutoff time, respectively. This tuple was referred to as the setting for a specific experiment. We developed a simulation model to simulate the operation of the electronic system. This model accepts input parameters, including the notification policy, observes the current state of the system, and then takes action according to the input policy.

Table 4.3 provides details on the breakdown of instances used for the different policies. We solved the stochastic formulation for 200 scenarios. We attempted to solve the formulation for 500 scenarios but encountered memory issues. ONP is trained on a different set of 1000 instances and validated on 500 validation instances. NAW is simulated on the 500 validation instances, and the best parameter is chosen. Finally, all policies are tested on the same 500 instances.

Table 4.3 Breakup of Number of Instances used

Policy	**Train**	**Validation**	**Test**
NAW	-	500	500
ONP	1000		
Stochastic	200	-	

We employ the following methodology to validate our dynamic policies:

- Threshold Estimation: Initially, we tackle the offline solution of 1000 training instances. This allows us to estimate the thresholds for all aggregator functions, forming the basis for our dynamic policies.

- Validation: Subsequently, we subject these policies to simulation using 500 validation instances to assess their performance.
- Selection and Evaluation: In the next step, we carefully select the most effective aggregator functions based on their performance during validation. These selected functions are then evaluated using 500 test instances.

This approach ensures that our chosen policy parameters can be generalized effectively when applied to previously unseen data. For our heuristic policies, we take a comprehensive approach:

- Parameter Exploration: For example, in a CW policy, we explore all possible combinations of parameters, such as η and w, using a predefined set of potential values.
- Performance Evaluation: Each combination of parameters is simulated, and the best-performing parameter set is determined during this evaluation phase, using the test set.

This rigorous process allows us to ensure that our policies, both dynamic and heuristic, are well-tuned and capable of achieving good results across a range of scenarios.

4.2.2 Real World Data

We use real-world data from our industrial partner to evaluate our approach. The shift scheduling system has been in operation for more than five years, and we use data from 2018-2020 for further analysis. Sometimes, the number of shifts required for a given day is available early, and hence, the on-demand scheduling system for that list of shifts can start as early as 4 days before. However, our experiments only consider scheduling personnel on a single day of operation with a six-hour planning period, that is, for shifts that are created only one day before their scheduled start. As described earlier, the management uses a static NAW policy with a fixed set of parameters every time the system starts sending notifications, which we again tune for real-world response delays.

After cleaning the data, there were 23204 data points for the response delay. Figure 4.1, shows the cumulative distribution of the response delays from the cleaned data. Note that we only consider response time with a planning period of a single day. Only 50% respond to the system to choose the shifts, but a large proportion of them do so within one minute of the system notifications. Those employees who do not respond within the horizon have their response delays artificially set to 1000, accounting for a 12-hour duration across the day. The

assumption is that all such employees do not want to select a shift to work. One thing to note is that the system allows employees to choose between vacant shifts even after the end of the planning period. Ideally, management would like to schedule all shifts within the planning period. This plot makes it clear that the number of employees needed to schedule all shifts is at least twice the number of shifts available since only half of the employees respond. Also, surprisingly, we do not see any effect of the cutoff.

Response delays are sampled for each employee using real-world data to generate instances. This data is divided into the train, validation, and testing datasets with the same split as the instances. Also, note that the company has a restriction of 5 notifications per epoch, and hence $W = 5$.

4.2.3 Offline Solution Analysis

Table 11 is used to estimate thresholds of all features for 1000 training instances for each setting. We use descriptive statistics such as the "mean" and "percentiles" as aggregator functions for all features across all instances for each decision epoch. Figure 4.2 shows the variation of cumulative notifications with respect to time when $D = 2, 3$ hours. Only the mean and the 90% percentile are plotted for clarity. Notifications are sent near-linearly for both aggregator functions from the start. However, the rate changes as the horizon nears and notifications are sent faster. This is expected as employees notified in this period will not have sufficient time to bump others. In other words, employees who take a long time to respond are pushed out of the horizon to avoid bumps. The main difference between the two aggregators is that more notifications are sent earlier with 90% aggregation. A shorter cut-off encourages a higher number of notifications being sent with a similar profile.

Table 4.4 shows the average bumps and vacancy across all offline instances for the two different cutoff values. It suggests that given prior knowledge of employee response times, one can almost eliminate bumps from the process without compromising on shifts being left vacant. This is mainly possible because a lot of the employees either respond quickly or do not respond at all. One only needs to manage the few that take long to respond.

Table 4.4 Average Optimal Bumps and Vacant Shifts in the offline solution

D	Bumps	Vacant shifts
2	0.80	0
3	2.23	0

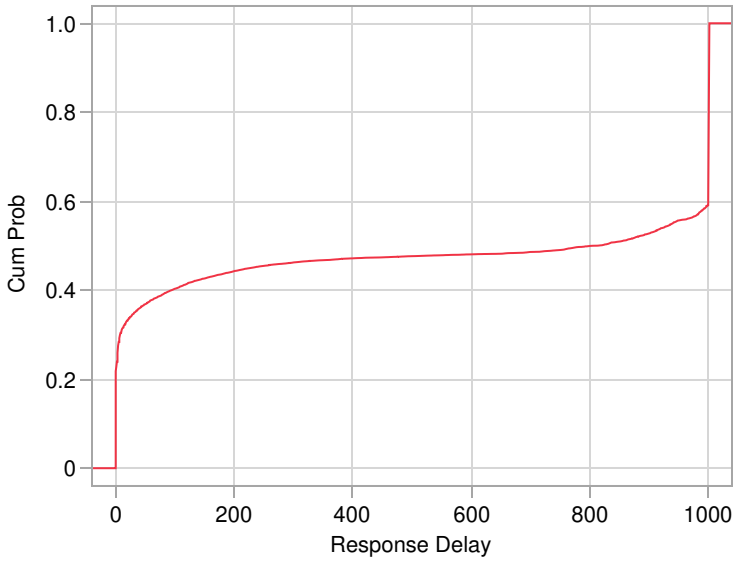


Figure 4.1 CDF plot of Response Delay

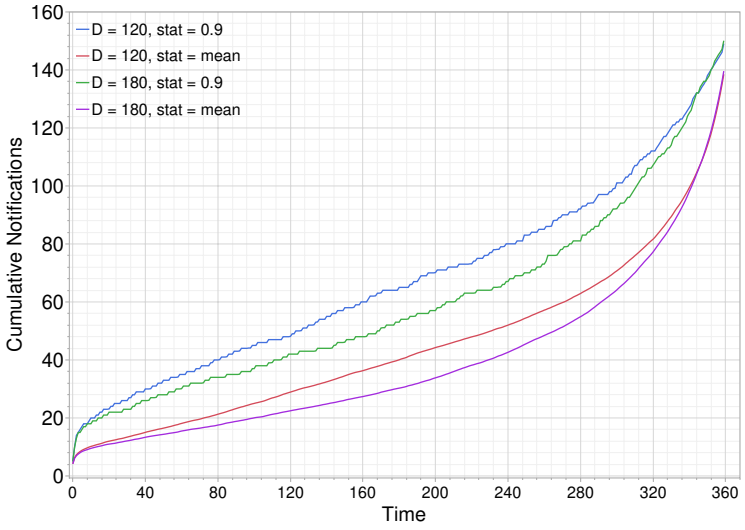


Figure 4.2 Cumulative Notifications Sent

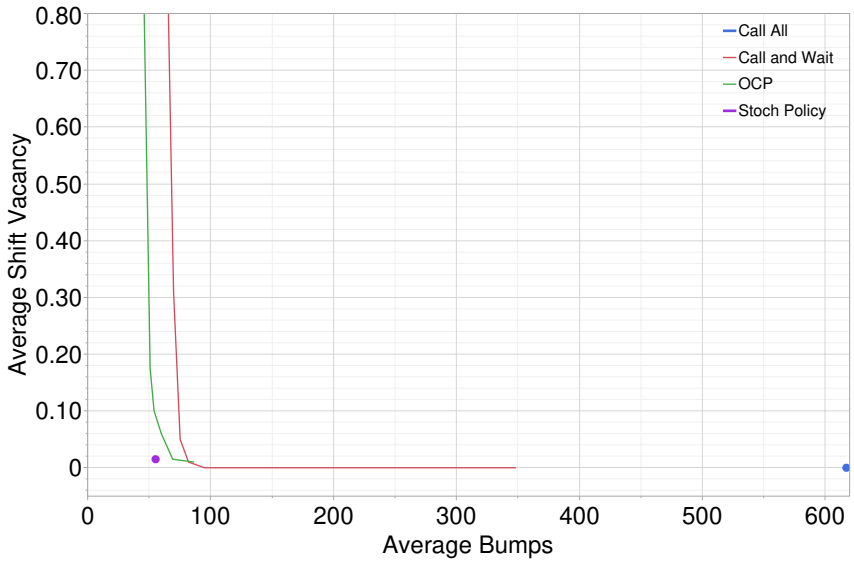


Figure 4.3 Pareto Front of all policies - Shift Vacancy % vs Number of Bumps, $D = 2$

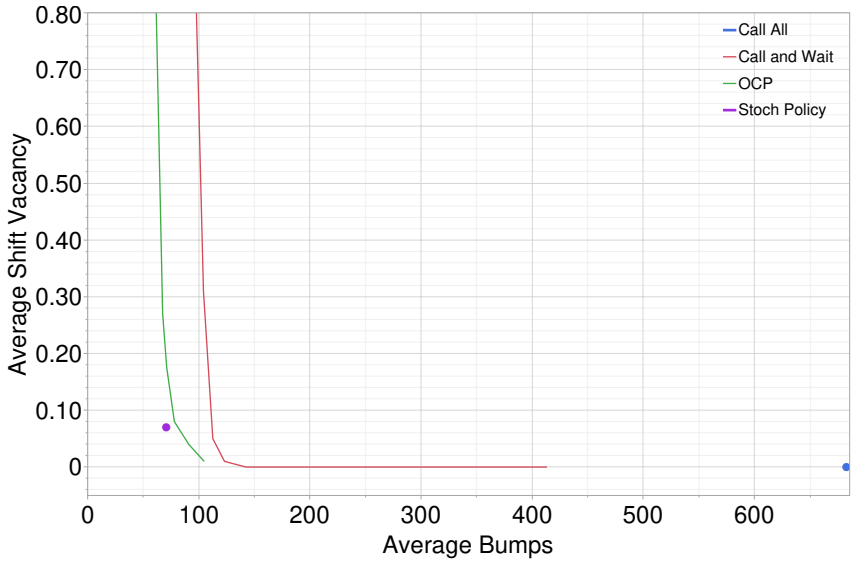


Figure 4.4 Pareto Front of all policies - Shift Vacancy % vs Number of Bumps, $D = 3$

4.2.4 Policy Evaluation - Same Preferences

Solving the stochastic formulation

First, we analyze the performance of a stochastic policy by solving MIP_{DNTP-S} using Stochastic Average Approximation (SAA). We solve the formulation with 200 training scenarios of equal probability for 7 days without any acceleration strategy for the solver. The obtained policy, referred to as the stochastic policy, is then simulated over 500 test instances. Each vacancy is penalised with a cost of $G = 200$. The results are given in Figure 4.5 and Figure 4.6. The average bumps are similar for both the cutoff values for the train and test instances. However, a major difference is seen in the vacancies. The average vacancies increase considerably in the test instances compared to the training instances. They do not meet the 0.3% threshold indicated by the orange line in the graph. There seems to be an opportunity to improve on the number of vacant shifts.

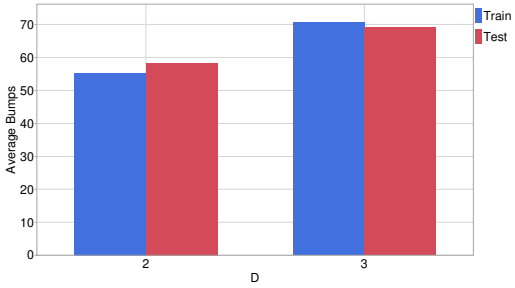


Figure 4.5 Train vs Test performance of MIP_{DNTP-S} - Average Bumps

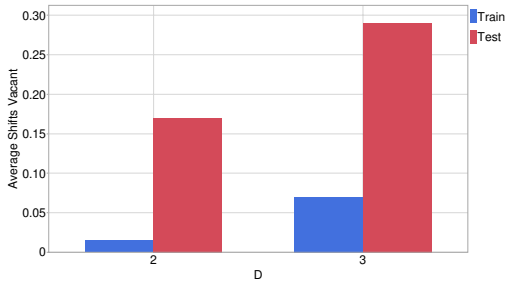


Figure 4.6 Train vs Test performance of MIP_{DNTP-S} - Average Vacant shifts

Comparing Stochastic Policy and PFAs on training instances

Figure 4.3, Figure 4.4 compare all the policies of the different values of D on the 200 training instances that were used as scenarios to solve MIP_{DNTP-S}. The figures plot the average number of bumps suffered by the employees against the average shifts vacant for a policy for a cutoff of 2 and 3 hours. The graphs contain one point for each unique parameter of the policy. In the case of the dynamic policy based on offline solutions, this parameter would be the descriptive statistic used. All policy points are connected by a line to form a Pareto frontier of that policy. It is clear from both plots that the offline solution-based policy is consistently better than the heuristic policy. The stochastic solution obtained from MIP_{DNTP-S} falls very close to the ONP frontier, suggesting we do very well with this policy. We also plot the performance of the Notify All (NA) policy. This policy will notify all employees in one go at the start of the horizon. We use this policy to demonstrate how bad

the system can be if everyone is notified at once. As expected, we see a higher number of bumps when $D = 3$ hours as compared to when $D = 2$ hours. The key takeaway from this plot is that ONP policy can come close to the solution where we have complete information.

Comparing Stochastic Policy and PFA on test instances

The decision-maker has two criteria to work with, reduction of total bumps and allocation of all shifts to employees. To select a policy to use with the electronic system, the management can establish weights for each criterion and choose a policy with minimum cost. Our industrial partner would also like a policy with a maximum shift vacancy of 0.3%, which corresponds to an average vacancy of 0.15 shifts. This vacancy percentage is tolerated because some employees may respond after the horizon ends and some manual changes are made. Also, it is possible to use overtime. However, relying on these options is generally costly. Table 4.5 highlights the best parameters found using the 500 validation instances. For offline solution-based policies, high percentile levels work very well. The intuition here is that for a single instance, the offline solution has information about the future and can decide exactly when to notify someone and when to wait. However, across several instances, one needs to protect against vacant shifts as there is a very high penalty for them.

Table 4.6 gives the performance of these policies with the best-found parameters in terms of total bumps for 500 test instances. ONP works best (indicated by bold) for both the cutoffs of 2 and 3 hours, respectively, in terms of cost. ONP has significantly fewer bumps, slightly compromising on vacancies from Table 4.8. Even the total cost is the least for ONP in Table 4.7. Note that meeting the shift vacancy constraint is more important. The policy obtained from MIP_{DNTP-S} is very poor in terms of shift vacancies. This policy does not meet the management requirements of 0.3% average vacancies highlighted by the red colour in Table 4.6. Thus, this shows that a dynamic threshold policy is better than the current static policy.

Table 4.5 Best Policy Parameters subject to 0.3% shift vacancy

	Policy			
D	**NA**	**NAW**	**ONP**	$\boldsymbol{MIP_{DNTP-S}}$
2	150	3, 7	95	-
3	150	3, 7	95	-

To ensure fairness in comparing the stochastic formulation, we train both the stochastic policy and the ONP using the same number of training instances (200). The results are summarized

Table 4.6 Average Bumps for the best policy on test instances

	Policy			
D	**NA**	**NAW**	**ONP**	$\boldsymbol{MIP_{DNTP-S}}$
2	618.90	83.13	**69.42**	58.28
3	671.3	115.80	**82.63**	69.15

Table 4.7 Average Cost for the best policy on test instances

	Policy			
D	**NA**	**NAW**	**ONP**	$\boldsymbol{MIP_{DNTP-S}}$
2	618.90	88.37	**83.42**	91.88
3	671.3	121.00	**102.23**	127.55

Table 4.8 Average Shift Vacancies for the best policy on test instances

	Policy			
D	**NA**	**NAW**	**ONP**	$\boldsymbol{MIP_{DNTP-S}}$
2	0	0.026	**0.07**	0.17
3	0	0.025	**0.098**	0.29

in Table 4.9. First, in terms of test and train performance for both cutoff values, the test performance is worse than the train performance for both policies. Second, while there is minimal difference in the number of bumps between the training and testing instances for both policies, we observe a notable increase in vacancy in the test cases. This increase is more pronounced for the stochastic policy, which fails to meet the 0.3% vacancy criterion. The ONP, on the other hand, appears to generalize better to unseen instances. This indicates that ONP policies still perform better even on a small training set. Additionally, when comparing ONP trained on 1000 and 200 instances, specifically the bumps and vacancies in Table 4.9, Table 4.6, Table 4.7, and Table 4.8, the results with 1000 training instances are slightly better than those trained on 200 instances.

Table 4.9 Comparing Stochastic Policy and ONP on 200 train and 500 test instances with Average Cost, Bumps and Vacant shifts

	Policy						
		$\boldsymbol{MIP_{DECTP-S}}$			ONP		
	D	**Cost**	**Bumps**	**Vacant Shifts**	**Cost**	**Bumps**	**Vacant Shifts**
200 Training Instances	**2**	**58.26**	55.26	0.015	73.95	69.95	0.02
	3	**84.68**	70.68	0.07	96.15	87.15	0.045
500 Test Instances	**2**	92.28	58.28	0.17	86.80	**70.80**	0.08
	3	127.55	69.15	0.29	105.11	**82.71**	0.11

4.2.5 Policy Evaluation - Different Preferences

One expects that each employee will have different shift preferences in real life. The preferences can be cognitive, cultural, social, temporal, situational, health conscious, or based on familiarity ([Rogers et al., 2003], [Daniel, 2017]). Without real-world preferences, we test some synthetic cases and see how the offline solution-based policies compare with the other policies. Let $\mathcal{L} = \{1, \ldots, L\}$ denote the set of shifts. Also, let $\mathcal{X}_i = \{p_1, \ldots, p_L\}$ denote the preference set of employee i. Thus, if $l = p_o$, it denotes that shift l occurs in position $o \in \{1, \ldots, L\}$ of the preference set. We consider the following preferences:

- **Fixed Ranked Preferences** - This distribution represents the base case where all preference sets are identical.
- **Undesirable Preferences** - Certain employees may dislike certain shifts. We assume that each employee randomly disapproves of $\bar{l}$ shifts from the full set. Starting from a fixed ranked preference set, we randomly move $\bar{l}$ shifts to the end of the preference list. These undesirable shifts remain consistent for any employee across all simulations.
- **Grouped Preferences** - Shifts are assumed to be divided into two groups of equal sizes. Any employee strictly prefers shifts from one set over the other. Within a group, all shifts follow the same preference order for all employees. If the shifts from the preferred group are exhausted, the employee can choose from the other group. The preference distribution represents where shifts can be divided into day and night shifts, and certain employees prefer one over the other. Each employee chooses a group with the same probability.
- **Perturbed Preferences** - We assume that there is still a general fixed ranked order of the shifts from 0 to L. However, the shift can deviate slightly from the order for a given employee. Specifically, any shift l can occur in position $o \in \{l-4, \ldots, l+4\}$ with equal probability. In other words, we have a sliding window around the original shift position for the shift.
- **Perturbed Preferences with Undesirable shifts** - This represents a combination of perturbed preferences and undesirable shifts.
- **Uniform Preferences** - Employees are equally likely to choose any shift from those available. This represents a very unrealistic scenario, but we use it as a baseline scenario.

The best parameters for each preference distribution across all policies are provided in Table 4.10, subject to the maximum vacancy constraint of 0.3%. The various preferences are listed roughly in the order of randomness observed, starting from fixed-ranked preferences. Consequently, we observe that a greater number of employees can be notified across all policies, as the best parameters generally increase for the same policy. Table 4.11 presents the average number of bumps for all policies, subject to the shift vacancy constraint. Note that all bumps in this case represent actual realised bumps. The ONP continues to perform the best across all preference distributions. However, as randomness increases, the performance gain over NAW diminishes. If employees behaved randomly (Uniform), the total average number of bumps would be very low. In such a scenario, one would notify the maximum number of employees, contrasting with the worst-case scenario of the same preferences. The table suggests that our designed policy will result in fewer bumps compared to the current NAW policy. Overall, ONP still proves to be very robust even with different preferences and hence suggests such a policy may be fruitful if implemented.

Table 4.10 Best Policy Parameters that minimize bumps in average subject to 0.3% shift vacancy on validation instances for real-world data

		Policy		
D	**Preference**	**NA**	**NAW**	**ONP**
2	**Fixed Ranked**	150	3, 7	95
	Undesirable	150	3, 7	95
	Grouped	150	2, 4	95
	Perturbed	150	2, 4	98
	Per. w. Undesirable	150	2, 4	98
	Uniform	150	3, 6	98
3	**Fixed Ranked**	150	3, 7	95
	Undesirable	150	3, 7	98
	Grouped	150	1, 2	98
	Perturbed	150	2, 4	98
	Per. w. Undesirable	150	4, 8	98
	Uniform	150	4, 8	98

The red color indicates policies that do not meet the maximum 0.3% shift vacancy requirement.

Table 4.11 Average bumps for each policy on real data-based test instances across various preference distributions

		Policy			
D	**Preference**	**NA**	**NAW**	**ONP**	MIP_{DNTP-S}
2	**Fixed Ranked**	623.73	83.86	**70.85**	54.56
	Undesirable	424.65	50.20	**44.50**	37.42
	Grouped	347.05	43.25	**33.93**	29.28
	Perturbed	225.46	28.93	**27.35**	19.89
	Per. w. Undesirable	197.50	25.30	**23.78**	17.21
	Uniform	57.71	5.23	**4.12**	4.20
3	**Fixed Ranked**	678.97	118.32	**84.84**	64.87
	Undesirable	464.5	67.72	**63.00**	44.27
	Grouped	380.47	56.08	**48.24**	35.09
	Perturbed	246.73	37.74	**32.69**	23.16
	Per. w. Undesirable	216.41	33.44	**28.25**	20.35
	Uniform	65.30	6.45	**5.18**	5.17

4.2.6 Managerial Insights

Experimental results provide important insights for decision-makers. While we established that the offline Notification Timing Problem (NTP) is challenging, having complete information allows us to nearly eliminate scheduling bumps compared to the stochastic NTP. Our comparison of different notification policies, particularly the performance of the Optimal Notification Policy (ONP), shows a significant reduction in bumps and shift vacancies as compared to the policy currently in use. This highlights the effectiveness of more advanced policies and suggests that managers should adopt them for better scheduling outcomes.

Analysis of real-world data revealed that only 20% of employees respond promptly (within 5 minutes) to shift notifications, while about 50% do not respond at all. If such early and slow responders could be identified, they could be targeted by adapting the ONP accordingly. Systems that can effectively segment employees by response time and tailor the notification strategy will see improved shift-filling rates. ONP remains scalable and can be redone without considerable effort as it is compiled offline.

4.3 Conclusion

In this chapter, we deal with a dynamic version of the NTP. We first give an MIP formulation for this problem. Further, we propose a PFA with a threshold structure for the dynamic

version of the problem. Our approach uses an estimate of the threshold obtained by solving several instances offline and aggregating them using easy-to-compute descriptive statistics at each decision epoch. The threshold estimates are tested and validated on unseen instances. We develop the policy on the worst-case assumption of the same preferences. This approach shows superior results considering the same preferences and synthetic preference distributions.

CHAPTER 5 THE TAXONOMY OF THE EXPERT

Sequential Decision Problems (SDPs) [Powell, 2022], as discussed in Section 1.1.1, represent a class of decision-making problems under uncertainty. These problems involve making sequential decisions over multiple stages, where the rewards or costs are influenced not only by the immediate action but also by the actions taken in previous stages. Even when the underlying uncertainty distributions are known, solving SDPs exactly remains intractable due to the well-known curse of dimensionality. This intractability is evident in the NTP discussed in Chapter 4, where addressing the stochastic problem in its exact form is computationally prohibitive.

Imitation learning offers a promising avenue to mitigate this challenge by leveraging ML models to directly predict actions at each decision epoch. In this approach, the ML model is trained offline using input features derived from states encountered at various decision epochs, mapping them to corresponding actions. Once trained, the model can be deployed online, allowing for efficient decision-making without the need for extensive computation. This offline training framework is particularly advantageous as it shifts the computational burden to the offline phase, enabling rapid inference during real-time operations.

The effectiveness of this supervised learning strategy hinges on the availability of high-quality experts or target actions for training. Ideally, these target actions represent the optimal decisions that account for all uncertainty given the current state. However, obtaining optimal actions, even in an offline setting, is computationally expensive and often impractical. Consequently, approximate expert actions, which are both high-quality and easier to derive, are frequently used as substitutes.

One potential source of such approximate targets is the optimal offline solution, derived with the benefit of hindsight regarding future uncertainty. Offline problems of this nature are often more tractable as they avoid the dimensionality challenges associated with large outcome spaces. These solutions can be generated by constructing contextual scenarios from the current state and solving the corresponding offline problem. The resulting solutions then serve as targets for the imitation learning process, effectively approximating the exact SDP at a given epoch.

Alternatively, instead of solving individual scenarios independently, one could consider a stochastic program that integrates all scenarios simultaneously to produce a unified optimal offline solution. This solution, which optimizes over all scenarios, can serve as a robust target for imitation learning. The choice of approximation method for the target generation strategy

can be tailored to the specific problem and computational resources, offering flexibility in designing effective learning-based approaches to approximate SDPs.

A natural question that may arise in the reader's mind is: If one can design a suitable approximate expert, can it be directly used in real-time? Why should one use IL? There are several reasons why IL can be useful. In a dynamic SDP, obtaining suitable actions is often a tradeoff between speed and quality. Since the uncertainty set can be huge, the actual problem may be difficult. Even if we only consider a small number of scenarios, the problem remains difficult to solve in real-time. Also, the decisions obtained may be biased to the scenarios used. Hence, there is always a chance of making a really bad decision due to rare scenarios. If one builds a model offline, these rare scenarios act as outliers and may be filtered out.

The concept of imitating a known target is relatively novel within the field of OR. To the best of our knowledge, the existing literature lacks a clear and comprehensive taxonomy for defining the expert and how to use it within an IL framework. Recognizing this gap, we have dedicated efforts to establishing a taxonomy for the expert, aiming to provide a structured foundation for their use in IL methodologies.

5.1 The Taxonomy on the Expert

An expert is an individual who possesses a deep understanding of a problem, backed by extensive research or practical experience and knows how to proceed in any given situation to achieve a goal. This goal could be to maintain feasibility, optimize a particular objective, or meet specific constraints. Other names for the expert include *Oracle*, *Demonstrator*, *Optimal Policy*, *Ground Truth*, *Reference Policy*, *Target*. In the context of Imitation Learning, high-quality expert demonstrations are essential for effectively guiding a model or agent to learn the desired behavior or task. The quality of these expert demonstrations plays a crucial role in the success of the learning process. Inaccurate or suboptimal expert actions can lead to poor learning outcomes and, in some cases, even dangerous behaviors. For instance, in autonomous vehicles, mimicking suboptimal expert demonstrations could result in unsafe actions, putting human lives at risk.

However, obtaining high-quality, optimal demonstrations is often a challenging task. Even for domain experts, certain complex tasks—such as high-frequency stock trading or playing intricate video games—are difficult to perform optimally. This challenge extends to OR problems, which are often mathematically complex and require sophisticated algorithms for optimal solutions. In such cases, the optimal solutions generated by state-of-the-art optimization solvers can serve as expert actions. Ideally, one would like to solve (1.4) to get the

optimal action. The equation is rewritten below by

$$a_k^* = \arg\min_{a_k} \mathbb{E}\Big\{ C_k(x_k, a_k) + V_{k+1}(x_{k+1}) | x_k \Big\} \tag{5.1}$$

We use the same notations as defined in Chapter 1, where x_k denotes the state, a_k denotes the action, and C_k denotes the cost at epoch k. V_k represents the total cost from epoch k.

However, for many real-world business problems, obtaining these true optimal actions may be impractical (even offline) due to the complexity of the tasks involved. In these situations, the use of suboptimal/near-optimal actions becomes necessary. In OR, it is common to design algorithms that provide near-optimal solutions—decisions that may not be the absolute best but are sufficiently close to the optimal solution to be practically useful. These near-optimal solutions, if obtained within a sufficient amount of computational time, can be effectively employed within the IL framework as expert demonstrations. Given this approach, various methods can be devised to obtain these near-optimal solutions. This section aims to classify different types of expert actions based on how they are obtained, their usage in the IL framework, and the information utilized in the context of dynamic SDPs for IL. We also try to classify some relevant literature that tries to imitate a policy within the taxonomy for OR problems. We consider the following articles [Pham et al., 2023], [Larsen et al., 2018], [Mandi et al., 2020], [Baty et al., 2024], [Greif et al., 2024], [Rautenstrauß and Schiffer, 2025].

5.1.1 Expert Type

Human Expert

Humans can serve as domain experts, possessing deep knowledge and expertise in specific fields or tasks. Take the dynamic vehicle routing problem, where drivers need to make deliveries. Human dispatchers or logistics planners can demonstrate how they plan the vehicle route for delivery under uncertain conditions (e.g., traffic delays and customer time windows). Even drivers make decisions in real-time based on environmental conditions, using critical contextual information. An example application can be found in [Özarık et al., 2024] where the aim is to learn a supervised ML model that imitates domain experts who are experienced in their delivery zones and have an intuitive understanding of the area, infrastructure, and customers they serve. In hospitals and healthcare systems, administrators and clinicians face complex resource allocation decisions (e.g., scheduling surgeries and managing ICU beds) under uncertainty. These decisions can be used to train ML models that replicate expert

routing strategies. Human experts have a strong understanding of real-world subtleties and are often better equipped to handle unexpected situations. However, human decisions can be inconsistent or suboptimal due to cognitive biases, fatigue, or incomplete information. Replicating these inconsistencies can lead to suboptimal learning models. In specialized fields like healthcare or large-scale manufacturing, it can also be challenging to find enough experts to provide sufficient demonstrations for training models.

Algorithmic/Optimization Expert

An algorithmic/optimization expert in the context of imitation learning refers to a computational system or algorithm that is used to generate optimal or near-optimal demonstrations for training learning agents. Instead of relying on human expertise, these experts use predefined rules, optimization models, or heuristic approaches to solve problems and generate expert-level solutions. Algorithmic experts are particularly useful when the task is well-defined and can be mathematically modelled, making them an ideal source for providing consistent, large-scale demonstrations. The main advantage of these experts is their ability to generate large-scale datasets and handle complexity.

The optimization expert is further classified into different categories based on the models they use. Each of these experts defines an action to be taken with respect to the objective function they optimize. We use $C(.)$ to denote the total cost function for each of these experts with the starting state as x_0 of the problem. Also, let $\mathcal{A}$ denote the space of all feasible actions for the problem.

1. **Heuristic** - A heuristic algorithm is an approach to solving complex problems by using practical rules or approximations to find good, but not necessarily optimal, solutions efficiently. It prioritizes speed and feasibility over exactness, making it useful for problems where finding an optimal solution is computationally expensive. Common types include greedy algorithms, local search, and metaheuristics like genetic algorithms and simulated annealing. The heuristic algorithm generally does not optimize the actual objective directly. Let a^h denote the action suggested by a heuristic.

2. **Myopic model** - A myopic model refers to a decision-making model that optimizes for short-term objectives without considering long-term consequences or future uncertainties solely based on the current state. Since they do not require forecasting or modelling future states, myopic models are generally computationally simpler and faster to solve than long-term or dynamic models. We define the expert myopic action as a^m:

$$\mathbf{a^m} = \arg\min_{a \in \mathcal{A}} \left\{ C(\mathbf{a}) | x_0 \right\} \tag{5.2}$$

The objective function we optimize simply considers all uncertainty revealed until the current epoch and greedily optimizes. The advantage is that these models enable rapid decision-making, which can be useful; however, the quality of this action is generally expected to be poor.

3. **Full-Information/Hindsight model** - A Full-Information/Hindsight model is an optimization framework where decisions are made as if the decision-maker had perfect knowledge of all future uncertainties or outcomes. We again use point values $\tilde{\boldsymbol{\xi}}$ to represent $\boldsymbol{\xi}$; however, we assume that we know the information beforehand. Alternatively, it is also referred to as hindsight optimization since, in reality, we cannot have this information before it is realised. In the end, we solve a problem with the same structure as the deterministic model. The key difference as compared to a deterministic OR expert is that the deterministic expert will use estimates or predictions $\hat{\boldsymbol{\xi}}$ while a Full Information expert will use realisations denoted by $\tilde{\boldsymbol{\xi}}$. Let a^o represent the optimal full information action given by

$$\mathbf{a^o} = \arg\min_{a \in \mathcal{A}} \left\{ C(\mathbf{a}, \tilde{\boldsymbol{\xi}}) | x_0 \right\} \tag{5.3}$$

4. **Deterministic model** - A deterministic model uses point estimates for uncertainty ξ. This means decisions are based on fixed or expected values of uncertain parameters rather than treating them as random variables with distributions. This approach assumes that we have access to historical data for the uncertain inputs. This allows the development of sophisticated prediction or forecasting models (g_θ) that give reliable single-point estimates $\hat{\boldsymbol{\xi}}$ for the uncertain parameters $\boldsymbol{\xi}$. In OR, this approach is widely referred to as *predict-then-optimize*. Once we have the estimates $\hat{\boldsymbol{\xi}}$, we can directly solve the deterministic problem in equation (5.4). In many instances, the problem in equation (5.4) can be represented as a mathematical program (Linear or Integer), and the OR community has developed efficient algorithms to solve it. Consider the ECTP; if we can predict all employee response times beforehand, we can solve the offline model to find the exact notification times for everyone. The optimal action a^p is referred to as the optimal *hindsight* action or optimal *anticipative* action.

$$\mathbf{a^p} = \arg\min_{a \in \mathcal{A}} \left\{ C(\mathbf{a}, \hat{\boldsymbol{\xi}}) | x_0 \right\} \tag{5.4}$$

A naive example of this is simply using mean values to represent $\boldsymbol{\xi}$. Many recent studies focus on the problem of estimating $\boldsymbol{\xi}$, which falls under the category of supervised machine learning problems. A set of past observation pairs $(x_i, \xi_i)_{i=1}^N$ is available as a training dataset, D, and is used to train an ML model g_θ (with trainable ML parameters θ), so that parameter predictions take the form $\hat{\xi} = g_\theta(x)$. The decisions $\mathbf{a}^p$ in (5.4) are referred to in the literature [Mandi et al., 2024] as *prescriptive decisions*.

5. **Two-Stage Stochastic model** - This represents a stochastic model where decisions are made in two stages, considering uncertainty in the problem. The problem is described in equation (5.5). The key feature of these models is that the first-stage decisions (here-and-now decisions) are made before the uncertain outcomes are known. In contrast, second-stage decisions (recourse actions) are made after the uncertainty is revealed. Some problems may only have first decisions, and in the second stage, only the uncertainty is observed. $C(a)$ represents first-stage costs with first-stage decisions a, while $Q(\mathbf{a}, \boldsymbol{\xi})$ represents second-stage costs taking the uncertainty into account subject to first-stage decisions.

$$\mathbf{a^t} = \arg\min_{a \in \mathcal{A}^1} \left\{ C(\mathbf{a}) + \mathbb{E}_{\mathbb{P}(\xi|x)}[Q(\mathbf{a}, \boldsymbol{\xi})] \right\} | x_0 \tag{5.5}$$

It is common to consider a scenario set Ω and alternatively solve the equation (5.6). However, this problem quickly explodes in size with the number of scenarios one considers.

$$\mathbf{a^t} = \arg\min_{a_1 \in \mathcal{A}^1} \left\{ C(\mathbf{a}) + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} Q(\mathbf{a}, \boldsymbol{\xi}_\omega) \right\} | x_0 \tag{5.6}$$

6. **Multi-Stage Stochastic model** - Multistage stochastic models [Birge and Louveaux, 2011] extend the concept of two-stage stochastic models to situations where decisions must be made over multiple time periods, with uncertainties unfolding over time. In each stage, decisions are made based on the available information, and future decisions are adjusted as new information becomes available. This will thus represent the

true optimal action a^* in the presence of uncertainty given by equation (5.7). Computing a^* is generally more computationally expensive than computing $\mathbf{a^t}$. In many applications, the decision-maker has access to historical data only of the corresponding uncertain parameters, i.e., the decision-maker does not know the true distribution of the uncertainty, making it impossible to have the exact a^*.

$$\mathbf{a}^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}_{\mathbb{P}(\xi|x)} \Big\{ C(\mathbf{a}, \boldsymbol{\xi}) | x_0 \Big\} \tag{5.7}$$

One can write (5.7) as follows to include the objective of each stage

$$\mathbf{a}^* = \arg\min_{a_0 \in \mathcal{A}} \Big\{ C_0(x_0, a_0) + \arg\min_{a_1 \in \mathcal{A}} \mathbb{E}_{\mathbb{P}(\xi_1|x)} \Big\{ C_1(x_1, a_1) + \cdots + \tag{5.8}$$

$$\arg\min_{a_K \in \mathcal{A}} \mathbb{E}_{\mathbb{P}(\xi_K|x)} \Big\{ C_K(x_K, a_K) \Big\} \Big\} \Big\} \tag{5.9}$$

If the problem under study satisfies the Markov property, it is then possible to define a MDP. We can write (5.9) in terms of the value function from the starting state x_0 as follows.

$$\mathbf{a}^* = \arg\min_{a_0} \Big\{ C_0(x_0, a_0) + \mathbb{E}[V_1(x_1)|x_0] \Big\} \tag{5.10}$$

The table 5.1 summarizes all the models.

Expert Action	**Model**	**Objective**	Uncertainty
a^m	Myopic	$\arg\min_{a \in \mathcal{A}} \Big\{ C(\mathbf{a}) \| x_0 \Big\}$	No Uncertainty
a^p	Deterministic	$\arg\min_{a \in \mathcal{A}} \Big\{ C(\mathbf{a}, \hat{\boldsymbol{\xi}}) \| x_0 \Big\}$	Point Estimates - $\hat{\boldsymbol{\xi}}$
a^o	Full Information	$\arg\min_{a \in \mathcal{A}} \Big\{ C(\mathbf{a}, \tilde{\boldsymbol{\xi}}) \| x_0 \Big\}$	Samples - $\tilde{\boldsymbol{\xi}}$
a^t	Two-Stage Stochastic	$\arg\min_{a \in \mathcal{A}^1} \Big\{ C(\mathbf{a}) + \mathbb{E}_{\mathbb{P}(\xi\|x)}[Q(\mathbf{a}, \boldsymbol{\xi})] \Big\} \| x_0$	Uncertainty Set - Revealed at Once
a^*	Multi-Stage Stochastic	$\arg\min_{a \in \mathcal{A}} \mathbb{E}_{\mathbb{P}(\xi\|x)} \Big\{ C(\mathbf{a}, \boldsymbol{\xi}) \| x_0 \Big\}$	Full Uncertainty - Revealed Sequentially

Table 5.1 Summary - Expert Models

[Pham et al., 2023] and [Larsen et al., 2018] employ the Full-Information Expert (FIE) to directly learn from its guidance, leveraging its complete knowledge to improve decision-making. Similarly, [Mandi et al., 2020], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and

Schiffer, 2025] integrate FIE into their learning pipelines, where it serves as a reference for defining regret, which is subsequently minimized during training. Among these, [Greif et al., 2024] is the only work that explicitly describes an idea conceptually similar to the use of the ADE. This is referred to as the voting policy in [Greif et al., 2024], where several scenarios of uncertainty from the current state are sampled and then solved independently. Finally, all solutions are made to vote, and the most frequent decision is selected. A similar idea was also used in [Bent and Van Hentenryck, 2004], where the authors use a a multiple scenario approach. The key idea is to continuously generate and solve scenarios. The final decisions or plans are selected by a consensus function that selects the plan most similar to the current pool or plans.

5.1.2 Policy Dynamics Spectrum

The classification of policies based on their temporal adaptability can aid in understanding their applications and limitations in expert-driven decision-making processes.

Policy Learned from the expert

- **Static/First Stage Policy** - A static/First Stage policy is fixed and does not change over time or across different decision points. Once it is determined, it is applied uniformly throughout the decision-making process. These are generally easier to design and implement since decisions do not depend on real-time information and have low computational costs during execution. ONP for NTP represents a static policy that is pre-calculated using offline solutions. The offline solutions are calculated, generating the entire future trajectory at once. This policy is static because we precisely know the number of employees that are going to be notified at any point in the horizon. Furthermore, some problems only involve a 2-stage decision process. In such a case, one can choose to learn a first-stage policy.
- **Dynamic Policy** - A dynamic policy changes over time, based on the evolving state of the system or new information. It is more adaptive and can respond to different scenarios.

[Pham et al., 2023], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and Schiffer, 2025] all study a dynamic problem, and thus their learning algorithm produces a dynamic policy that accounts for the state seen at each epoch. [Larsen et al., 2018], [Mandi et al., 2020], on the other hand, study a two-stage problem and thus predict action for the first stage only. [Larsen et al., 2018] predicts the first stage tactical decision for a load planning

problem. [Mandi et al., 2020] considers two different problems. First, a stochastic knapsack problem with random item values. Second, a resource-constrained job scheduling problem for total energy cost minimization given randomness in energy usage for scheduling each job.

Policy Variable

The policy variable indicates whether the expert outputs direct actions or intermediate features, which subsequently guide decision-making. This distinction affects how the model interprets and utilizes the expert's guidance.

- **Direct decision-based policy** - In this case, the expert provides a direct mapping from states to actions, producing specific actions without intermediary steps. This approach is efficient for real-time implementation and can simplify the decision process by minimizing interpretive layers. For instance, in the NTP, a direct decision-based expert would specify the number of notifications to be sent at each time step, providing a straightforward, actionable rule.
- **Feature-based policy** - The expert may generate intermediate features or latent variables that are used by another system to make the final decision. The decision-making process is broken into two steps — feature extraction and action selection. As an example, for the NTP, we can have an expert who tells us how many total notifications should have been at the current epoch, observing the current state of the system. A final decision on the number of notifications to be sent can then be made. Alternatively, one can also think of any model g_θ that predicts point estimates for the response delays as an expert prediction. Then, we solve a simpler optimization problem to get the actual action to be executed.

In [Larsen et al., 2018] and [Pham et al., 2023], the authors train a model that directly predicts the optimal action to take in a given state, bypassing the need for an intermediate representation of uncertainty. In contrast, [Mandi et al., 2020], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and Schiffer, 2025] first generate predictions for uncertainty and then formulate an optimization problem to determine the final decision.

Specifically, [Greif et al., 2024] and [Baty et al., 2024] predict prize values related to random demand in their respective problem settings. Given these predicted prizes, they then solve a deterministic optimization problem to derive the optimal decisions. Similarly, [Mandi et al., 2020] predicts item values in the knapsack problem and energy costs in job scheduling, using these estimates to inform decision-making through an optimization-based approach. This

distinction highlights two different methodologies: direct action prediction versus uncertainty estimation followed by optimization, each with its own advantages depending on the problem structure and available data.

5.1.3 Optimality of Expert Actions

The quality of expert actions can vary based on how closely they align with an ideal solution to a decision problem. We categorize these actions by their degree of optimality relative to a well-defined objective function.

- **True Optimal Action** - A true optimal action is the best possible action according to a clearly defined objective function. Such an action is achievable only when the policy has complete knowledge of the environment and the underlying dynamics. In the context of an SDP, the true optimal action can be represented by equation (1.3).
- **Sub-optimal Action** - We define sub-optimal action as the solution that, while optimal for a specific objective, does not achieve the absolute global optimum. Actions defined by equations (5.2) - (5.5) fall into this category, as they represent the best solutions for their specific objectives but not the true optimal action for the actual objective given by (5.1). For example, consider the Full-Information expert. It can solve the deterministic problem under full information to optimality. This action is optimal for the problem considered, but it is not the true optimal action.
- **Approximate Action** - An approximate action aims to closely approximate an optimal or sub-optimal action, often through approximation techniques that balance accuracy with computational feasibility. Many decision problems, particularly complex SDPs, cannot be solved exactly; therefore, approximate algorithms are employed. For example, problems expressed by equations (5.2) - (5.5) may not be solved to optimality but can be approached through approximation.

[Larsen et al., 2018] and [Pham et al., 2023] employ different strategies to solve their deterministic optimization problems, using either an optimality gap or a time limit, respectively. These approaches ensure computational efficiency while still producing high-quality solutions.

[Baty et al., 2024] tackles the deterministic problem—a Vehicle Routing Problem with Time Windows (VRPTW)—using a metaheuristic, which provides approximate but effective solutions for imitation. In contrast, [Greif et al., 2024] solves a Mixed-Integer Linear Program (MILP) for the Capacitated Prize Collecting Traveling Salesman Problem (CPC-TSP),

leveraging exact optimization techniques to derive high-quality decisions. [Rautenstrauß and Schiffer, 2025] solves the offline problem to obtain optimal offline decisions.

[Mandi et al., 2020] solves knapsack problems to optimality, ensuring precise decision-making in this setting. However, for the more challenging resource-constrained job scheduling problem, they resort to solving a relaxation, balancing computational feasibility with solution quality.

5.1.4 Information Availability

The type and quality of information available to an expert significantly affect their ability to make optimal decisions. Different categories of information availability can help classify experts based on the level of uncertainty and the constraints they operate under.

1. **Complete** - A complete information expert has access to comprehensive and accurate information about the system, environment, or task, including all possible outcomes or the true distribution of uncertainties. With this complete information, the expert can theoretically provide optimal decisions. However, such experts are mostly hypothetical, as real-world Stochastic Decision Problems (SDPs) inherently involve unknowns or uncertainties.

2. **Partial/Limited** - A partial information expert has access to only a subset of relevant information or operates with highly restricted features within the state variable, which may limit their ability to make fully informed decisions. This expert may lack access to certain variables or possess incomplete knowledge of the environment, potentially leading to suboptimal actions. In the absence of the true distribution of uncertainties, an expert relying on available data operates with partial information, making decisions based on incomplete inputs. For example, in the case of a self-driving car, the agent may only receive information about the state variables visible in front of the vehicle via cameras. However, due to the absence of rear cameras, the car lacks crucial state information about its surroundings behind it. Additionally, the expert can also intentionally disregard the information provided by rear cameras while making decisions despite having access to such data. A possible reason to ignore such information is to reduce the complexity of the problem. The Full-Information model works under such a setting, as it only uses the knowledge from one specific scenario.

3. **Noisy** - A noisy information expert receives inputs that may be corrupted or contain measurement errors, complicating accurate decision-making. For example, an expert

using sensor-based data, like GPS, might encounter inaccuracies due to sensor noise or faults, which can affect prediction accuracy or action reliability.

4. **Delayed** - A delayed information expert functions in environments where there is a time lag between data generation and data availability. This delay introduces additional challenges, as decisions may no longer be optimal when finally executed. For instance, a traffic control system receiving delayed vehicle position updates might lead to suboptimal routing and coordination actions.

[Mandi et al., 2020], [Larsen et al., 2018], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and Schiffer, 2025] also tackle Full-Information problems while operating under limited-information settings.

5.1.5 Expert Action Reproducibility

The reproducibility of expert actions refers to the consistency with which an expert provides the same action given a particular state. We categorize expert actions into two main types:

- **Consistent** - The expert consistently produces the same action for a given state, a behavior typically observed when an agent is guided by established heuristics or rules designed to simplify decision-making, with no inherent randomness. Algorithmic models, such as Myopic and Deterministic policies, also exhibit this characteristic, consistently generating identical actions for identical states when the solver is deterministic and no symmetries are present in the problem.
- **Inconsistent** - In this case, the expert produces varying actions for the same state. There are several possible reasons for such inconsistency:
  - **Imperfect Expertise**: Some expert models, such as the Full-Information Expert or the scenario-based Two-Stage Expert, may exhibit inconsistencies due to variations in context or the information available to them. This phenomenon is also evident in human behavior. For instance, in the context of driving, different drivers may adopt different driving policies. While some may exhibit a highly performant but aggressive style, others may prefer a safer, more conservative approach.
  - **Randomized Optimal Policy**: In some instances, the optimal policy is designed to be randomized over a set of actions, meaning that for the same state, multiple actions could be optimal with varying probabilities [Rajaraman et al., 2020]. Alternatively, this can also be interpreted as the existence of symmetric optimal solutions to the problem.

- **Randomization in Solution Methodology**: Certain optimization algorithms like metaheuristics, particularly those that incorporate stochastic elements, may rely on the generation of random numbers. As a result, these algorithms can produce different solutions each time they are executed, even when applied to the same problem, due to the inherent randomness in their processes.
- **Approximation in Algorithmic Experts**: Some algorithmic experts are approximated rather than exact, often by setting a time limit in optimization solvers or by using heuristic approaches. This can result in different actions when the approximation or solution method is invoked multiple times for the same state.

[Mandi et al., 2020] and [Larsen et al., 2018] focus on a two-stage problem, and generate decisions based on the Full-Information expert. [Pham et al., 2023], [Baty et al., 2024], [Greif et al., 2024], [Rautenstrauß and Schiffer, 2025] consider dynamic problems, and the demonstrations from the expert are also subject to the Full-Information scenario. Thus, the experts for all cases are inconsistent for the current state. Moreover, the resulting demonstrations are also inconsistent due to approximation errors when problems are solved with a Gap or time limit, as in [Larsen et al., 2018], [Pham et al., 2023]. [Pham et al., 2023] is also known to have symmetry in optimal solutions.

5.1.6 Expert Usage

This point outlines various strategies for incorporating expert demonstrations into the learning process. Depending on the cost, availability, and reliability of the expert, different approaches can be employed to leverage expert knowledge effectively. Experts can be used to collect demonstrations in the following ways.

- **Non-Iterative Usage** - In this approach, the expert is consulted only once, typically at the start of the learning process, to label all observed states in a batch. The collected data is then used to train a model that is ready for deployment. This method is commonly applied when invoking the expert is highly resource-intensive or time-consuming, as it minimizes the number of calls to the expert.
- **Iterative Usage** - In contrast, iterative usage allows for repeated interactions with the expert throughout the training process, akin to the DAgger (Dataset Aggregation) framework. Iterative algorithms help address limitations of initial imitation learning (IL) models by allowing the model to learn from errors as they arise, gradually improving performance. Within iterative usage, expert interaction can follow one of two patterns:

Additionally, the expert can be used in the following way to label the states seen within an iterative or a non-iterative algorithm.

- **Single Complete Usage** - Here, the expert is used to label all observed states in each iteration of the loop. At the end of each episode, the accumulated data is used to train an updated model. This approach is feasible when the expert can provide labels quickly, as frequent interaction allows the model to adapt and refine its performance based on comprehensive feedback from the expert.
- **Multiple Complete Usage** - In practice, experts may not always provide the same action for a given state due to imperfections or noise in their decision-making. In this case, instead of relying on a single recommendation, the learner can query the expert multiple times and aggregate the actions suggested. This helps mitigate the impact of errors or inconsistencies in the expert's advice, leading to more reliable decisions. Common aggregation strategies include majority voting or averaging over the set of recommendations.
- **Targeted Usage** - When calling the expert is costly, a targeted approach can be more practical. In this case, the expert is only consulted for specific scenarios where the model underperforms or encounters entirely new states. By focusing expert demonstrations on challenging or unfamiliar cases, the model can learn to handle edge cases or rectify errors without excessive reliance on the expert, which helps balance accuracy with efficiency.

[Pham et al., 2023], [Larsen et al., 2018], [Baty et al., 2024], and [Mandi et al., 2020] utilize the expert within a Non-Iterative framework, where expert demonstrations are collected upfront and used directly for training without further interaction. In contrast,

[Greif et al., 2024] integrates the expert into an iterative learning algorithm, employing a single complete pass of the DAgger procedure. [Rautenstrauß and Schiffer, 2025] proposes two algorithms. The first is a standard DAgger algorithm, similar in structure to that of [Greif et al., 2024]. The second is an alternative approach designed to enhance the training dataset before model training. This algorithm uses the expert in a non-iterative yet targeted manner to reduce the computational overhead typically associated with DAgger, which requires continual regeneration of training examples during learning.

Specifically, the alternative strategy follows a behavior policy up to a specified point $\tilde{t}$ in the decision horizon. Beyond this point, the expert is invoked to label all subsequent states encountered within the episode. This hybrid approach seeks to balance computational effi-

ciency with the benefit of expert supervision and reduces the total runtime by about 87% as compared to the DAgger algorithm.

5.1.7 Number of Experts Used

Additionally, algorithms can be classified based on the number of experts involved and how their expertise is utilized during the learning process. Below are three key categories that outline the usage of experts in imitation learning:

- **Single Expert** - This is the traditional assumption in imitation learning, where a single, optimal expert provides guidance. The learner imitates this expert's actions with the assumption that the expert is both inexpensive and highly knowledgeable. This approach typically assumes that the expert consistently provides accurate decisions for any given state, which simplifies the learning process but may limit flexibility in complex environments.
- **Multi-Expert** - In some scenarios, multiple experts with different areas of expertise may be available. In such cases, the learning framework can aggregate the actions of all these experts, leveraging their complementary knowledge to create a composite "meta-expert". By pooling the knowledge of several experts, each contributing their strengths for different parts of the task, this approach can result in more robust and adaptable decision-making.

[Pham et al., 2023], [Larsen et al., 2018], [Mandi et al., 2020], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and Schiffer, 2025] all rely on a single expert to label the states encountered during the decision-making process. The expert provides guidance or labels for the states seen, which are then used to train the model or solve the optimization problem.

5.1.8 Learning Control

When experts are used in an iterative framework, they are also used to control trajectories to correct mistakes made by the learned model. This ensures the model learns to act in situations it should encounter during deployment. There are multiple ways to execute this control.

- **Total Expert Control** - The expert controls all states visited in a learning trajectory. An example is pure supervised learning without an iterative learning algorithm, where the expert is just used once to label all states.

- **Regressive Expert Control** - Start with high reliance on the expert and gradually reduce its usage as the model improves, e.g., VANILLA DAgger.
- **Conditional Expert Control** - The expert is selectively invoked based on certain performance thresholds, eg. SAFE DAgger. For instance, if the learned model has low prediction confidence (below a certain threshold), one calls the expert; otherwise, it proceeds autonomously. Alternatively, an external policy can be introduced to help guide trajectories. This is useful when the true optimal expert is unavailable, one may rely on an imperfect expert prone to occasional mistakes.

[Larsen et al., 2018] and [Mandi et al., 2020] do not generate trajectories, as they focus on a two-stage problem where the decision-making process does not involve sequential steps or evolving states. In contrast, [Pham et al., 2023] and [Baty et al., 2024] allow the expert to have total control over the trajectories generated, meaning the expert directly influences the entire sequence of states and actions during the decision-making process. [Rautenstrauß and Schiffer, 2025] uses conditional expert control in their algorithm for enhanced dataset generation.

[Greif et al., 2024], on the other hand, employs the Vanilla DAgger decision rule, where the expert has regressive control over the generated trajectories.

5.1.9 Expert Evolution

Expert Evolution refers to scenarios where the expert policy itself evolves or improves during the imitation learning process.

- **Static Expert** - A static expert represents the conventional imitation learning framework, where the expert is assumed to be fixed and generally optimal, providing consistent demonstrations without adaptation. This approach works well when the expert is highly reliable, but it may struggle in environments with evolving dynamics or imperfect expert demonstrations.
- **Adaptive Expert** - An adaptive expert leverages feedback from the learner's behavior to refine their own decision-making process. This approach is particularly valuable when the expert is imperfect and may require improvements over time. For instance, consider the Deterministic Optimization expert, which relies on a single scenario for decision-making. The demonstrations by such an expert can be biased to the scenario generated. Making changes to the underlying optimization model can be helpful to improve decisions. As learning iterations progress, modifications can be made to the

underlying optimization models to enhance their effectiveness. A parallel can be drawn to reinforcement learning, where an initial, naive policy generates actions and is iteratively updated based on rewards collected. This continuous refinement enables the expert to provide increasingly improved demonstrations, ultimately leading to more robust decision-making.

[Pham et al., 2023], [Larsen et al., 2018], [Mandi et al., 2020], [Baty et al., 2024], [Greif et al., 2024], and [Rautenstrauß and Schiffer, 2025] all rely on the same static expert for demonstrations, where the expert provides consistent guidance based on predefined knowledge or decisions. In contrast, algorithms in reinforcement learning (RL), such as those in [Kullman et al., 2022] and [Zhang et al., 2020], can be viewed as adaptive experts. These RL algorithms improve over time by learning from the consequences of their decisions, continuously refining their strategies based on feedback from the environment, rather than relying on fixed demonstrations. One could classify such experts as heuristic experts who adaptively learn as they collect data.

5.2 Learning from Imperfect Experts

Imitation learning has found extensive applications in various problems, largely due to recent advancements in interactive algorithms that tackle covariate shifts and compounding errors associated with traditional methods such as behaviour cloning. IL methods generally have the following three key assumptions:

1. Expert demonstration is perfect/optimal.
2. Expert provides a single and identical target action for a given state.
3. All demonstrations are from a single expert.

In Section 5.1, we introduced five distinct types of algorithmic experts used in the learning process. Their complexity follows the order: Myopic $<$ Deterministic $<$ Full-Information $<$ Two-Stage Stochastic $<$ Multistage Stochastic. The quality of these experts depends on the amount of information they utilize. Among them, only the Multistage Stochastic expert provides truly optimal demonstrations. However, its computational intractability makes it impractical for complex real-world problems. The remaining experts, while more feasible, are imperfect and produce suboptimal decisions of varying quality. Selecting an expert for learning thus requires balancing model complexity with decision quality. This section explores the implications of learning from imperfect experts. We also discuss some articles in the literature that use faulty experts.

5.2.1 Imperfect Expert and its Effect on Learning

Consider a myopic expert who ignores future uncertainties and makes decisions purely based on immediate, short-term gains. In the NTP, where there is a high penalty for leaving shifts unfilled, a myopic expert would likely choose the greedy action of notifying all available employees at once to minimize the risk of vacancies. However, this strategy can lead to significant downstream costs—in this case, a high "bump" cost. If we employ a Full-Information expert, the decisions are influenced by access to perfect future knowledge. This approach can result in decisions that appear optimal when future events are fully known but may not translate well to the realistic, stochastic context of the problem. For instance, imagine a scenario in the NTP involving 5 shifts, 10 employees, and 30 epochs within a planning period. In an unlikely scenario where the first five employees all have a response delay of zero, a Full-Information expert might determine that the optimal action is to notify all employees at once, as it immediately resolves all shift assignments. Alternatively, a strategy that calls a single employee at each epoch—spreading notifications over time—could also be considered optimal. In fact, any policy that ensures at least five employees are contacted before the planning horizon ends would qualify as optimal under perfect information. This results in multiple target actions for the same initial state when guided by a Full-Information expert.

However, it is critical to recognize that actions deemed optimal under Full-Information may carry significant costs in a more realistic, stochastic environment. For example, notifying the first 5 employees simultaneously might be optimal for one certain scenario, but it would be costly in most other scenarios. A stochastic expert, aware of the unpredictability inherent to the problem, would not advocate for such a strategy, instead opting for more cautious actions that hedge against uncertain future responses. Thus, if our dataset includes demonstrations from any of these imperfect experts, it may contain "faulty" demonstrations. These demonstrations, optimal under certain assumptions, would affect the performance of any learned model. This highlights the challenge of relying on imperfect experts: their demonstrations may not generalize well to the stochastic reality, and one may need to filter out such examples. Even after filtering out unsafe demonstrations and smoothing out outlier decisions in the dataset, a fundamental challenge remains: imperfect experts contribute guidance that, while well-intentioned, is ultimately suboptimal. Each expert attempts to optimize its decisions within its unique, albeit flawed, approach. This imperfection means that the learned model will naturally inherit any suboptimal tendencies present in the training data, replicating the limitations of its teachers. This raises a crucial question: can a model improve beyond the suboptimal decisions demonstrated by its experts? Addressing this requires mechanisms that not only learn from the experts but also refine and enhance the decisions made over time.

5.2.2 Mitigating Faulty Demonstrations

In scenarios with inherent uncertainty, randomized policies often perform well; common examples include Multi-Armed Bandits and Inventory Control problems. Thus, given that some demonstrations may have multiple conflicting target actions from the same expert, one can even try to learn a distribution over the set of possible actions. In these cases, the learning model must go beyond a single deterministic outcome and instead learn a distribution over possible actions that are appropriate for a given state. This shift requires the IL model to generalize across varying demonstrations while accurately capturing the underlying decision-making distribution.

Given the list of these experts, one can employ a strategic combination to train the novice model, depending on the specific context or requirements. This approach mirrors real-life scenarios where multiple experts, each with their own strengths, are utilized to teach or provide demonstrations. A practical analogy can be found in educational settings, where different teachers employ unique teaching styles, or in driving contexts, where individual drivers follow distinct driving policies. Each expert contributes their perspective, enriching the learning experience with a diverse set of strategies. In our specific case, we can adopt a hybrid approach, leveraging the strengths of various experts based on the situation at hand. For instance, the computationally expensive stochastic expert, who considers uncertainties and future scenarios, can be utilized to label states that are deemed risky or unfamiliar. These are scenarios where high-quality guidance is critical, as errors can have significant consequences. Conversely, for safer or more predictable states, we can rely on simpler, less costly experts to provide the necessary labels. This adaptive labelling strategy not only optimizes computational resources but also balances the trade-off between complexity and decision quality. By prioritizing the use of more sophisticated experts only when it is most beneficial, the learning process becomes more efficient, achieving robust performance without unnecessary overhead.

5.2.3 Literature Review on Learning from Imperfect Experts

In the literature, significant effort has been directed toward learning from imperfect experts in imitation learning (IL). For instance, [Wu et al., 2019] introduces a novel approach that utilizes confidence scores to capture the quality of demonstrations, where these scores indicate the likelihood that a given trajectory is optimal. In this framework, target actions can be labelled as optimal, non-optimal, or left unlabeled. They propose the 2IWIL algorithm, which first trains a semi-supervised classifier to predict confidence scores for unlabeled demonstrations. These scores are then integrated into a weighted Generative Adversarial Imitation

Learning (GAIL) framework to train the agent.

In another approach, [Wang et al., 2021] explores methods for weighting imperfect expert demonstrations in GAIL without relying on extensive prior information. [Brown et al., 2020] addresses the challenge of deriving a reward function that effectively fits a set of ranked trajectories, providing a way to learn from relative performance. To address noisy or mixed-quality demonstrations, [Valko et al., 2013] leverages semi-supervised support vector machines to differentiate between good and bad demonstrations when training a policy, thereby enhancing the performance of the agent. More recently, [Wang et al., 2023] proposes a two-step process for dealing with imperfect demonstrations: first, purifying noisy demonstrations through a diffusion process and then performing imitation learning on these refined examples. These existing approaches focus on learning from imperfect demonstrations provided by a single expert, often relying on inverse reinforcement learning. Further research is needed to expand these techniques to scenarios involving multiple imperfect experts, where each expert may contribute unique biases or incomplete information.

In [Raykar et al., 2009] and [Raykar et al., 2010], authors explore imitation learning in settings with multiple experts, working under the assumption that a pre-labelled dataset is available. [Sun et al., 2023] introduces MEGA-DAgger, a variant of DAgger designed for interactive learning with multiple imperfect experts. To ensure high-quality training data, MEGA-DAgger begins by filtering out unsafe demonstrations, reducing the influence of any imperfect or suboptimal examples and thereby focusing the training of the novice on accurate, relevant data. Following this, the method incorporates a scenario-specific evaluation of the experts, using tailored metrics to assess their performance and resolve any conflicts in the guidance provided. This process helps ensure that the novice model benefits from the strengths of each expert while mitigating the risk of learning from flawed demonstrations. MEGA-DAgger's approach of combining filtering and selective expert evaluation supports a more robust and effective imitation learning process in environments with multiple, diverse expert sources.

[Brantley et al., 2020] considers an active learning setting with access to a cheaper but noisy expert (or heuristic) that provides guidance with some level of inaccuracy, whereas accessing the true expert is costly. They propose an algorithm called LEAQI (Learning to Query for Imitation), which aims to reduce the number of expensive true expert queries needed for target actions. LEAQI operates by always consulting the noisy expert for a label and only querying the true expert when it predicts a likely disagreement between the true expert and the noisy expert. This prediction is achieved by training a difference classifier that detects potential disagreements between the two sources. The challenge in training this classifier lies

in the one-sided feedback: if the classifier predicts agreement (and thus does not query the true expert), it cannot learn from any potential error in its prediction. [Laskey et al., 2016] is another example that aims to reduce the queries.

Recently, [Sekhari et al., 2024] studied a similar problem to [Brantley et al., 2020] with the goal of designing algorithms that achieve low regret for correct action predictions while minimizing the number of true expert queries in the presence of a noisy expert. Their algorithm's query count is bounded by the confidence level of the noisy expert's predictions. They further extend their approach to a multi-expert setting, where the learner has access to multiple experts, each specializing in different regions of the state space. [Xu et al., 2022], [Li et al., 2023] show the idea of using a supplementary dataset obtained from a behavior policy and propose a cooperation framework to boost learning in a Behavior Cloning framework.

5.3 Conclusion

We presented a comprehensive taxonomy of expert-based algorithms that can be utilized in imitation learning to address operations research (OR) problems. This taxonomy outlines various properties of the expert, enabling readers to better understand and effectively incorporate the expert in any imitation learning setting. Recognizing that obtaining a perfect expert for OR problems is often expensive or impractical, we highlighted key distinctions between traditional imitation learning problems and those encountered in OR.

CHAPTER 6 AN IMITATION LEARNING APPROACH FOR THE NTP

This chapter represents an initial attempt to develop machine learning-based policies for the stochastic version of the NTP. The primary goal is to train a model that can directly predict the next action at each decision point in the process. To achieve this, we utilize Imitation Learning (IL), specifically Behavior Cloning (BC), where the model learns from offline, fully informed solutions. The problem assumptions remain consistent with those outlined in Section 4.1.

Section 6.1 introduces the methodology employed in this study, detailing the structure of the machine learning model and the process of training it on offline data. The proposed algorithm builds upon the DAgger framework [Ross et al., 2011], with adaptations to address the challenge of obtaining exact optimal actions for imitation. Instead, we utilize an approximation based on the optimization models described in Chapter 5. We also give a perspective of the algorithm with respect to the taxonomy defined. Subsequently, Section 6.2 presents the experimental results, evaluating the trained model's performance and providing key insights derived from these findings.

6.1 DAgger with Algorithmic Experts

The DAgger algorithm [Ross et al., 2011], as defined in Section 2.2.3, is an iterative imitation learning method designed to enhance policy performance by incorporating expert corrections for states encountered by the learned policy. Unlike standard behavioral cloning, which suffers from covariate shift, DAgger progressively expands the training set by including states visited by the learner, thereby improving generalization. Variants of DAgger have been successfully applied in various domains, including autonomous driving [Kelly et al., 2019], video games [Ross et al., 2011] and drone control [Wang and Chang, 2021].

However, DAgger relies on the availability of an optimal expert capable of providing demonstrations across all encountered states during training. In practice, this assumption is often impractical for real-world operations research (OR) problems due to computational intractability or the lack of a well-defined optimal policy. Many OR problems, particularly those involving large-scale combinatorial decision-making, are $\mathcal{NP}$-hard, making it infeasible to generate expert demonstrations for all possible scenarios.

To overcome this challenge, our approach relaxes the requirement of a true optimal expert and instead utilizes approximate or imperfect experts (Section 5.1.1) to guide the learning pro-

cess. These experts are derived from heuristic methods, rule-based policies, or optimization models with relaxed constraints. The DAgger-based algorithm we develop relies on multiple optimization-based experts to inform decision-making. While these approximations provide valuable guidance, they may also introduce inconsistencies or errors in policy execution.

Since some experts may produce differing optimal actions for the same state, inconsistencies arise that must be addressed. As discussed in Section 5.2, there are several reasons why this discrepancy can occur. To mitigate the impact of such expert imperfections, we implement an aggregation strategy that consolidates multiple decisions from the same expert into a single target action. This approach helps smooth out individual inaccuracies by leveraging diverse expert perspectives, reducing the risk of systematic bias introduced by any single imperfect expert. By structuring the aggregation process effectively, our approach enhances the robustness of the learned policy while maintaining computational efficiency.

6.1.1 Constructing training examples and training policies - DAgger

The DAgger algorithm follows an iterative policy training framework, leveraging online learning principles that require access to an expert. In each iteration, the primary classifier is retrained using all previously encountered states from the learner's experience. This iterative refinement allows the learner to correct past mistakes at each stage. Initially, the first policy, π_0, is either directly derived from expert demonstrations or serves as a strong heuristic. After executing π_0, the learner observes the states it encounters. A new dataset is then constructed, incorporating corrections from the expert to address errors in π_0. The next policy, π_i, is trained on a combination of the original trajectories and the newly collected ones from the current iteration. This process is repeated iteratively to refine the policy over time.

To enhance stability, we redefine the DAgger algorithm using the VANILLA DAgger decision rule, which mitigates the risks associated with strictly following the learned model's actions. Instead of immediately adopting the model's suggested actions, VANILLA DAgger retains expert intervention with a certain probability. As training progresses, the learned policy gradually assumes greater responsibility for decision-making, ensuring a controlled transition from expert-guided behavior to autonomous decision-making. This approach reduces the likelihood of the model entering undesirable or unsafe states early in training, ultimately fostering a more stable learning process.

Furthermore, we extend the original DAgger framework by allowing flexibility in selecting different types of experts. Algorithm 12 presents a more detailed version of Algorithm 3 from Chapter 2, explicitly incorporating an expert based on mathematical optimization, as

shown in line 7. Since solving exact optimization models may be computationally intractable, these models may instead serve as approximations. In this context, we denote the expert's action as $\hat{a}$. Throughout the algorithm, indices i and k represent the current iteration and epoch, respectively. Additionally, we assume direct access to the true state x_k rather than the observation o_k, as discussed in Chapter 2.

Algorithm 12: DAgger - DAgger algorithm with Optimization-based Expert using VANILLA DR

1 **Function** `DAgger()`:
2 Initialize $D \leftarrow \emptyset, \pi_0, \lambda, \beta_0$
3 **for** $i \in \{0, \ldots, itr\}$ **do**
4 $\beta_i \leftarrow \lambda^i \beta_0$;
5 Sample K-step trajectories as follows;
6 **for** $k \in \{1, \ldots, K\}$ **do**
7 $\hat{a}_k^i = \text{ExpertAction}(x_k)$;
8 $a_k = \text{VANILLADAgger}(x_k, \hat{a}_k^i, \beta_i, \pi_i)$;
9 $x_{k+1} = X^M(x_k, a_k, \xi_k)$
10 **end**
11 Collect $D_i = \{(x_k, \hat{a}_k^i)\}$ dataset of visited states and target actions;
12 $D \leftarrow D \cup D_i$;
13 Train policy π_{i+1} on D;
14 **end**
15 **end**

Algorithm 13: VANILLA DAgger Decision Rule

1 **Function** `VanillaDAgger`$(x_k, i, \beta_0, \lambda)$:
2 $a_k \leftarrow \pi_i(x_k)$;
3 $a_k^* \leftarrow \pi^a(x_k)$;
4 $\beta_i \leftarrow \lambda^i \beta_0$;
5 $z \sim \text{Uniform}(0, 1)$;
6 **if** $z \leq \beta_i$ **then**
7 **return** a_k^*;
8 **else**
9 **return** a_k;
10 **end**
11 **end**

6.1.2 Experts Used

Given this general DAgger framework, we can now define different expert functions based on the taxonomy defined in Section 5.1.1. We restrict ourselves to the Full-Information (FIE),

Deterministic Expert (DE), Two-stage stochastic (2SSE) and Aggregated Deterministic Expert (ADE). This function will take as input the current state and return an action. Within the function, they will generate the required number of scenarios and then use them to solve an optimization problem.

1. **Full-Information Expert** - This expert has knowledge of the future scenarios $\tilde{\boldsymbol{\xi}}$ with respect to the current state x_k. Alternatively, this means we use the same scenario to generate a trajectory and the target action. We then solve a deterministic optimization problem Equation (5.3) to obtain the optimal action a^o.

2. **Deterministic Expert** - This expert samples one future scenario $\hat{\boldsymbol{\xi}}$ with respect to the current state x_k. We then solve a deterministic optimization problem defined in Equation (5.4) to obtain the optimal action a^p.

3. **Aggregated Deterministic Expert** - First, n samples of future scenarios are generated, and we obtain optimal actions denoted by $\bar{a}_n^p$ for each of these scenarios using Equation (5.4). Then, a target action is generated by aggregating all actions given by $\bar{a}^p = Agg(\bar{a}_n^p)$ using an aggregator function Agg.

4. **Two-stage Stochastic Expert** - Similar to the Aggregated Deterministic Expert, we generate n future scenarios $\tilde{\boldsymbol{\xi}}_l$. However, now we solve a single two-stage stochastic program defined by Equation (5.5). The smaller the number of scenarios, the easier it is for the underlying stochastic program to solve. This expert considers the aggregation within the optimization model as opposed to the Aggregated Deterministic Expert.

We use the same Vanilla Dagger decision rule defined by Algorithm 13. This function will take as input the current state x_k, the expert-provided action $\bar{a}_k$, the mixing parameter β_i, and the learned model π_i. It returns the action to be taken to proceed in the episode. The main difference with respect to Algorithm 5 is that the action is suggested by an imperfect expert denoted by π^a.

However, given that the list of experts discussed may be imperfect or suboptimal, each expert is implemented individually and evaluated based on their performance. This evaluation can be conducted using various metrics, such as cumulative cost or the ability to avoid undesirable states. By analyzing their effectiveness over multiple trials, the expert who demonstrates the best overall performance will be selected.

Finally, we would like to describe the DAgger algorithm with respect to the taxonomy defined in the Chapter 5 for NTP. Table 6.1, Table 6.2 provide a structured comparison of different expert configurations used in the DAgger algorithm. The DAgger algorithm is designed to

work with a variety of expert types, making it adaptable to different learning settings. The goal is to learn a dynamic policy that directly predicts decisions, leveraging expert-provided labels to guide the learning process. Each column represents a specific expert type, while the rows outline key characteristics that differentiate them within the DAgger algorithm.

- **Expert Type**: We focus on four specific types: FIE, DE, ADE, and 2SSE.
- **Policy Type & Policy Variable**: All policies learnt are dynamic, meaning they take into account the current state of the system. The decision variable is always a direct decision that is to be executed in the episode.
- **Optimality**: The primary distinction between the experts lies in the optimality of their actions. FIE, DE, and 2SSE are suboptimal in the sense that they generate optimal decisions only for the specific scenarios they encounter, rather than achieving true global optimality across all possible situations. ADE, in contrast, aggregates optimal solutions from multiple deterministic experts, effectively providing an estimate of the optimal action. This aggregation process helps mitigate the limitations of individual deterministic experts by leveraging multiple perspectives.
- **Information Used**: All four expert types operate with limited information, as their decisions are based on a finite set of randomly generated scenarios rather than exhaustive knowledge of the problem space.
- **Reproducibility**: Since these experts rely on sampled scenarios, their recommendations can be inconsistent; that is, the recommended action may change for the same state.

Table 6.1 Vanilla DAgger Algorithm Classification

	Expert	
	Full-Information	**Deterministic**
Policy Type	Dynamic	Dynamic
Policy Variable	Direct Decision	Direct Decision
Optimality	Sub-Optimal	Sub-Optimal
Information Used	Limited	Limited
Reproducibility	Inconsistent (Imperfect Expert)	Inconsistent (Imperfect Expert)
Expert Usage	Single Complete	Single Complete
Number of Experts	Single	Single
Learning Control	Regressive	Regressive
Expert Evolution	Static	Static

Table 6.2 Vanilla DAgger Algorithm Classification

	Expert	
	Two - Stage Stochastic	**Aggregated Deterministic**
Policy Type	Dynamic	Dynamic
Policy Variable	Direct Decision	Direct Decision
Optimality	Sub-Optimal	Approximate
Information Used	Limited	Limited
Reproducibility	Inconsistent (Imperfect Expert)	Inconsistent (Imperfect Expert)
Expert Usage	Single Complete	Mulitple Complete
Number of Experts	Single	Single
Learning Control	Regressive	Regressive
Expert Evolution	Static	Static

- **Expert Usage**: ADE differs in how it utilizes expert guidance. Instead of relying on a single deterministic expert, it acts as a proxy expert by querying the DE multiple times and selecting a target action based on an aggregation criterion. This makes ADE more robust to variations in deterministic expert decisions and helps smooth out inconsistencies. The FIE, DE, and 2SSE are all used just once to provide a demonstration.
- **Number of Experts**: All algorithms use a single distinct expert.
- **Learning Control**: All experts follow a regressive learning approach, meaning they influence trajectory generation iteratively under the Vanilla DAgger framework.

6.1.3 Using the ML model

To evaluate the effectiveness of our approach, we apply it to the Notification Timing Problem (NTP), a real-world challenge that involves determining the optimal timing for notifying employees about available shifts. In typical online NTP scenarios, employees are usually notified in batches, with a waiting period between notifications to allow for responses. This sequential notification strategy helps prevent unnecessary notifications and mitigates the risk of overwhelming employees with excessive alerts. However, while this approach reduces the chance of over-notification, it often results in delays, as employees may take longer to respond, and subsequent notifications are delayed accordingly.

The overall efficiency of the notification process can be improved by proactively sending additional notifications based on predicted response patterns, thereby reducing total wait times and ensuring faster shift assignments. By anticipating when employees are likely to respond, the system can optimize the timing of notifications, facilitating quicker decisions and a smoother operational flow.

While anticipatory actions promise to improve decision-making efficiency, they also introduce significant challenges, particularly for human planners. Human decision-makers often face substantial uncertainty and variability in real-world settings, and predicting future responses accurately is difficult due to limited available information and the complexity of the problem. Analysing and modelling the myriad factors influencing employee behavior, such as response delay distributions and external influences on availability, may exceed the cognitive capacities of human planners. Furthermore, traditional methods for forecasting and scheduling often fail to account for the dynamic nature of the problem, where conditions change rapidly as employees are notified and respond.

These complexities underscore the need for machine learning assistance in managing the NTP effectively. Our approach utilizes machine learning techniques to train a model based on historical experiences and expert demonstrations. By learning from past data, the model can recognise patterns in employee responses and optimize the timing of notifications. Once the model is trained, it can be seamlessly integrated into real-time decision-making, offering a structured and efficient solution for managing the notification process in online NTP scenarios. The ability to make data-driven, anticipatory decisions allows for improved resource allocation, reduced wait times, and overall operational efficiency.

We next describe how the simulation setting operates given a learned policy π. In the simulation setup, at time 0, the initial notification is sent, and this notification for the employee is placed in a buffer. The learned model is then invoked to determine the timing for subsequent notifications using the current state of the system. The current state of the system is then condensed into a set of features as the size of the state at each epoch is large. For each decision epoch within the planning horizon, a training example is constructed, represented as $(F_k, \hat{a}_k)$, where F_k signifies the input features and $\hat{a}_k$ denotes the label indicating the target expert action at epoch k. The learned model takes as input the features of the current state and estimates a wait time for the next employee. If the model estimates a wait time of zero, the next notification is immediately placed in the buffer. This process continues until either the buffer reaches its limit of W notifications per minute or the model suggests a positive wait time before notifying the next employee. If W notifications were sent in the previous minute, the model is re-invoked to assess the timing of the next action. When a positive wait time is estimated, the simulation waits until this time elapses, after which the next notification is placed directly in the buffer. Additionally, the model is invoked whenever an employee responds, allowing the decision-making to adjust based on recent actions. Thus, the decision epochs—where the model is invoked—can be triggered by three possible events: (i) reaching the buffer limit in the previous time interval, (ii) the expiration of a wait time, and (iii) an employee response. This setup enables the model to adapt decisions dynamically

based on real-time feedback, simulating a responsive and realistic operational environment.

We condense the state into corresponding features that are given as input to the ML model. The features utilized in our models are detailed in Table 6.3. While most features are self-explanatory, additional detail is provided for some. The *cumulative leftover cutoff* represents the sum of the remaining cutoff time for all employees. Next, to obtain the *delay buckets*, we first divide the interval $[0, D]$ into six intervals to define delay buckets. For example, when $D = 180$, intervals such as $[(0, 10), (10, 30), (30, 60), (60, 120), (120, 150), (150, 180)]$ are used to create six buckets. For any interval (l, u), the feature captures the count of employees whose current delay usage falls between l and u.

ONP cumulative notifications represent the fixed number of employees that the ONP policy would have notified until the current epoch. Meanwhile, the *to notify now* feature represents the number of employees waiting to be notified at the current epoch. This scenario occurs when the model suggests notifying an employee and immediately receiving the decision for the next employee in the seniority order.

Features: F	
time remaining (tr)	time since last notification (tsln)
shifts vacant (sv)	cumulative employees notified (cn)
ratio of time remaining and shifts vacant (rtbysv)	cumulative leftover cutoff (cl)
action at last epoch (la)	delay buckets (db_j)
ONP cumulative notifications (cn_onp)	number of response at last epoch (lr)
to notify now (tc)	

Table 6.3 Set of features used to make predictions

For the learning part, we employ a neural network. We use 3 layers for the neural network with 128, 64, and 16, respectively. The target action is the time to wait before sending the next notification, and hence, we still use mean square error as the loss function. The neural network is trained for 100 epochs with a batch size of 32. All features were normalised before being input into the NN.

6.2 Results

All experiments and models were implemented in Python, and GUROBI 10.0 was employed to solve the offline optimization problem instances. A maximum solving time of 4 minutes was set for each instance. GUROBI consistently demonstrated efficient performance, often finding the optimal solution within seconds for all instances. In rare cases, it required additional time to confirm optimality.

6.2.1 Experimental Settings

We consider the NTP with the same settings of $N = 150$ employees, $H = 6$ hours, with a discretization of 1 minute, a total of $L = 50$ shifts, and a cutoff $D = 3$ hours. For each iteration i of the DAgger algorithm, we perform 10 runs with the learned policy π_i. A total of 50 iterations are performed. Additionally, in each iteration, we perform 10 runs using the policy π_i. This additional loop is to collect sufficient demonstrations for the current policy. The policy is then updated using data from all 10 runs. Thus, in all, there are a total of 500 trajectories to collect the data. We set $G = 200$ and $W = 2$.

We present the results for the DAgger algorithms, Vanilla Decision Rule - Algorithm 5 for each of the experts. We set $\lambda = 0.7$, $\beta_0 = 1$ for both the decision rules. Furthermore, we restrict ourselves to *median* as the aggregator function. The Full-Information (FIE), Deterministic Expert (DE), and Aggregated Deterministic Expert (ADE) both use MIP_{NTP2} (4.15-4.24) formulation. The Two-Stage Stochastic Expert (2SSE) uses MIP_{DNTP-S} (4.2-4.14) as models to get the optimal action. A time limit is set for obtaining the solution from the expert. For the FIE, DE, and ADE, this limit is 60 seconds. All optimization problems for the single-scenario experts are solved to optimality within predefined limits. For the 2SSE, we evaluate two configurations—one with a 90-second time limit and another with a 180-second time limit. Depending on the current time period within the planning horizon, the two-stage stochastic model may not always reach optimality, especially in the earlier periods when the problem size is larger. We provide additional analysis using 9 scenarios to examine this aspect in detail.

These experiments aim to provide a comprehensive evaluation of the DAgger algorithms under varying levels of information and uncertainty, thus demonstrating the robustness and adaptability of the algorithms to different expert paradigms. We refer the reader to Appendix A.3, which presents the initial implementation of the DAgger algorithm using the Naive decision rule. In this section, we provide a detailed comparison between the Deterministic Expert (DE) and the Aggregated Deterministic Expert (ADE), including a description of the algorithm. We also explore different aggregator functions for the ADE expert. In the ADE approach, actions are aggregated across 9 distinct scenarios, allowing it to capture a richer set of optimal actions than any single scenario can provide. The choice of nine scenarios is motivated by initial experimental results to balance computational time and quality of solution. Our findings indicate that the ADE expert outperforms the DE when using the median as the aggregation function. However, ADE does not significantly outperform the ONP baseline introduced in Chapter 4.

6.2.2 Data and Scenarios

The real-world response delay data was divided into train and test sets. The train set was utilised to select the best-performing models, and then the test set was used to assess performance. We compare our results with ONP from Chapter 4. The ONP is a heuristic policy based on the aggregation of pure offline solutions. This policy will send notifications if the current cumulative notifications do not meet a dynamic threshold obtained from offline solutions for each decision epoch. Note that this is a static policy and does not account for the current state of the system. The ADE is similar to the way the ONP policy is formulated, just that we do the learning to take into account the current state of the system using DAgger.

Figure 6.1 shows the notification profile of the ONP across time. After sending a few notifications at the start, the rate of sending notifications slows down and is then accelerated when the end of the planning period is close. The only difference is that we enforce a restriction of not notifying more than $W = 2$ employees per time unit. We do this after seeing the ONP notification patterns in Figure 6.1. This policy is known to produce good results from Chapter 4.

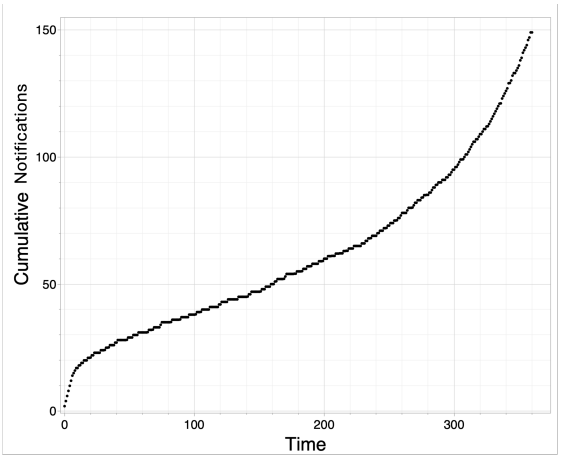


Figure 6.1 Notification profile

6.2.3 Comparing the time limit for the 2SSE

Solving the two-stage stochastic problem to optimality within the DAgger framework imposes a substantial computational burden. To mitigate this, we introduce three time limits—90 seconds, 180 seconds and 600 seconds, and evaluate their impact on solution quality. The 600-second test is just for reference as implementing it would be really costly even in offline training. Specifically, we generate a set of 200 instances, each comprising 9 scenarios, and solve the corresponding optimization problems at the start of the horizon, when the problem

is at its most complex. It is important to note that, in practice, the problem becomes easier at later stages of the horizon, as some employees will have already been contacted, reducing the decision space. Table 6.4 presents the results, illustrating the effect of time limits on the quality of the solutions obtained.

We observe that increasing the time limit to 180 seconds significantly improves solution quality compared to the 90-second limit. However, extending the limit further to 600 seconds yields only marginal gains in objective value, despite a notable reduction in the MIP gap. This suggests diminishing returns in terms of decision quality beyond a certain point. The results clearly indicate that a 90-second limit may often produce subpar expert decisions for imitation. Nevertheless, the improvement obtained with a 180-second limit comes at the cost of doubling the computational effort during DAgger training, underscoring the trade-off between expert quality and training efficiency. Figure 6.2 presents a box plot of the MIP gaps across 200 instances.

Table 6.4 Average Objective and Average Gap given the time limit

Time Limit (secs)	**Avg. Objective**	**Avg. Gap**
90	87.43	74.69
180	51.02	47.47
600	48.25	23.91

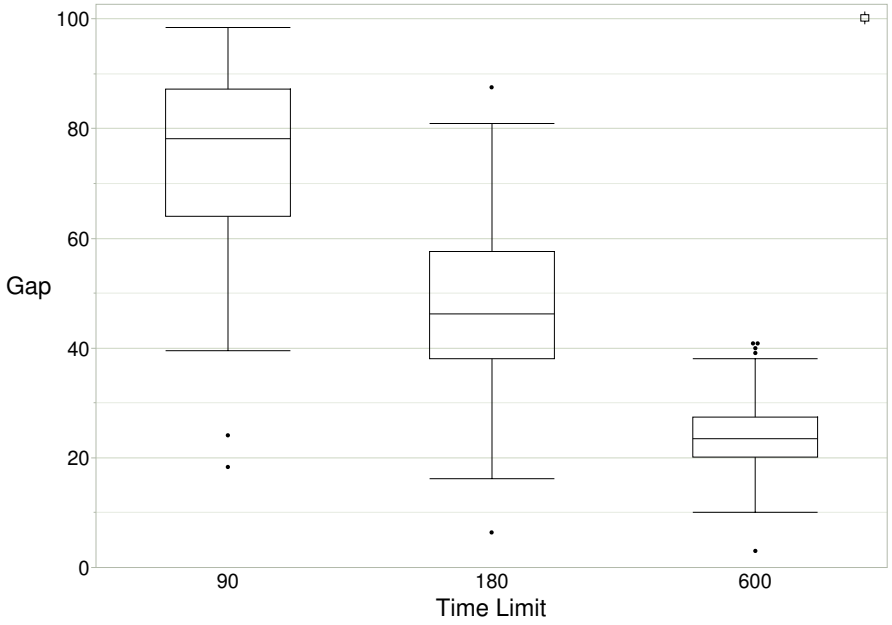


Figure 6.2 Box Plot of MIP gap

6.2.4 Training Time

Table 6.5 shows the approximate training time and the number of iterations with each expert and the time limit allowed to each expert.

Table 6.5 Training Time and Runtime given the time limit

Expert	Time Limit	Runtime (Hours)	Iterations
FIE	60	8	50
DE	60	8	50
ADE	60	98	50
2SSE	90	200	50
2SSE	180	200	28

We next present results for Vanilla DAgger applied to the NTP using the FIE, DE, 2SSE, and ADE as experts. We consider a set of 9 scenarios based on results obtained in Appendix A.3 both for the ADE and the 2SSE. We report the average cost of the learned policy over 50 iterations, with each policy evaluated through simulations on 1000 validation instances. Additionally, we analyse the average number of bumps and vacant shifts across iterations. Finally, we provide insights into the imperfections of these experts and try to understand their impact on policy performance.

6.2.5 Total Average Cost

Figure 6.3 illustrates the cost of each learned policy over 50 iterations using Vanilla DAgger. The 2SSE expert is referenced along with its corresponding time limit. Ideally, a well-performing policy should demonstrate a steadily declining cost over iterations. However, the results reveal that both DE and FIE exhibit highly erratic behavior, with large fluctuations in performance. As the learned model increasingly governs the trajectories, some iterations yield reasonable performance, followed by sharp spikes in cost. DE, which generates demonstrations based on a single sampled future scenario, is particularly unstable. In contrast, 2SSE90 begins with volatile performance but gradually stabilizes. While 2SSE180 offers greater stability from the outset, it struggles to show consistent improvement over iterations. Among all the experts, the ADE stands out, leveraging its aggregation over multiple deterministic scenarios to provide the most robust and reliable guidance for learning.

Figure 6.4 provides a zoomed-in view of Figure 6.3, offering a closer look at ADE's performance. The aggregation of multiple full-information optimal actions enables ADE to outperform other experts. As training progresses, the total average cost exhibits a clear downward trend, reinforcing the effectiveness of ADE's aggregation approach in achieving more reliable

and cost-efficient decisions. There is a slight jump in cost at the end, which may be due to a change in state distribution in the training dataset. Also, we need to investigate further the FIE and DE experts as to why they show oscillations.

The advantage of the DAgger algorithm is seen in Figure 6.3. The initial point of the plots represents the policy obtained by the first iteration of the DAgger. This policy purely learns from the expert trajectories. The ADE and 2SSE clearly show the effect of using DAgger as we get better policies from the one obtained in the first iteration. Figure 6.4 shows a general progression in obtaining better policies as the iterations progress. The other experts may not have good targets to imitate and hence show some erratic behaviour; however, there are stages when it performs better in some iterations than in other iterations.

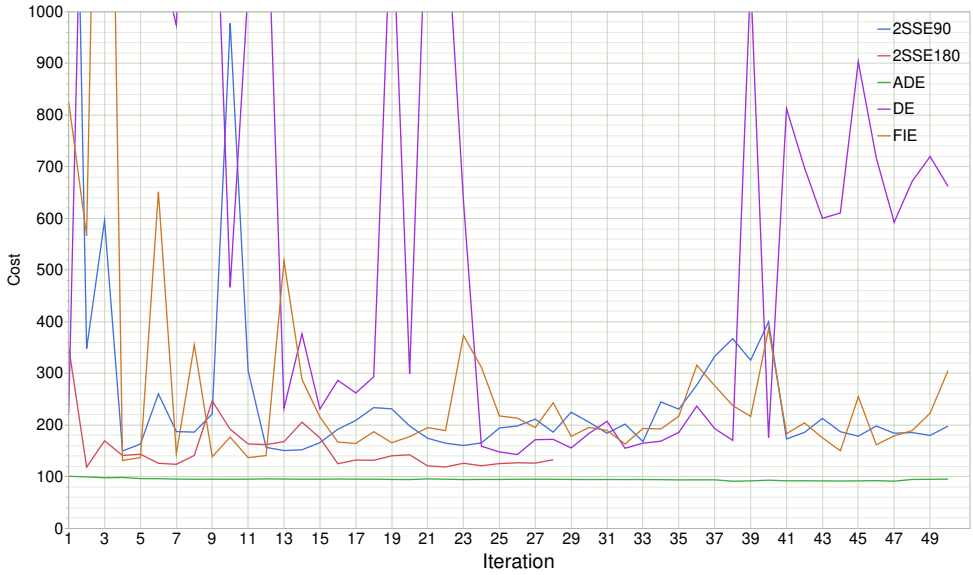


Figure 6.3 Cost progression across learning iterations

6.2.6 Total Average Bumps and Total Average Vacant Shifts

Figure 6.5 and Figure 6.6 provide additional insight by displaying each policy's average number of bumps and vacant shifts, respectively. From these plots, it becomes apparent that the primary contributor to the cost function in the DE and FIE is the number of vacant shifts. The learned models consistently fail to contact enough employees to ensure all shifts are filled, leading to a high number of vacant shifts that substantially impact the cost. This issue arises because these experts do not sufficiently account for the long-term impact of their decisions on shift filling. Even the 2SSE90, 2SSE180 exhibits higher vacancies than the management-set threshold of 0.15 vacant shifts, as specified in Chapter 4. We emphasize

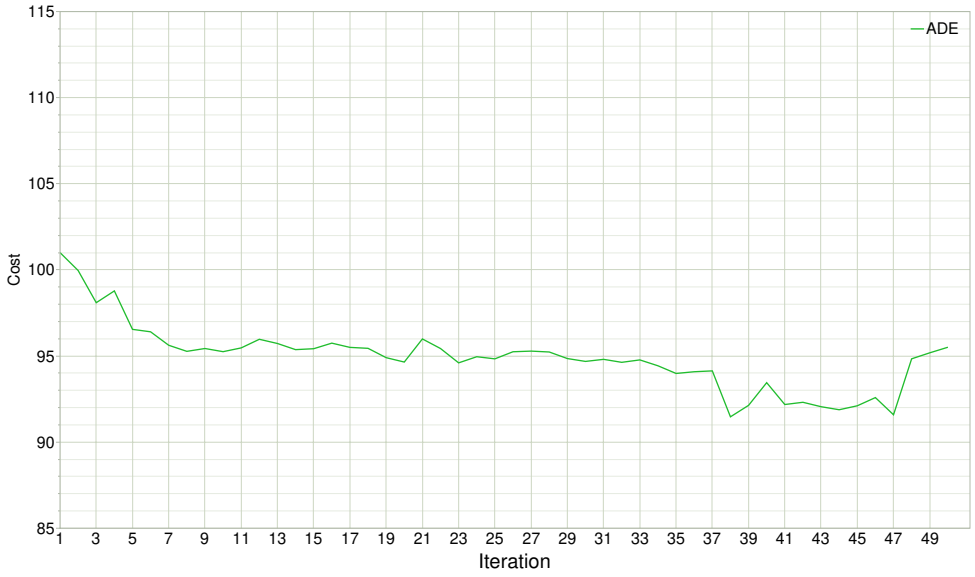


Figure 6.4 Cost progression across learning iterations - ADE

that for the NTP, vacant shifts are highly undesirable from a management perspective, as they can lead to significant operational disruptions. This highlights a key limitation of the stochastic expert in ensuring shift coverage even after considering multiple scenarios.

Figure 6.5 shows that the number of bumps remains relatively stable across iterations for all experts, suggesting that learned policies do not drastically alter the bumping dynamics. Interestingly, ADE exhibits the highest number of bumps, which, in turn, results in significantly fewer vacant shifts. This suggests that ADE prioritizes filling shifts at the cost of potentially increasing schedule disruptions. Figure 6.7 presents a zoomed-in plot of the total average vacant shifts for ADE. Initially, only minor improvements are observed in the average number of vacancies, but toward the final iterations, a noticeable drop occurs, leading to a significant reduction in cost. This final drop indicates that ADE can leverage its aggregated decision-making approach more effectively in later iterations, reinforcing the benefits of expert aggregation in guiding policy learning.

6.2.7 Comparing Best Policies

In this subsection, we present the results of the best-performing policies obtained by each method and compare them to the ONP heuristic introduced in Chapter 4. The selection of the best policies was based on their performance across 1000 validation instances, with the policy achieving the highest performance for each expert being chosen.

Given the insights from the plots discussed in the previous subsections, it is evident that the

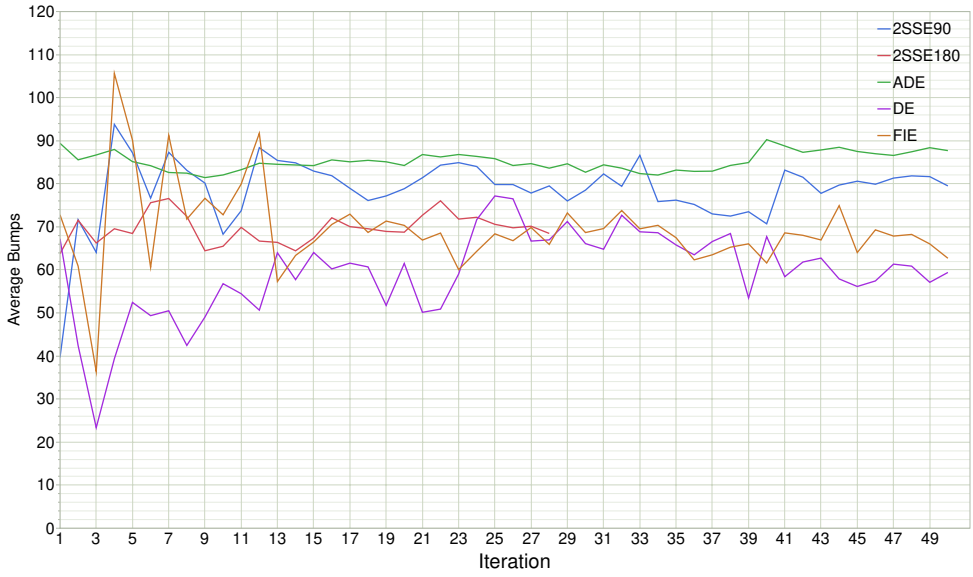


Figure 6.5 Total Average Bumps

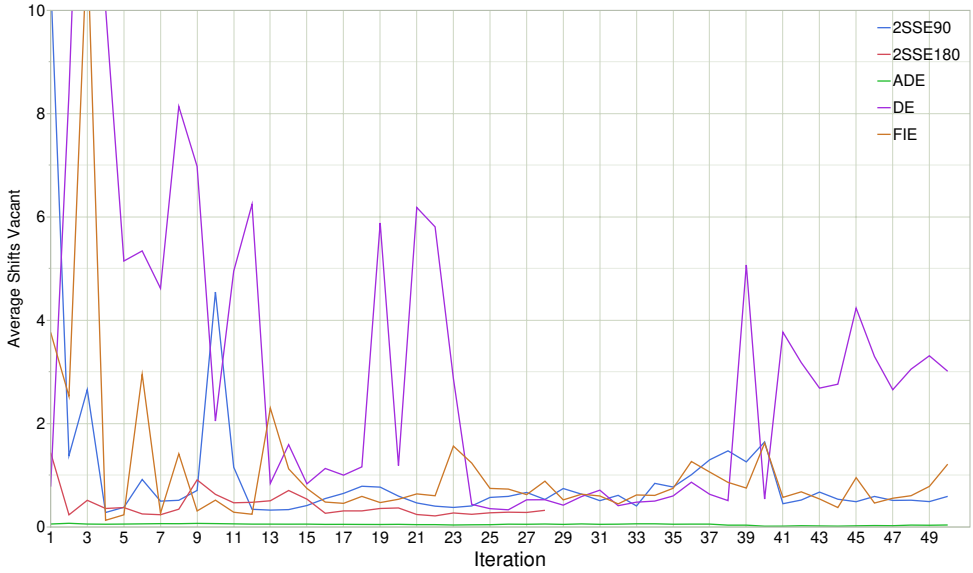


Figure 6.6 Total Average Vacant Shifts

DE, FIE and 2SSE are expected to underperform. After selecting the best policies based on validation results, these models were tested on a new set of 500 test instances, with their outcomes compared against the ONP heuristic. Table 6.6 summarizes the results, where the model number corresponds to the iteration of the DAgger algorithm in which the policy was obtained. Figure 6.8 represents the distribution of all employees having a response delay of less than 270 minutes across all 500 test instances. The threshold of 270, though less than H, is big enough, especially considering that many of the employees are notified late in the

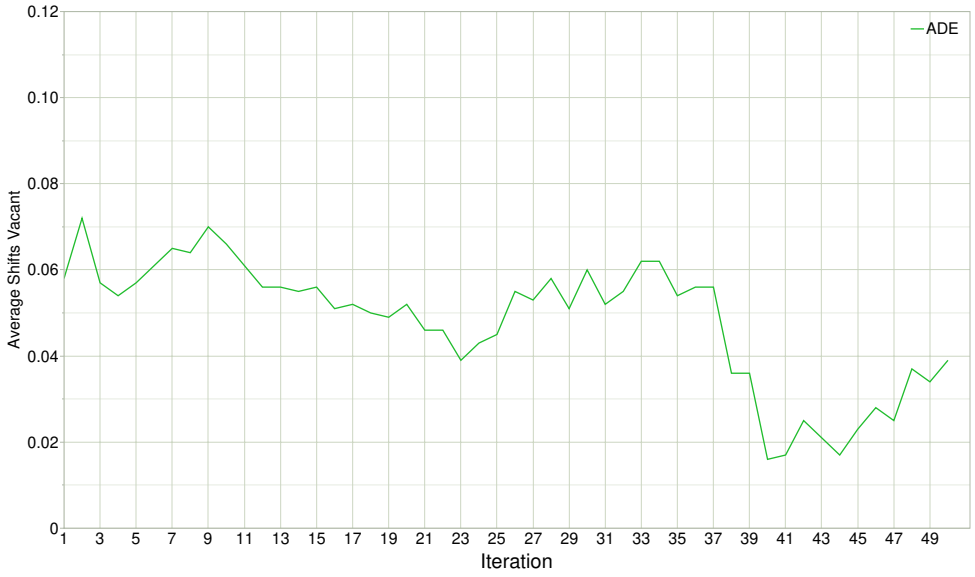


Figure 6.7 Total Average Vacant Shifts - ADE

Horizon. All such employees are referred to as effective available employees. Those instances with effective available employees close to 50 represent difficult instances which are prime candidates to have vacant shifts.

The findings demonstrate a significant improvement over the ONP when utilizing the ADE. The ADE effectively reduces both bumps and shift vacancies, showcasing its ability to achieve a better balance between these competing objectives. In contrast to ADE, DE, and 2SSE perform poorly, particularly in minimizing shift vacancies, underscoring their limitations in addressing the practical challenges of the problem. These models are not able to improve the heuristic solutions. Notably, the best models for the FIE, and the 2SSE were obtained in the earlier iterations of the DAgger algorithm. This means the addition of further data

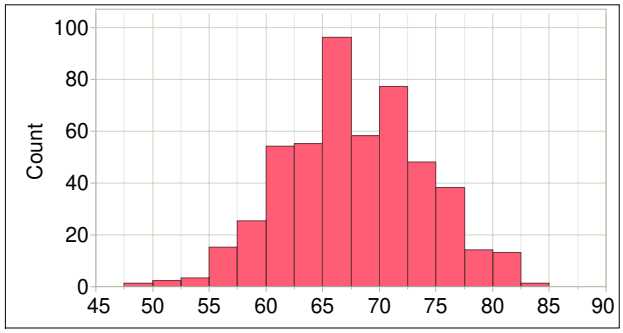


Figure 6.8 Distribution - Number of Users with response delays less than 270 minutes across all 500 test instances

did not lead to any improvement in the models. The results obtained are much better than the initial results given in the Appendix. The ADE is still, however, the best-performing expert. The reason for improved results can be attributed to the use of the VANILLA DAgger decision rule and a more tuned NN model. In the initial phases of the research, we only used Gradient-Boosted Decision Trees as the ML model.

Table 6.6 Final Model Results

Policy	**Model Number**	**Average Bumps**	**Average Shift Vacant**	**Average Cost**
ONP	-	87.22	0.064	100.02
Stoch	-	76.59	0.204	116.99
FIE	4	73.8	0.39	151.8
DE	25	66.25	0.148	155.05
ADE	47	**85.28**	**0.032**	**91.68**
2SSE90	4	90.93	0.32	154.93
2SSE180	2	69.92	0.274	124.72

We also perform a deeper dive into the results obtained by the best ADE policy and the ONP. Let π_{47}^{ADE} denote the best policy obtained using the ADE. There were a total of 16 vacant shifts across 500 test instances using π_{47}^{ADE} seen across only 7 instances. Likewise, the ONP gives a total of 32 vacant shifts seen across 12 instances. Table 6.7 gives details of all vacant shifts seen across different instances. We see that for all instances where there exist some vacant shifts using π_{47}^{ADE}, the ONP also had vacant shifts. In the final column, we list the effective employees available. This table highlights that π_{47}^{ADE} performs well even on riskier instances.

Table 6.7 Vacant shifts in ONP and ADE

	Vacant Shifts		
Instance ID	**ONP**	$\boldsymbol{\pi_{47}^{\text{ADE}}}$	**Effective Available Employees**
4	1	1	53
10	2	0	53
50	1	1	58
63	3	2	56
135	1	0	56
156	1	0	57
165	2	0	53
197	3	2	51
240	5	2	55
275	5	3	51
280	7	5	48
393	1	0	58

Figure 6.9 shows the feature importance using Shapley values. The y-axis denotes the feature names, with lower positions indicating greater influence on the output. The analysis reveals that features such as time remaining, number of vacant shifts, the ratio of time remaining to shift vacant, and the cumulative notifications sent are the most important features.

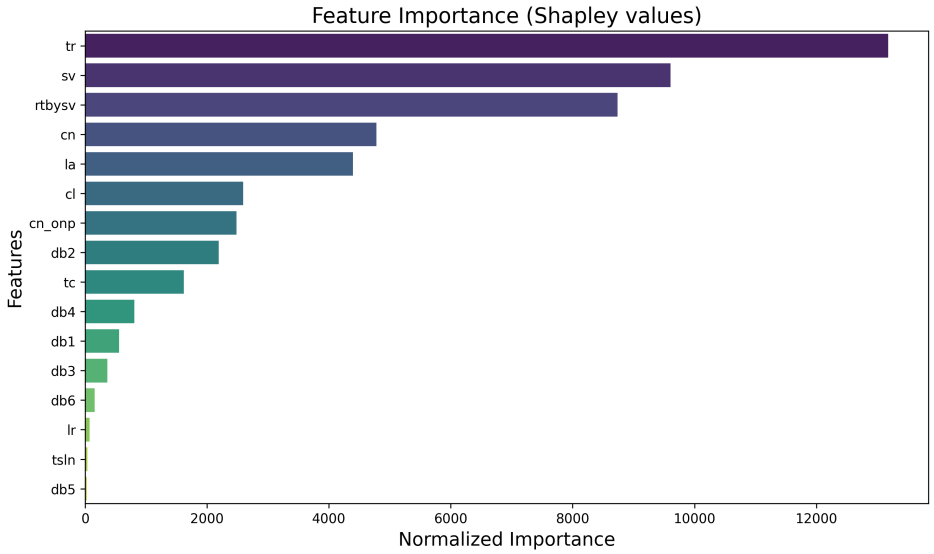


Figure 6.9 Feature Importance for π_{47}^{ADE}

Analysis of Expert Imperfection

The plots in Figure 6.10, Figure 6.11, and Figure 6.12 show the mean wait times recommended by different experts across all decision epochs. The dark blue line represents the mean, while the shaded region indicates the standard deviation. Since actual wait times are state-dependent, these plots primarily illustrate the smoothing effect achieved by the ADE.

The results show that FIE, ADE, and 2SSE90 exhibit highly variable mean wait times, with significant fluctuations. This variability suggests that the learning process lacks stable targets, leading to potential overfitting in the neural network. Furthermore, this explains why FIE, DE, and 2SSE90 initially performed well, as early-stage data collection has a lower variance, making it easier for the model to fit.

Among the experts, both 2SSE180 and ADE exhibit lower variability in mean wait times across epochs, suggesting more stable decision-making. However, ADE demonstrates slightly less variability than 2SSE180. The models learned from these experts recommend shorter wait times that contribute to a more proactive notification strategy, reducing shift vacancies. The smoother target provided by these experts facilitates more effective learning. This

highlights the benefits of aggregation techniques in capturing diverse future scenarios, leading to improved robustness and efficiency in decision-making.

Although an expert like the ADE demonstrates promising results, the plots suggest that further improvements can be made by refining the expert labels. Enhancing the quality of these labels could help the algorithm discover better policies in the earlier iterations. This process may involve identifying incorrect labels and subsequently relabeling them to ensure consistency and accuracy in the training data, thereby improving the overall performance of the learning algorithm.

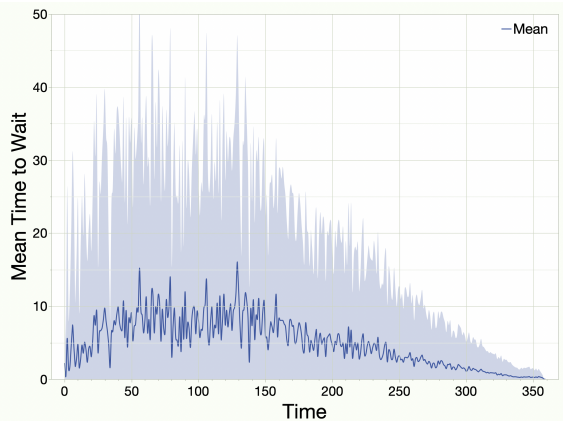
Given the plots, the experts can induce errors in two ways for the NTP. Either suggesting actions to notify too many employees too soon, or waiting too long to send a notification. These represent outlier actions due to the presence of outlier scenarios in the scenario set. We can introduce bounds on the suggested actions to filter out outlier actions.

Additionally, one could consider increasing the number of scenarios used in the 2SSE expert to better capture uncertainty. However, solving the resulting larger stochastic problems becomes increasingly expensive. A practical workaround is to rely on a known behavior policy to perform actions in situations where solving the full stochastic problem is computationally prohibitive. For example, we could use the ONP early in the horizon and then rely on the stochastic expert after a certain point. This allows the system to continue operating while reserving the use of the expensive 2SSE expert for more critical decision points.

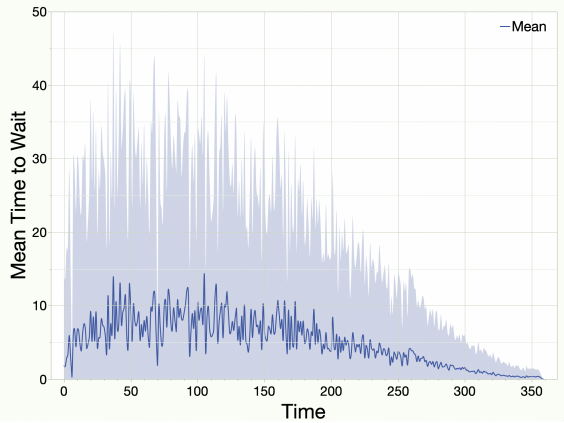
Moreover, the behavior policy can also serve as a sanity check or baseline to validate or filter the actions proposed by the expert, thereby adding a layer of robustness to the decision-making process. In Appendix Appendix A.4, we provide a detailed algorithm that is based on this idea by incorporating both expert and behavior policy in a complementary way.

6.2.8 VSS for the NTP

The **Value of the Stochastic Solution (VSS)** measures the benefit of explicitly considering uncertainty in decision-making compared to relying on a deterministic approximation. To compute the VSS, we first determine a solution assuming that all uncertain parameters—such as demand, travel time, or appointment duration—are fixed at their expected values. This formulation is referred to as the **Expected Value Solution (EV)**. We then evaluate the performance of this solution under real uncertainty and compare it to a solution optimized while considering uncertainty, referred to as the **Stochastic Solution (SS)**. The VSS is computed as the difference in objective values between the EV and SS solutions. A high VSS indicates that explicitly modelling uncertainty leads to significantly improved

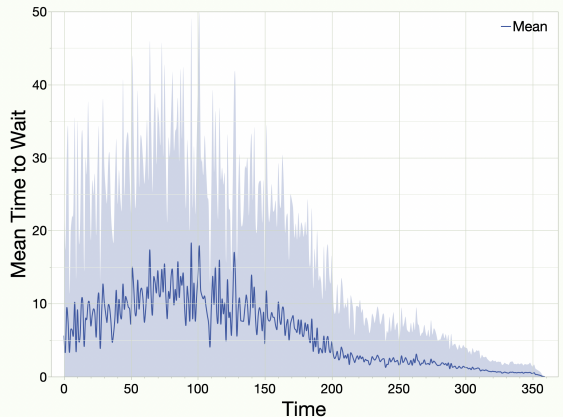


(a) FIE

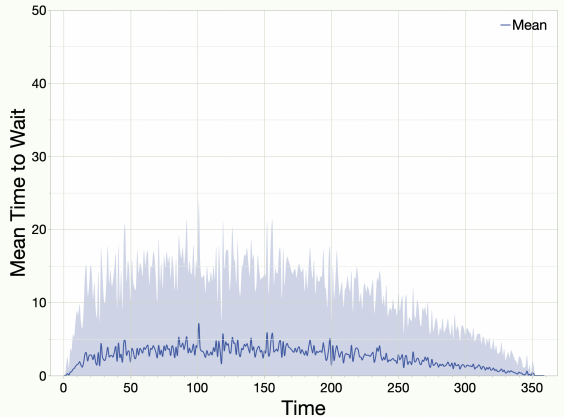


(b) DE

Figure 6.10 Mean time to wait across decision epoch



(a) 2SSE90



(b) ADE

Figure 6.11 Mean time to wait across decision epoch

decision-making, whereas a low VSS suggests that ignoring uncertainty has minimal impact.

In the literature, the VSS has been widely used to quantify the benefits of incorporating uncertainty into optimization models. [Florio et al., 2020] consider the Vehicle Routing Problem (VRP) with stochastic demands and report an average VSS of 4%, with a maximum of approximately 10%. Similarly, [Torres et al., 2022] studies the Vehicle Routing Problem with Time Windows (VRPTW) with stochastic vehicle availability, finding a maximum VSS of 17%. In the context of last-mile delivery using crowd-shipping, [Nieto-Isaza et al., 2022] reports a VSS ranging from 1.1% to 10% for networks with 10 nodes and 1.1% to 7.1% for networks with 20 nodes. The highest reported VSS among these studies appears in the work of [Berg et al., 2014] on patient appointment scheduling with uncertainty in procedure times

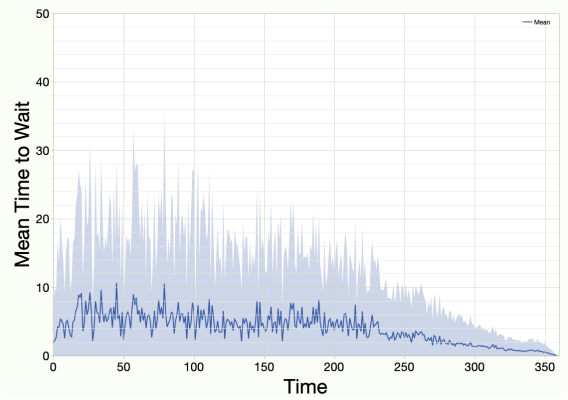


Figure 6.12 Mean time to wait across decision epoch - 2SSE180

and patient attendance, where the VSS reaches approximately 32%.

To compute the VSS for the Notification Timing Problem (NTP), we apply the following approach. In the EV formulation, we assume that all employee response delays are fixed at their mean values. Under this assumption, no "bumps" (schedule disruptions) would occur, as all employees could be notified simultaneously, ensuring responses arrive at the same epoch. However, such a policy would perform poorly in practice, as it fails to account for variability in response times. This deterministic policy corresponds to the NA policy, as shown in Table 4.7.

Another challenge in estimating the EV is that not all employees respond to system notifications. In practice, this will lead to high average response delays as some employees may never respond. The response delays for such cases are censored to a maximum value in the data. As a result, the most effective deterministic policy would be to notify all employees simultaneously as early as possible. We present the VSS considering this policy. Alternatively, one could consider using the mean response delay conditioned on only those employees who actually respond. However, even under this metric, the average delay remains high, as employees do tend to respond late in practice. As a result, this adjustment does not change the structure of the optimal policy.

Additionally, we will consider an alternate policy that is optimal for the EV problem. This policy sends notifications sequentially, starting from time zero, with each employee receiving a notification one minute apart. That is, employee i would receive a notification at minute i. Such a policy is optimal under mean response delays for EV problem and also reduces the cost under uncertainty. We refer to this policy as the Notify and Wait policy with parameters $\eta = 1, w = 1$ ($\text{NAW}_{1,1}$).

The actual NTP problem is a multistage decision problem. However, obtaining the true

stochastic optimal policy is infeasible, so we approximate the solution using a Two-Stage stochastic formulation with 200 scenarios. Consequently, the obtained VSS serves as an estimate or bound of the actual VSS. The VSS is computed as a percentage of the EV objective value, reported below:

- VSS % for NA policy = 81.3%
- VSS % for $\text{NAW}_{1,1}$ policy = 46.0%

The actual VSS is expected to be higher than the values reported above. Even under an optimistic scenario where we use the $\text{NAW}_{1,1}$ policy, the VSS remains significantly high. In both cases, this suggests that the NTP is highly sensitive to uncertainty.

This observation justifies why models trained on offline solutions derived from single scenarios, such as the FIE or DE, exhibit erratic behavior. Even the 2SSE model, which incorporates a limited number of scenarios, struggles to perform well due to extreme uncertainty. The ADE, despite being based on a limited number of scenarios, smooths out some of these uncertainties, leading to more stable and satisfactory results.

6.3 Extending DAgger to consider Contextual Data

In many real-world applications, some parameters of the SDP are uncertain and must be inferred from contextual data. This is also referred to as contextual optimization [Sadana et al., 2024], where the distribution of uncertain parameters that affect the objective and the constraints is unknown. However, correlated side information (covariates or features) is available and can be exploited to make better decisions. For example, GPS-based navigation systems like Google Maps or Waze use contextual optimization to suggest the fastest routes by analyzing current traffic patterns, road closures, and construction work. Airline companies use contextual data such as the time of year (holiday season, back-to-school period) to optimize fares dynamically. In these scenarios, decision-makers utilize historical data, such as previous values of covariates (e.g., weather) and the associated uncertain parameters (e.g., congestion). Data-driven contextual optimization approaches leverage this data to estimate the conditional distribution of the uncertain parameter (or a sufficient statistic) based on the covariate. In contrast, traditional stochastic optimization models disregard contextual information, relying on unconditional distributions of uncertain parameters to inform decisions [Birge and Louveaux, 2011]. This approach can lead to suboptimal decisions [Ban and Rudin, 2019]. The availability of vast data and computational power, along with advances in

ML and optimization techniques, has driven a shift towards contextual optimization [Mišić and Perakis, 2020].

We redefine the basic equations (1.1)-(1.3) for SDP from Chapter 1 to include contextual information reflecting the equations in [Sadana et al., 2024]. Let $\hat{x}$ denote the contextual features that are a part of the state variable, which are simply denoted by x. Alternately, $\hat{x}$ represents a vector of relevant covariates, which are correlated with the uncertain parameters ξ and are revealed before having to choose an action. ξ may represent a vector of random variables which are considered to be independent. We can rewrite the objective as in equation (6.2) below to include the conditional distribution of ξ given the state x.

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{\mathbb{P}(\xi|x)} \left\{ C(\pi(\mathbf{x}), \boldsymbol{\xi}) | x_0 \right\} \tag{6.1}$$

$$\pi^* = \arg\min_{\pi} V^{\pi}(x_0) \tag{6.2}$$

Here $C(\pi(\mathbf{x}), \boldsymbol{\xi}) = \sum_{k=0}^{K} C_k(x_k, \pi(x_k))$ is the total cost. We can also represent equation (6.2) in the form of the famous Bellman equations [Puterman, 1990] by using equation (5.7). The version of the NTP considered so far does not consider any contextual side information. We directly and uniformly sample the response delays from a single dataset. However, the response delays may depend on several contextual features. For example,

- Weather - Some employees may not prefer to work on bad weather days.
- Personal Schedule - Employees may only want to work on certain weekdays and hence may not respond to shifts on certain days.
- Shift Preferences - First introduced in Chapter 4, shift preferences represent important data. Employee response delays may be dependent on whether their favorite shift is still available or not.

Considering contextual factors within the problem settings can significantly enhance decision-making by allowing for a more adaptive response to various real-world conditions. For instance, during adverse weather conditions, fewer people may be inclined to respond to notifications promptly, suggesting that notifications should be sent out faster to ensure sufficient coverage. Conversely, certain segments of the workforce may prefer to work on weekends and, as a result, respond more quickly to system notifications on those days. This scenario implies a more gradual notification policy during such times.

In the offline optimization setting, these contextual factors are not explicitly included in the mathematical formulation. However, the optimization process, given access to full future information, inherently adapts its decisions based on these conditions. Thus, offline decisions are implicitly shaped by contextual influences. In other words, the expert's decisions will inherently reflect adjustments based on contextual information, allowing the model to learn policies that are more responsive and attuned to real-world conditions. The DAgger algorithm presented in Algorithm 15 can be seamlessly adapted to incorporate contextual data. All relevant contextual information previously discussed can be treated as part of the state variable. Consequently, when training a machine learning model to make decisions, these contextual features can be integrated directly into the dataset D. This approach enables the model to learn correlations between contextual variables and the decisions recommended by the expert.

6.4 Conclusion

This chapter presents the results obtained from the three different experts introduced in Chapter 5. Among the tested experts, the Aggregated Deterministic Expert, combined with the Vanilla DAgger decision rule, demonstrated the best overall performance. It significantly outperformed the ONP heuristic, achieving notable improvements in reducing both total bumps and vacant shifts.

Additionally, we analyzed the imperfections inherent in the experts considered, shedding light on their limitations and the potential impact on policy performance. These analyses provide valuable insights into the trade-offs involved in leveraging different types of experts to learn effective decision-making policies.

CHAPTER 7 CONCLUSION

This dissertation explores the Notification Timing Problem (NTP) in dynamic, on-demand personnel scheduling, a new challenge in certain modern labor platforms that must balance quick shift fulfillment with minimizing disruptions for employees. Through the novel integration of machine learning, optimization, and imitation learning, this research contributes a methodologically rigorous and practically impactful solution that advances the state-of-the-art in personnel scheduling. In this chapter, we present the summary of our work in Section 7.1 and future research directions in Section 7.2.

7.1 Summary of Works

In Chapter 3, we define the Notification Timing Problem (NTP), a novel scheduling decision problem (SDP) that arises within on-demand labor platforms. These platforms match clients who need specific tasks completed with casual workers seeking job opportunities. The platform operates as a self-scheduling system, announcing open shifts to workers. However, shifts are made available to workers at specific times, prioritized by their seniority to align with client objectives. This prioritization allows senior workers to select shifts earlier, even allowing them to replace junior workers who may have chosen their preferred shift, responding before the senior employee.

The objective of the NTP is to develop a policy that sequentially notifies workers of job openings according to their seniority. This policy aims to minimize disruptions caused when senior employees replace juniors while ensuring all shifts are filled, accounting for the uncertain behavior of employees. A key finding in this research is the demonstration that the NTP, even in a simplified offline (full information) form, is NP-complete. This result highlights the computational intractability of finding exact solutions in real-time applications. The complexity of the NTP stems from the need to manage "bumps" while fully staffing shifts. By reducing the NTP to the subset sum problem, we provide foundational insights into its computational limits, reinforcing the necessity for heuristic and approximation methods.

To address the dynamic nature of the NTP, this research introduces an Offline Notification Policy (ONP) heuristic in Chapter 4. This approach utilizes aggregated offline solutions to inform real-time decisions. Specifically, it simulates optimal solutions across various scenarios and applies these deterministic solutions in real-time contexts. By leveraging features from multiple offline instances, the ONP strikes a balance between optimality and computa-

tional efficiency. Empirical results on real-world data demonstrate that the ONP significantly reduces scheduling replacements. We further validate the policy's robustness by testing it against different hypothetical employee preference profiles.

We then turn our attention to leveraging machine learning models for online decision-making by imitating an expert. Ideally, we would aim to imitate a multistage optimization model as the expert. However, obtaining optimal or even near-optimal actions for all possible states is infeasible due to the complexity of the problem. As a practical alternative, we focus on imitating simpler optimization models that operate under limited information. These models are designed to generate optimal decisions within a restricted set of scenarios, making them more computationally tractable. Based on these considerations, we define four types of experts: Full-Information Expert (FIE), Deterministic Expert (DE), Two-Stage Stochastic Expert (2SSE), and Aggregated Deterministic Expert (ADE). Chapter 5 provides a detailed discussion on expert models suitable for imitation learning. We establish a taxonomy of experts that categorizes different approaches to expert-based learning, offering a structured framework to guide both theoretical understanding and practical experimentation. This taxonomy helps contextualize the choice of expert models and their implications for algorithm design and performance evaluation.

One of the most innovative contributions of this research is the application of imitation learning, specifically through the DAgger (Dataset Aggregation) algorithm, as discussed in Chapter 6. The DAgger algorithm iteratively refines the scheduling policy by aggregating a dataset of expert decisions from an algorithmic expert—the offline solution, which is tractable for the problem sizes considered. This technique allows the model to learn from optimal decisions without incurring the computational costs of generating exact real-time solutions. Unlike traditional behavior cloning, which may suffer from compounding errors, DAgger mitigates this issue by continuously updating the policy with corrective feedback from the expert. We combine different expert optimization models within the DAgger framework.

Experimental results indicate that the DAgger-trained model using the Aggregated Deterministic Expert (ADE) not only surpasses the performance of the currently deployed policy used by our industrial partner but also outperforms the ONP. However, models trained on other expert types fail to achieve comparable results. A deeper analysis reveals that the problem exhibits a high degree of stochasticity, making the optimal actions or targets defined by the Full-Information Expert (FIE), Deterministic Expert (DE), and even the Two-Stage Stochastic Expert (2SSE) unstable. Specifically, there is significant variance among the target actions for a given state, as these experts are highly biased toward the specific scenarios they solve. This variability reduces their reliability for imitation learning in a dynamic environment. The

strong performance of the ADE-trained model highlights the potential of imitation learning in tackling complex, real-world scheduling problems, particularly when traditional optimization methods are computationally prohibitive. By leveraging aggregated decision-making across multiple scenarios, ADE provides more stable and generalizable targets, making it a promising approach for large-scale stochastic decision problems.

7.2 Limitations and Future Research

The Notification Timing Problem (NTP) includes a wealth of side information that could significantly enhance the decision-making process. This information encompasses environmental variables such as weather, shift types, and locations, as well as worker demographics like gender, location, and age. Some of this information is available while others are not stored by our industrial partner. However, the most influential factors are likely to be historical user response delays and individual shift preferences. By incorporating this data into the state representation, the learning process can leverage it to improve decision-making, potentially reducing "bumps" more effectively. Additionally, parameters such as the number of employees, number of shifts, cutoff time, and the planning horizon have been treated as constants. In real-world scenarios, these parameters may vary and exhibit stochastic behavior. As a result, the Offline Notification Policy (ONP) approach may not be directly applicable to derive an effective policy under such uncertainty. This limitation presents an area for further research, justifying the potential for machine learning models to address the dynamic and stochastic nature of the problem.

In our experimentation, we explored both naive and vanilla versions of the DAgger algorithm. In the literature review, we discussed variations of DAgger as well as other imitation learning methods, with particular interest in MEGA DAgger, which aims to address key limitations encountered in our approach. As discussed in Chapter 6, a primary challenge is the suboptimality of the expert used in training. Our expert model is derived from static, deterministic optimization problems and assumes future knowledge, which biases its guidance. These expert models are also somewhat volatile, often providing inconsistent target actions for the same state. To mitigate this, we aggregate multiple target actions to smooth out volatility. However, the resulting targets can still be imperfect. Improving these target labels could involve methods such as confidence intervals and similarity scoring. For instance, grouping states with high similarity and adjusting labels based on confidence intervals could refine the learning process and provide more consistent guidance. A possible solution is to use an alternate decision rule within the DAgger algorithm that can identify instances where the expert suggests actions that appear to be outliers due to an outlier scenario. This can

be achieved by introducing bounds to detect such deviations or by comparing the expert's suggested actions against a well-performing heuristic. By incorporating these checks, the decision rule can selectively override unreliable expert actions, improving the stability and robustness of the learned policy.

Using the DAgger algorithm, we predict the time to wait before sending the next notification. Alternatively, one could consider alternate variables as targets, like the number of notifications to be sent. One could even change it to categorical and learn a proper distribution over the wait times in a given epoch.

The NTP problem, as discussed in Chapter 3, is marked by high stochasticity, introducing substantial uncertainty beyond the decision-maker's control. This unpredictability makes it difficult to design policies that generalize well across different problem realizations. Extending this framework to other SDPs, such as resource allocation or vehicle routing, where the decision-maker has more control over outcomes, presents an exciting avenue for future research.

Several challenges would arise in this context. The first involves action aggregation in the ADE, particularly for problems involving discrete decisions, such as selecting which nodes to visit in vehicle routing. In such cases, a voting-based approach could be used: for each state, solve across multiple scenarios and identify the nodes most frequently chosen as targets.

The second challenge is ensuring feasibility. The aggregated targets must satisfy problem-specific constraints, such as vehicle capacity limits. Even after constructing a feasible aggregated target, the learned model may still produce infeasible predictions at test time. To address this, one may need to take a corrective step to restore feasibility. However, this step must remain lightweight, ensuring that the bulk of the decision-making burden is handled by the model's predictions, with only minor adjustments made afterwards. The NTP also has an easier deterministic version of the problem to solve, which makes applying DAgger easier. This might not be the case for all problems. Thus, one may need to put effort into defining simple experts for such cases which signifies a trade off between quality and efficiency.

Another promising area of investigation within imitation learning is handling problems with an expanding decision space. Many problems in operations research involve combinatorial decision-making at each stage, making it difficult to apply standard behavior cloning techniques directly. As the decision space grows, naïve imitation learning approaches may struggle to capture complex dependencies, necessitating more advanced strategies to effectively mimic expert behavior. Addressing these challenges could significantly enhance the applicability of imitation learning in large-scale combinatorial optimization problems.

REFERENCES

[Abbeel and Ng, 2004] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1.

[Allon et al., 2023] Allon, G., Cohen, M. C., and Sinchaisri, W. P. (2023). The impact of behavioral and economic drivers on gig economy workers. *Manufacturing & Service Operations Management*, 25(4):1376–1393.

[Alsaleh and Sayed, 2020] Alsaleh, R. and Sayed, T. (2020). Modeling pedestrian-cyclist interactions in shared space using inverse reinforcement learning. *Transportation research part F: traffic psychology and behaviour*, 70:37–57.

[Alvarez et al., 2017] Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195.

[Applegate, 2006] Applegate, D. L. (2006). *The traveling salesman problem: a computational study*, volume 17. Princeton university press.

[Arora and Doshi, 2021] Arora, S. and Doshi, P. (2021). A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500.

[Arulkumaran et al., 2017] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

[Ban and Rudin, 2019] Ban, G.-Y. and Rudin, C. (2019). The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108.

[Baty et al., 2024] Baty, L., Jungel, K., Klein, P. S., Parmentier, A., and Schiffer, M. (2024). Combinatorial optimization-enriched machine learning to solve the dynamic vehicle routing problem with time windows. *Transportation Science*.

[Bello et al., 2016] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

[Ben-Tal and Nemirovski, 1999] Ben-Tal, A. and Nemirovski, A. (1999). Robust solutions of uncertain linear programs. *Operations research letters*, 25(1):1–13.

[Bengio et al., 2021] Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

[Benjaafar et al., 2022] Benjaafar, S., Ding, J.-Y., Kong, G., and Taylor, T. (2022). Labor welfare in on-demand service platforms. *Manufacturing & Service Operations Management*, 24(1):110–124.

[Bent and Van Hentenryck, 2004] Bent, R. W. and Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6):977–987.

[Berg et al., 2014] Berg, B. P., Denton, B. T., Erdogan, S. A., Rohleder, T., and Huschka, T. (2014). Optimal booking and scheduling in outpatient procedure centers. *Computers & Operations Research*, 50:24–37.

[Bertsekas, 2012] Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific.

[Bertsimas and Kallus, 2020] Bertsimas, D. and Kallus, N. (2020). From predictive to prescriptive analytics. *Management Science*, 66(3):1025–1044.

[Bertsimas and Koduri, 2022] Bertsimas, D. and Koduri, N. (2022). Data-driven optimization: A reproducing kernel hilbert space approach. *Operations Research*, 70(1):454–471.

[Bertsimas and Thiele, 2006] Bertsimas, D. and Thiele, A. (2006). A robust optimization approach to inventory theory. *Operations research*, 54(1):150–168.

[Bhatti et al., 2020] Bhatti, S. S., Gao, X., and Chen, G. (2020). General framework, opportunities and challenges for crowdsourcing techniques: A comprehensive survey. *Journal of Systems and Software*, 167:110611.

[Birge and Louveaux, 2011] Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

[Bishop and Nasrabadi, 2006] Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

[Borodin and El-Yaniv, 2005] Borodin, A. and El-Yaniv, R. (2005). *Online computation and competitive analysis*. cambridge university press.

[Brantley et al., 2020] Brantley, K., Sharaf, A., and Daumé III, H. (2020). Active imitation learning with noisy guidance. *arXiv preprint arXiv:2005.12801*.

[Brown et al., 2020] Brown, D. S., Goo, W., and Niekum, S. (2020). Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on robot learning*, pages 330–359. PMLR.

[Cachon et al., 2017] Cachon, G. P., Daniels, K. M., and Lobel, R. (2017). The role of surge pricing on a service platform with self-scheduling capacity. *Manufacturing & Service Operations Management*, 19(3):368–384.

[Causa et al., 2022] Causa, O., Abendschein, M., Luu, N., Soldani, E., and Soriolo, C. (2022). The post-covid-19 rise in labour shortages.

[Chan et al., 2012] Chan, C. W., Farias, V. F., Bambos, N., and Escobar, G. J. (2012). Optimizing intensive care unit discharge decisions with patient readmissions. *Operations research*, 60(6):1323–1341.

[Chu et al., 2023] Chu, H., Zhang, W., Bai, P., and Chen, Y. (2023). Data-driven optimization for last-mile delivery. *Complex & Intelligent Systems*, 9(3):2271–2284.

[Daniel, 2017] Daniel, K. (2017). *Thinking, fast and slow*.

[Daumé et al., 2009] Daumé, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75:297–325.

[De Filippo et al., 2021] De Filippo, A., Lombardi, M., and Milano, M. (2021). Integrated offline and online decision making under uncertainty. *Journal of Artificial Intelligence Research*, 70:77–117.

[Donovan et al., 2016] Donovan, S. A., Bradley, D. H., and Shimabukuru, J. O. (2016). What does the gig economy mean for workers?

[Donti et al., 2017] Donti, P., Amos, B., and Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30.

[Elmachtoub and Grigas, 2022] Elmachtoub, A. N. and Grigas, P. (2022). Smart "predict, then optimize". *Management Science*, 68(1):9–26.

[Farazi et al., 2021] Farazi, N. P., Zou, B., Ahamed, T., and Barua, L. (2021). Deep reinforcement learning in transportation research: A review. *Transportation research interdisciplinary perspectives*, 11:100425.

[Ferreira et al., 2016] Ferreira, K. J., Lee, B. H. A., and Simchi-Levi, D. (2016). Analytics for an online retailer: Demand forecasting and price optimization. *Manufacturing & service operations management*, 18(1):69–88.

[Florio et al., 2020] Florio, A. M., Hartl, R. F., and Minner, S. (2020). New exact algorithm for the vehicle routing problem with stochastic demands. *Transportation Science*, 54(4):1073–1090.

[Garstka and Wets, 1974] Garstka, S. J. and Wets, R. J. B. (1974). On decision rules in stochastic programming. *Mathematical Programming*, 7:117–143.

[Gary and Johnson, 1979] Gary, M. R. and Johnson, D. S. (1979). Computers and intractability: A guide to the theory of np-completeness.

[Gasse et al., 2019] Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32.

[Gombolay et al., 2018] Gombolay, M., Jensen, R., Stigile, J., Golen, T., Shah, N., Son, S.-H., and Shah, J. (2018). Human-machine collaborative optimization via apprenticeship scheduling. *Journal of Artificial Intelligence Research*, 63:1–49.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Graham and Woodcock, 2019] Graham, M. and Woodcock, J. (2019). The gig economy: a critical introduction. *Polity*, 54.

[Greif et al., 2024] Greif, T., Bouvier, L., Flath, C. M., Parmentier, A., Rohmer, S. U., and Vidal, T. (2024). Combinatorial optimization and machine learning for dynamic inventory routing. *arXiv preprint arXiv:2402.04463*.

[Hastie et al., 2001] Hastie, T., Friedman, J., and Tibshirani, R. (2001). *The Elements of Statistical Learning*. Springer New York.

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573.

[Hoque et al., 2021] Hoque, R., Balakrishna, A., Putterman, C., Luo, M., Brown, D. S., Seita, D., Thananjeyan, B., Novoseller, E., and Goldberg, K. (2021). Lazydagger: Reducing context switching in interactive imitation learning. In *2021 IEEE 17th international conference on automation science and engineering (case)*, pages 502–509. IEEE.

[Hussein et al., 2017] Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

[Karachiwalla and Pinkow, 2021] Karachiwalla, R. and Pinkow, F. (2021). Understanding crowdsourcing projects: A review on the key design elements of a crowdsourcing initiative. *Creativity and innovation management*, 30(3):563–584.

[Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

[Kelly et al., 2019] Kelly, M., Sidrane, C., Driggs-Campbell, K., and Kochenderfer, M. J. (2019). Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE.

[Khalil et al., 2017] Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30.

[Khalil et al., 2016] Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[Kong et al., 2022] Kong, L., Cui, J., Zhuang, Y., Feng, R., Prakash, B. A., and Zhang, C. (2022). End-to-end stochastic optimization with energy-based model. *Advances in Neural Information Processing Systems*, 35:11341–11354.

[Kool et al., 2018] Kool, W., Van Hoof, H., and Welling, M. (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

[Kotary et al., 2021] Kotary, J., Fioretto, F., Van Hentenryck, P., and Wilder, B. (2021). End-to-end constrained optimization learning: A survey. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4475–4482. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

[Kullman et al., 2022] Kullman, N. D., Cousineau, M., Goodson, J. C., and Mendoza, J. E. (2022). Dynamic ride-hailing with electric vehicles. *Transportation Science*, 56(3):775–794.

[Larsen et al., 2018] Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., and Lodi, A. (2018). Predicting solution summaries to integer linear programs under imperfect information with machine learning. *arXiv preprint arXiv:1807.11876*, 2(4):14.

[Laskey et al., 2016] Laskey, M., Staszak, S., Hsieh, W. Y.-S., Mahler, J., Pokorny, F. T., Dragan, A. D., and Goldberg, K. (2016). Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 462–469. IEEE.

[Li et al., 2023] Li, Z., Xu, T., Qin, Z., Yu, Y., and Luo, Z.-Q. (2023). Imitation learning from imperfection: Theoretical justifications and algorithms. *Advances in Neural Information Processing Systems*, 36:18404–18443.

[Liu et al., 2021] Liu, S., He, L., and Max Shen, Z.-J. (2021). On-time last-mile delivery: Order assignment with travel-time predictors. *Management Science*, 67(7):4095–4119.

[Liu et al., 2020] Liu, S., Jiang, H., Chen, S., Ye, J., He, R., and Sun, Z. (2020). Integrating dijkstra's algorithm into deep inverse reinforcement learning for food delivery route planning. *Transportation Research Part E: Logistics and Transportation Review*, 142:102070.

[Mandi et al., 2024] Mandi, J., Kotary, J., Berden, S., Mulamba, M., Bucarey, V., Guns, T., and Fioretto, F. (2024). Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701.

[Mandi et al., 2020] Mandi, J., Stuckey, P. J., Guns, T., et al. (2020). Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1603–1610.

[Mao et al., 2016] Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56.

[Marcos Alvarez et al., 2016] Marcos Alvarez, A., Wehenkel, L., and Louveaux, Q. (2016). Online learning for strong branching approximation in branch-and-bound.

[Mazyavkina et al., 2021] Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400.

[Menda et al., 2017] Menda, K., Driggs-Campbell, K., and Kochenderfer, M. J. (2017). Dropoutdagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1709.06166*.

[Menda et al., 2019] Menda, K., Driggs-Campbell, K., and Kochenderfer, M. J. (2019). Ensembledagger: A bayesian approach to safe imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5041–5048. IEEE.

[Mišić and Perakis, 2020] Mišić, V. V. and Perakis, G. (2020). Data analytics in operations management: A review. *Manufacturing & Service Operations Management*, 22(1):158–169.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[Mukhopadhyay and Vorobeychik, 2017] Mukhopadhyay, A. and Vorobeychik, Y. (2017). Prioritized allocation of emergency responders based on a continuous-time incident prediction model. In *International conference on autonomous agents and multiagent systems*.

[Nazari et al., 2018] Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31.

[Nieto-Isaza et al., 2022] Nieto-Isaza, S., Fontaine, P., and Minner, S. (2022). The value of stochastic crowd resources and strategic location of mini-depots for last-mile delivery: A benders decomposition approach. *Transportation Research Part B: Methodological*, 157:62–79.

[Oroojlooyjadid et al., 2020] Oroojlooyjadid, A., Snyder, L. V., and Takáč, M. (2020). Applying deep learning to the newsvendor problem. *IISE Transactions*, 52(4):444–463.

[Özarık et al., 2024] Özarık, S. S., da Costa, P., and Florio, A. M. (2024). Machine learning for data-driven last-mile delivery optimization. *Transportation Science*, 58(1):27–44.

[Pang et al., 2020] Pang, Y., Kashiyama, T., Yabe, T., Tsubouchi, K., and Sekimoto, Y. (2020). Development of people mass movement simulation framework based on reinforcement learning. *Transportation research part C: emerging technologies*, 117:102706.

[Pham et al., 2023] Pham, T. S., Legrain, A., De Causmaecker, P., and Rousseau, L.-M. (2023). A prediction-based approach for online dynamic appointment scheduling: A case study in radiotherapy treatment. *INFORMS Journal on Computing*, 35(4):844–868.

[Powell, 2007] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons.

[Powell, 2022] Powell, W. B. (2022). *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. John Wiley & Sons.

[Puterman, 1990] Puterman, M. L. (1990). Markov decision processes. *Handbooks in operations research and management science*, 2:331–434.

[Rajaraman et al., 2020] Rajaraman, N., Yang, L., Jiao, J., and Ramchandran, K. (2020). Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924.

[Rautenstrauß and Schiffer, 2025] Rautenstrauß, M. and Schiffer, M. (2025). Optimization-augmented machine learning for vehicle operations in emergency medical services. *arXiv preprint arXiv:2503.11848*.

[Raykar et al., 2009] Raykar, V. C., Yu, S., Zhao, L. H., Jerebko, A., Florin, C., Valadez, G. H., Bogoni, L., and Moy, L. (2009). Supervised learning from multiple experts: whom to trust when everyone lies a bit. In *Proceedings of the 26th Annual international conference on machine learning*, pages 889–896.

[Raykar et al., 2010] Raykar, V. C., Yu, S., Zhao, L. H., Valadez, G. H., Florin, C., Bogoni, L., and Moy, L. (2010). Learning from crowds. *Journal of machine learning research*, 11(4).

[Rogers et al., 2003] Rogers, E. M., Singhal, A., and Quinlan, M. M. (2003). Diffusion of innovations/everett m. *Rogers. NY: Simon and Schuster*, 576:2003.

[Ross and Bagnell, 2010] Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings.

[Ross and Bagnell, 2014] Ross, S. and Bagnell, J. A. (2014). Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

[Ross et al., 2011] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

[Sadana et al., 2024] Sadana, U., Chenreddy, A., Delage, E., Forel, A., Frejinger, E., and Vidal, T. (2024). A survey of contextual optimization methods for decision-making under uncertainty. *European Journal of Operational Research*.

[Schrittwieser et al., 2020] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

[Sekhari et al., 2024] Sekhari, A., Sridharan, K., Sun, W., and Wu, R. (2024). Selective sampling and imitation learning via online regression. *Advances in Neural Information Processing Systems*, 36.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[Slivkins and Vaughan, 2014] Slivkins, A. and Vaughan, J. W. (2014). Online decision making in crowdsourcing markets: Theoretical challenges. *ACM SIGecom Exchanges*, 12(2):4–23.

[Sun et al., 2023] Sun, X., Yang, S., and Mangharam, R. (2023). Mega-dagger: Imitation learning with multiple imperfect experts. *arXiv preprint arXiv:2303.00638*.

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

[Tang et al., 2020] Tang, Y., Agrawal, S., and Faenza, Y. (2020). Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR.

[Taylor, 2018] Taylor, T. A. (2018). On-demand service platforms. *Manufacturing & Service Operations Management*, 20(4):704–720.

[Torabi et al., 2018] Torabi, F., Warnell, G., and Stone, P. (2018). Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*.

[Torres et al., 2022] Torres, F., Gendreau, M., and Rei, W. (2022). Vehicle routing with stochastic supply of crowd vehicles and time windows. *Transportation Science*, 56(3):631–653.

[Valko et al., 2013] Valko, M., Ghavamzadeh, M., and Lazaric, A. (2013). Semi-supervised apprenticeship learning. In *European workshop on reinforcement learning*, pages 131–142. PMLR.

[Vinyals et al., 2015] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. *Advances in neural information processing systems*, 28.

[Wang and Chang, 2021] Wang, T. and Chang, D. E. (2021). Robust navigation for racing drones based on imitation learning and modularization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13724–13730. IEEE.

[Wang et al., 2023] Wang, Y., Dong, M., Du, B., and Xu, C. (2023). Imitation learning from purified demonstration. *arXiv preprint arXiv:2310.07143*.

[Wang et al., 2021] Wang, Y., Xu, C., Du, B., and Lee, H. (2021). Learning to weight imperfect demonstrations. In *International Conference on Machine Learning*, pages 10961–10970. PMLR.

[Wu et al., 2019] Wu, Y.-H., Charoenphakdee, N., Bao, H., Tangkaratt, V., and Sugiyama, M. (2019). Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827. PMLR.

[Xu et al., 2022] Xu, H., Zhan, X., Yin, H., and Qin, H. (2022). Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *International Conference on Machine Learning*, pages 24725–24742. PMLR.

[Yan et al., 2022] Yan, X., Liu, W., Shi, V., and Liu, T. (2022). On-demand service platform operations management: a literature review and research agendas. *Modern Supply Chain Research and Applications*, 4(2):105–121.

[Yoganarasimhan, 2013] Yoganarasimhan, H. (2013). The value of reputation in an online freelance marketplace. *Marketing Science*, 32(6):860–891.

[Yu et al., 2021] Yu, C., Liu, J., Nemati, S., and Yin, G. (2021). Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36.

[Zhang et al., 2020] Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., and Chi, X. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632.

[Zhang and Cho, 2016] Zhang, J. and Cho, K. (2016). Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*.

[Zhen et al., 2021] Zhen, Y., Khan, A., Nazir, S., Huiqi, Z., Alharbi, A., and Khan, S. (2021). Crowdsourcing usage, task assignment methods, and crowdsourcing platforms: A systematic literature review. *Journal of Software: Evolution and Process*, 33(8):e2368.

[Zhou et al., 2002] Zhou, Z.-H., Wu, J., and Tang, W. (2002). Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263.

[Ziebart et al., 2008] Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. (2008). Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.

APPENDIX A PROOF

A.1 Pseudo Code to find a bump chain

Let $I = \{I_l\}_{l \in \mathcal{L}}$ and $L = \{L_j\}_{j \in \mathcal{E}}$.

Algorithm 14: To construct $\mathcal{B}_i$

```
1  Function Compile(i, e_i, I, L):
2      B_i = {};
3      i' ← i;
4      while I_{L_{i'}} ≠ φ do
5          B_i = B_i ∪ {I_{L_{i'}}};
6          i' ← I_{L_{i'}};
7      end
8      i ← i + 1;
9      return B_i;
10 end
```

A.2 Proofs

Note: Any reference to a bump is a reference to a potential bump.

Theorem 1. *Given an instance $I = \{H, M, \mathbf{r}\}$, let $S^*_{\mathcal{X}}$ denote an optimal schedule with preferences $\mathcal{X}$ such that $\mathcal{X} = \{\mathcal{X}_i\}_{i \in \mathcal{E}}$. Also, let S^*_p be the schedule that minimizes the total potential bumps for this instance, and S^*_I be the schedule that minimizes the total bumps with identical preferences. Then, $B_{S^*_{\mathcal{X}}} \leq B_{S^*_p} = B_{S^*_I}$.*

Proof. We show this in two parts.

1) It is trivial to say that $B_{S^*_{\mathcal{X}}} \leq B_{S^*_p}$. This is because potential bumps are all cases when a senior employee has responded later than a junior employee.

2) Now consider the case when preferences are identical. By 1), $B_{S^*_I} \leq B_{S^*_p}$. Let $\{1, \ldots, M\}$ represent the set of shifts where 1 represents the most preferred shift, 2 represents the second most preferred and so on in the identical preferences setting. Consider an employee i responding back at e_i and choosing the current preferred and

available shift L_i. Let

$$\mathcal{P}_i = \{j_1, j_2, \ldots, j_k : e_{j_o} < e_i, i < j_1 < \cdots < j_k\} \tag{A.1}$$

represent the potential bumps as in Equation 2. j_1 represents the least junior employee responding before i; ($j_1 = \{\min j : j > i, e_i > e_j \, j \in \mathcal{E}\}$) and is the first potential candidate to get bumped if i responds. If $\mathcal{B}_i = \phi$, there are no potential bumps induced by a response from i.

Consider any i such that $\mathcal{P}_i \neq \phi$, that is, there are some potential bumps. Under identical preferences, $\mathcal{X}_i = \mathcal{X}_j = \{1, \ldots, M\}, \forall j \in \mathcal{P}_i$. We now show that $\mathcal{P}_i$ is also a bump chain defined by Equation 1, for that employee i when preferences are identical. That is, all potential bumps are actually realised bumps when preferences are identical ($\mathcal{P}_i = \mathcal{B}_i$) for all employees $i \in \mathcal{E}$ and, as a consequence, $B_{S_I^*} \geq B_{S_p^*}$.

Let $I_l = j_1$ at time e_i, that is employee j_1 occupies shift l. All shifts in the set $\{1, \ldots, l-1\}$ are chosen by senior employees to j_1, otherwise j_1 would occupy them as they more preferred in $\mathcal{X}_{j_1}$. This also means that $L_j = l + 1$. Consequently, for any employee in $j_o \in \mathcal{P}_i$, $I_{l+o-1} = j_o$. Also

$$L_{j_o} = \begin{cases} l + o & \text{if } l + o \leq M; \\ \phi & \text{else;} \end{cases} \tag{A.2}$$

As i is senior to j_1, shift l will always be available to him as long as j_1 occupies it. When i responds, he too will not have any shifts available to him in the set $\{1, \ldots, l-1\}$. If such a shift were available, then j would occupy it because it is more preferred. Hence $l = L_i$ and $I_l = j_1$ at the time i responds. Employee i will bump j_1, and j_1 is the first element of the bump set in Equation 1. As a consequence, j_1 will then similarly bump j_2 for the next shift $l + 1$ as $L_j = l + 1$. Thus, this will induce a chain of bumps for all employees in $\mathcal{P}_i$. For the last person in the chain, either there is no shift available for j_1, or his preferred shift will be unoccupied from Equation (A.2). Thus, all employees in $\mathcal{P}_i$ are bumped $\forall i \in \mathcal{E}$, and it is a bump chain $\mathcal{B}_i$. Therefore, $B_{S_p^*} \leq B_{S_I^*}$ and $B_{S_p^*} = B_{S_I^*}$.

Proposition 1. *The minimum makespan of a NBS is* $C_0^* = r_1 + \sum_{i=1}^{n} (r_{i+1} - r_i)^+$.

Proof. **Case 1**: $r_i \leq r_{i+1}, \forall i \in \mathcal{E}$. Therefore, we can send notifications to employees i and $i+1$ at the same time ($s_i = s_{i+1}$) without introducing a bump. The makespan of the schedule

is increased by $r_{i+1} - r_i$.

Case 2: $r_i > r_{i+1}, \forall i \in \mathcal{E}$. The only way to avoid a bump, in this case, is to make sure that both of them respond at the same time $e_i = e_{i+1}$. This is easily ensured by setting $s_{i+1} = s_i + r_i - r_{i+1}$. Hence, the makespan is not increased in this case.

□

Proposition 2. *The block schedule has the following properties:*

1. $\forall k \in A, s_{i_k} = s_{i_k - 1}$.
2. $\forall i \in \mathcal{E}_k^S, \forall k \in A$, $e_i = e_{i_k}$ *or* $s_i = s_{i_k}$.
3. $s_{i_{k+1}} = s_{i_k} + \mathbf{1}_{k \notin A'} a_k$, *where* $\mathbf{1}$ *denotes indicator variable.*
4. $B(S(A')) = \sum\limits_{j \in A'} a_j$
5. $C(S(A')) = C_0^* - \sum\limits_{j \in A'} a_j$.

Proof. Property 1, 2, and 3 are true by construction.

4. Consider some $k \in A'$, $r_{i_k} > r_i$ from Equation (7) in the main article, and $s_{i_k} = s_i$ from Equation (8) in the main article, $\forall i \in \mathcal{E}_k^S$. As a consequence

$$e_{i_k} = r_{i_k} + s_{i_k} > r_i + s_i = e_i, \forall i_k < i < i_{k+1}.$$

   Since $a_k = i_{k+1} - i_k$ and using proposition Corollary 1, $b_{i_k} = a_k$.
   Now consider some $k \in A \backslash A'$, and $e_{i_k} = s_{i_k} + r_{i_k} = s_i + r_i = e_i, \forall i_k < i < i_{k+1}$. As a consequence, no employee is potentially bumped by i_k and $b_{i_k} = 0$. Hence, the total potential bumps are given by

$$B(S(A')) = \sum_{i \in \mathcal{E}} b_i = \sum_{k \in A} b_{i_k} = \sum_{k \in A'} a_k \tag{A.3}$$

5. $C(S(A')) = C_0^* - \sum\limits_{j \in A'} a_j$.
   Using property 3 and by recursion:

$$s_{i_{k+1}} = \sum_{j=1 : j \notin A'}^{k} a_j \tag{A.4}$$

Now consider the employee with priority M.

$$s_M = s_{M-1} = s_{i_N} + \mathbf{1}_{N \notin A'} a_N. \tag{A.5}$$

Substitute equation Equation (A.4) in Equation (A.5),

$$s_M = \sum_{j=1: j \notin A'}^{N-1} a_j + \mathbf{1}_{N \notin A'} a_N$$
$$s_M = \sum_{j \notin A'} a_j$$

As a consequence one can now find $C(S(A')) = e_M$

$$\begin{aligned} C(S(A')) &= s_M + r_M \\ &= \sum_{j \notin A'} a_j + \sum_{j \in A} a_j \\ &= 2 * \sum_{j \in A} a_j - \sum_{j \in A'} a_j \\ &= C_0^* - \sum_{j \in A'} a_j \end{aligned}$$

□

Proposition 3. *For any* $NTP\Big([a_j]_{j \in \mathcal{A}}, W\Big)$, $C = e_M$.

Proof. **Proof of Proposition 3** It suffices to show that $r_M \geq r_i$, since $s_M \geq s_i$, due to seniority $\forall i \in \mathcal{E}$.
Case 1: $i = i_k \in \mathcal{E}^{\mathcal{C}}$.

$$r_{i_k} = \sum_{j=1}^{k} a_j \leq \sum_{j \in A} a_j = r_M$$

Case 2: $i \in \mathcal{E}_k^S : k \in A$.

$$r_i = r_{i_{k-1}} = \sum_{j=1}^{k-1} a_j < \sum_{j \in A} a_j = r_M$$

Hence $r_M \geq r_i$, $\forall i \in \mathcal{E}$. □

Proposition 4. $\forall k \in A$, $\forall l \in A$ *employee* i_k *cannot bump employee* i_l.

Proof.

Case 1 $k \geq l$: The proposition is true due to seniority $\forall i_k, i_l \in \mathcal{E}$.

Case 2 $k < l$: To show that $e_{i_k} \leq e_{i_l}$. Consider $i_k \in \mathcal{E}^{\mathcal{C}}$ such that $r_{i_k} = \sum_{j=1}^{k} a_j$ from Equation 7 in the main article;

$$r_{i_l} - r_{i_k} = \sum_{j=1}^{l} a_j - \sum_{j=1}^{k} a_j$$
$$r_{i_l} - r_{i_k} = \sum_{j=k+1}^{l} a_j > 0$$

Hence $r_{i_l} > r_{i_k}$. Due to seniority $s_{i_l} \geq s_{i_k}$. Adding both inequalities

$$r_{i_l} + s_{i_l} > r_{i_k} + s_{i_k}$$
$$e_{i_l} > e_{i_k}$$

□

Proposition 5. $\forall i \in \mathcal{E}^{\mathcal{S}}$, $\forall i' \in \mathcal{E}$ *employee* i *cannot bump employee* i'.

Proof.

Case 1 $i \geq i'$: The proposition is true due to seniority $\forall i \in \mathcal{E}^s, i' \in \mathcal{E}$.

Case 2 $i < i'$: To show that $e_i \leq e_{i'}$.

Response delay for i, $r_i = r_{i_{k-1}} = \sum_{j=1}^{k-1} a_j$ where $i_k < i < i_{k+1}$ and $k \in A$ from Equation (7) in the main article;

$$r_{i'} = r_{i_{k'-1}} : i_{k'} < i' < i_{k'+1} \tag{A.6}$$

Since $i < i'$, $k \leq k'$.

$$r_{i'} - r_i = r_{i_{k'-1}} - r_{i_{k-1}}$$
$$r_{i'} - r_i = \sum_{j=1}^{k'-1} a_j - \sum_{j=1}^{k-1} a_j$$
$$r_{i'} - r_i \geq 0$$

Hence $r_{i'} \geq r_i$. Due to seniority $s_{i'} \geq s_i$. As a consequence, $e_{i'} \geq e_i$. □

Proposition 6. $\forall k \in A$, $\forall i \in \mathcal{E} : i \geq i_{k+1}$, *employee* i_k *cannot bump employee* i.

Proof. $\forall k \in A$, $\forall i \in \mathcal{E} : i \geq i_{k+1}$, it suffices to show $r_{i_k} \leq r_i$.
Case 1: For $i \in \mathcal{E}^{\mathcal{C}} \cap \{i' : i' \geq i_{k+1}, i' \in \mathcal{E}\}$, the inequality holds due to proposition Proposition 3.
Case 2: For $i \in \mathcal{E}^{\mathcal{S}} \cap \{i' : i' \geq i_{k+1}, i' \in \mathcal{E}\}$, $r_i = r_{i_{l-1}}$ where $i \in \mathcal{E}_k^S$ and $l \in A$. Since $i > i_{k+1}$ and $i_{k+1} \in \mathcal{E}^{\mathcal{C}}$, $l \geq k+1$.

$$r_i - r_{i_k} = r_{i_{l-1}} - r_{i_k}$$

Using Proposition 4, where the response times of stable employees are monotonically non-decreasing, to get $r_{i_{l-1}} \geq r_{i_{k+1-1}} = r_{i_k}$,

$$\begin{aligned} r_{i_{l-1}} - r_{i_k} &\geq r_{i_{k+1-1}} - r_{i_k} \\ r_{i_{l-1}} - r_{i_k} &\geq 0 \end{aligned}$$

Hence $r_i \geq r_{i_k}$, and since $s_i \geq s_{i_k}$ due to seniority, $e_i \geq e_{i_k}$. □

Corollary 1. $\forall k \in A$, $\forall i \in \mathcal{E}$ *if employee* i_k *bumps employee* i, *then* $i \in \mathcal{E}_k^S$.

Proof. This is straightforward from Proposition 3 and Proposition 5 and by construction in Equation (7) from the main article, where $r_{i_k} > r_i$, such that $k \in A$, $i \in \mathcal{E}_k^S$. □

Lemma 1. $\forall S^* \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, $\forall k \in A, b_{i_k} = a_k$ *or* 0.

Proof. Let $S^* = [s_i, e_i]_{i\in\mathcal{E}} \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, and let $B^* = B(S^*)$ be the optimal number of potential bumps and $C(S^*)$ represent the makespan of the notification schedule. Note we use $B^*, B(S^*)$ interchangeably. Consider the employee block $\mathcal{E}_k^S$, along with critical employee i_k, for $k \in A$. The following two cases are possible:
Case 1: $\forall i$ such that $i \in \mathcal{E}_k^S$, $e_i < e_{i_k}$. This directly implies $b_{i_k} = a_k$.
Case 2: $\exists i'$ such that $e_{i'} \geq e_{i_k}$. A new schedule $S' = [s'_i, e'_i]_{i\in\mathcal{E}}$ can be formed by redefining end times for employees of the $\mathcal{E}_k^S$ as follows

$$e'_i = \begin{cases} e_{i_k} & \forall i \in \{i : e_i < e_{i_k}, i \in \mathcal{E}_k^S\}, \\ e_i & \text{else} \end{cases}$$

This new schedule S' is a feasible schedule since $\forall i \in \{i : e_i < e_{i_k}, i \in \mathcal{E}_k^S\}, r_i = r_{i_{k-1}}$ and $e'_i = e_{i_k}$ implies $s'_i = s_{i_k} + a_k$. Also $\forall i \in \{i : e_i \geq e_{i_k}, i \in \mathcal{E}_k^S\}$,

$$\begin{aligned} s'_i &= s_i \\ &\geq s_{i_k} + r_{i_k} - r_{i_{k-1}} \\ &\geq s_{i_k} + a_k \end{aligned}$$

In general, $\forall i \in \mathcal{E}_k^S$, $s'_i \geq \max\{s_{i_k} + a_k, s_{i-1}\}$. Hence, it is easy to see that for the schedule S', the potential bumps for i_k, $b'_{i_k} = 0$. Hence $\forall k' \in A : k' \neq k$,

$$b'_{i'_k} = b_{i_k} \tag{A.7}$$

This equation follows from Proposition 6 and Corollary 1. Also from Proposition 5, $b_i = b'_i = 0, \forall i \in \mathcal{E}^S$. Hence for schedule S', $b'_{i_k} = 0$ since $\forall i \in \mathcal{E}_k^S, e'_i \geq e'_{i_k}$. Since S^* is optimal, $B(S^*) = B(S')$ and from equation Equation (A.7) $b_{i_k} = 0$. □

Corollary 2. *$\forall S \in \mathcal{F}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, the total potential bumps $B(S) = \sum\limits_{k\in A} b_{i_k}$.*

This corollary directly follows from Lemma 1.

Lemma 2. *$S(A) \in \mathcal{F}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$.*

Proof. For $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$, any feasible schedule must satisfy the seniority constraint and the horizon constraint. Any block schedule $S(A')$ already satisfies the seniority constraint. For the horizon constraint, the following equation must hold:

$$H = C_0^* - W \geq C_0^* - \sum_{j\in A'} a_j = C(S(A))$$

Since $H \geq C(S(A))$, $S(A)$ is a feasible schedule.

□

Lemma 3. *$\exists A' \subseteq A$ such that $S(A') \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$.*

Proof. Let $S^* = [s_i^*, e_i^*]_{(i\in\mathcal{E})}$ be an optimal schedule with total potential bumps $B(S^*)$ and makespan $C(S^*) \leq H$ for $NTP\Big([a_j]_{j\in\mathcal{A}}, W\Big)$. From Lemma 2, a feasible solution always exists. We are now going to show $\forall S^* \in \mathcal{O}_{NTP}\Big([a_j]_{j\in\mathcal{A}}, W\Big)$ there exists a block schedule $S(A') = [s_i, e_i]_{(i\in\mathcal{E})}$ where $A' \subseteq A$ such that $B(S^*) = B(S(A'))$ and $C(S(A')) \leq C(S^*) \leq H$. From Lemma 1, $\forall k \in A, b_{i_k} = a_k$ or 0. One can simply construct $A' = \{k : b_{i_k} = a_k\}$. The

block schedule $S(A')$ has total potential bumps $B(S(A')) = \sum_{j \in A'} a_j = B(S^*)$. Also $\forall k \in A$, the following two cases exists:
Case 1 $k \in A'$, $b_{i_k} = a_k$: For this condition to be true $\forall i \in \mathcal{E}_k^S, e_i < e_{i_k}$. Hence $s_{i_k} \leq s_i \leq s_{i_{k+1}}$.
Case 2 $k \notin A'$, $b_{i_k} = 0$: For this condition to be true $\forall i \in \mathcal{E}_k^S, e_i \geq e_{i_k}$. This implies $s_i + r_i \geq s_{i_k} + r_{i_k}$. Hence it follows that $s_{i_k} + a_k \leq s_i \leq s_{i_{k+1}}$.
Combining the two cases to get:

$$s_{i_N} \geq \sum_{j=1}^{N-1} \mathbf{1}_{j \notin A'} a_j$$

Therefore, for the last employee, $s_M \geq s_{i_N} + \mathbf{1}_{j \notin A'} a_N \geq \sum_{j \notin A'} a_j$. Hence $C(S(A')) = e_M + s_M + r_M \geq \sum_{j \notin A'} a_j + \sum_{j \in A} a_j$ and $S(A')$ is also an optimal schedule. □

Corollary 3. *$\forall S^* \in \mathcal{O}_{NTP}\Big([a_j]_{j \in \mathcal{A}}, W\Big)$, the optimal objective value $B(S^*) \geq W$.*

Proof. From Lemma 3, $\exists A' \subseteq A, S(A) \in \mathcal{O}_{NTP}\Big([a_j]_{j \in \mathcal{A}}, W\Big)$ such that $B(S(A')) = \sum_{j \in A'} a_j$ and $C(S(A')) = C_0^* - \sum_{j \in A'} a_j$.

$$C(S(A')) = C_0^* - B^* \leq H = C_0^* - W$$

Simplifying, $B^* \geq W$. □

Theorem 2. *For $NTP\Big([a_j]_{j \in \mathcal{A}}, W\Big)$, $B(S^*) = W \Leftrightarrow \exists A^* \subseteq A$ such that $\sum_{j \in A^*} a_j = W$.*

Proof.
Case 1 $\exists\ S^* = [(s_i^*, e_i^*)]_{i \in \mathcal{E}}$ such that $B(S^*) = W$: From Lemma 3, one can build an optimal block schedule $S(A^*)$ with $B(S(A^*)) = \sum_{j \in A^*} a_j$. Hence, $W = \sum_{j \in A^*} a_j$.
Case 2 Suppose $\exists A^* \subseteq A$ such that $W = \sum_{j \in A^*} a_j$: Construct a feasible block schedule $S(A^*)$ with $B(S(A^*)) = \sum_{j \in A^*} a_j = W$ and makespan $C(S(A^*)) = C_0^* - \sum_{j \in A^*} a_j = H$. By Corollary 3, $S(A^*)$ is optimal. □

Theorem 3. *$NTP(I) \in$ NP.*

Proof. Given a finite vector $\mathbf{r}$ with duration $r_i \in Z^+$, for each $i \in \mathcal{E} = \{1, .., M\}$, a target horizon H and a target number of potential bumps B^*.

- **Certificate**: A notification schedule $[s_i]_{i \in \mathcal{E}}$, such that $s_i + r_i \leq H, \forall i \in \mathcal{E}$ with total bumps B.

- **Verification**: One can verify $s_i + r_i \leq H$ in linear time with respect to the number of employees. To count the potential bumps, we need to count for a given employee i how many senior employees ($i\prime$) respond after i. Thus, for each employee we would need to check almost $i - 1$ checks, which is again linear in time. The checks for all employees can thus be performed at a maximum $O(n^2)$ time, which is polynomial and thus $NTP(I) \in NP$.

□

Theorem 4. *$NTP(I)$ is* NP-complete.

Proof. This conclusion follows from the fact that Subset-Sum is *NP-complete*. As NTP(I) $\in NP$, by providing a reduction from the Subset-Sum problem to NTP(I), we establish the NP-completeness of NTP(I). □

Theorem 5. *Given any online algorithm, there exists an instance I in which the algorithm suffers the maximum potential bumps or has vacant shifts, and the offline counterpart will have zero potential bumps and zero vacancies for that same instance.*

Proof. Let H be the horizon and $\mathcal{E}$ denote the employee set. Let s_i denote the times when the online algorithm sends the notifications to employee i. The goal is to design a suitable instance (response delays r_i) such that the online algorithm behaves as badly as possible. In other words, we designing an adversary for the online algorithm. As the online algorithm starts sending notifications, two cases can happen

Case 1 $\nexists\, \hat{i}$ such that $\hat{i} = \{\min i : i \in \mathcal{E}, s_i > 0\}$.
Alternately, this means all employees are sent notifications at time 0. In such a case set

$$r_i = H + 1 - i, \quad \forall i \in \mathcal{E} \tag{A.8}$$

The online algorithm will then suffer the maximum potential bumps as employees will respond in the reverse order of seniority ($e_i = H + 1 - i$). The offline algorithm, however, can easily prevent bumps by simply ensuring that all employees respond at the same time. Let s_i^* denote the offline solution.

$$s_i^* = H - r_i, \quad \forall i \in \mathcal{E} \tag{A.9}$$

This will ensure all employees respond at the horizon.
Case 2 $\exists\, \hat{i}$ such that $\hat{i} = \{\min i : i \in \mathcal{E}, s_i > 0\}$.

$\hat{i}$ represents the first employee not notified at time = 0. In such a case, we can set

$$r_i = H \tag{A.10}$$

Since $e_i = s_i + H$, it would mean all junior employees to $\hat{i}$, and as well as $\hat{i}$ would respond after the horizon. Hence, the moment the online algorithm waits for even one unit, we make sure that, irrespective of the notification times for the next employees, we set their response times such that there are some vacant shifts. It is easy to see that the offline solution will have no bumps due to $r_i = r_j, \forall i, j \in \mathcal{E}$. Additionally, we can also ensure that an offline schedule has zero vacancies if

$$s_j^* = 0 \tag{A.11}$$

This way, you can ensure that all employees respond at the horizon and that there are zero vacancies or bumps. We also respect seniority in the notifying order.

A.3 Intial Results using DAgger based algorithm

The initial phases of the project concentrated on the use of DAgger for the SDPs. However, the issue with directly using DAgger for dynamic SDPs is that getting an expert is highly expensive, even for offline training. Hence, one would need to use approximate or pseudo-experts to imitate. These experts must be easier to obtain as well offer quality demonstrations to learn from. Hence, the initial goal was to learn whether one could design suitable pseudo-experts.

A.3.1 Algorithm

For the NTP, we rely on a pseudo-expert constructed using offline solutions to replace the actual expert. This pseudo-expert is still expensive to compute in real-time, but is viable for offline training. We derive our main idea for the pseudo-expert from the ONP policy in Chapter 4. This policy is derived by first solving several instances offline using (MIP_{NTP2}). Next, the optimal solutions, the number of notifications to be sent at each epoch, are aggregated using an aggregator function. The aggregator functions used are simple functions like mean and percentiles. We run DAgger algorithms for each of the aggregator functions, and the best-performing one is chosen among them. The algorithm used to generate trajectories and train models is given by Algorithm 15. This algorithm represents an extension of the naive version of DAgger.

Each of the n scenarios is then solved offline using MIP_{NTP2} to obtain a vector of optimal actions $\bar{a}_k$ for a given decision epoch. This is also referred to as Hindsight optimization. Note MIP_{NTP2} gives the optimal number of notifications to send at each epoch. However, with this information, one can then easily calculate the optimal wait time between notifications. This effectively provides us with n distinct expert actions for each scenario. Subsequently, an aggregator function is applied to this vector to define a singular target $\hat{a}_k$. The state variable, denoted by x_k, encapsulates comprehensive information. Our objective is to distil this wealth of data into a fixed set of relevant features $\mathcal{F}$, elaborated upon in the results section. For each decision epoch within the planning horizon, a training example is constructed, represented as $(F_k, \hat{a}_k)$, where F_k signifies the input features and $\hat{a}_k$ denotes the label indicating the expert action at epoch k. Note that we skip the iteration subscript used in the algorithm to denote the feature vector and the target. Ultimately, the best policy π_i is selected based on a set of validation instances.

Algorithm 15: DAgger algorithm to generate training examples and train policy

Function `HindOptDAgger()`:
  Initialize $D \leftarrow \phi, \pi_0 \leftarrow ONP$
  for $i \in \{0, \ldots, itr\}$ **do**
    for $j \in \{0, \ldots, I\}$ **do**
      Sample K-step trajectories using π_i ;
      for $k \in \{1, \ldots, K\}$ **do**
        Generate n scenarios ;
        Get optimal actions for each scenario $\boldsymbol{a}_k^{ij} = [a_{kl}^{ij}]_{l \in \{1,\ldots,n\}}$ using MIP_{NTP2};
        Get the input feature vector F_k^{ij};
        Get target action $\hat{a}_k^{ij} = Agg(\boldsymbol{a}_k^{ij})$;
      end
      Collect $D_{ij} = \{(F_k^{ij}, \hat{a}_k^{ij})\}$ dataset of visited states and target actions;
      $D \leftarrow D \cup D_{ij}$;
    end
    Train policy π_{i+1} on D;
  end
end

We also simulate the currently employed policy by our industrial partner, a linear Notify-and-wait (NAW) policy that notifies a fixed number of employees $\eta = 5$ every $\tau = 1$ time unit. Additionally, we attempted to improve this heuristic policy (TNAW) through brute force search on its parameters, simulating and testing all possible combinations. Furthermore, a Notify-all (NA) policy was implemented, which notifies all employees at the beginning as

soon as possible, respecting the maximum notifications per decision epoch constraint. This policy was included to emphasize its adverse impact on bumps. Finally, we also compare our results with ONP from Chapter 4.

For the DAgger algorithm, we employ gradient-boosted decision trees (GBDT) with XGBoost to build a learning model. Mean and percentile are used as aggregator functions. The percentile functions utilized are spaced 5 units apart, namely $\{45, 50, 55\}$. Such policies are referred to as percentile DAgger policies. Our regression model predicts the waiting time before sending the next notification as a continuous value, which is then rounded down to the nearest integer to obtain a discrete number. We use the same features as described in Table 6.3 in Chapter 5.

A.3.2 Results across aggregator functions

Table A.1 displays the average cost across 500 test instances obtained for different policies. Algorithm 15 is executed with various aggregator functions, gathering different operating policies while utilizing $n = 9$ scenarios to define the target. The cost function remains consistent with the objective defined in MIP_{NTP2}. The primary observation indicates that the DAgger policy DT_{45} slightly outperforms the ONP in terms of cost. While DT_{50}, exhibits nearly identical performance in cost compared to the ONP.

However, DT_{μ} significantly underperforms compared to DAgger policies. This observation is noteworthy, indicating that an average offline decision derived from a set of contextual scenarios is notably inadequate for the stochastic problem. Both the current policy, NAW and the NA policy demonstrate considerably higher costs. Even a tuned version (TNAW) of the current policy exhibits notably weaker performance than the percentile DAgger policies.

Figure A.1 illustrates the bumps and vacancies for each policy. It is noticeable that as the number of bumps increases within the various policies, the number of vacancies decreases. The current heuristic (NAW) and Notify-all (NA) policies perform poorly in terms of bumps to ensure a complete shift allocation. Compared to the ONP, DT_{45} results in fewer vacancies, which may be of greater interest to decision-makers.

It is also important to highlight the substantial value of stochastic information in this problem. Solving 1000 instances offline under complete information, where all uncertainty is known at the start of the planning period, yielded an average of about 5.5 bumps with 0 average vacancies. Thus, if all response delays are known at the start, nearly all bumps can be eliminated, and all shifts can be scheduled.

	Policies							
	NA	**NAW**	**TNAW**	**ONP**	DT_{μ}	DT_{45}	DT_{50}	DT_{55}
Cost	668.8	418.0	122.9	103.7	143.06	**99.48**	103.4	114.8

Table A.1 Cost for different policies

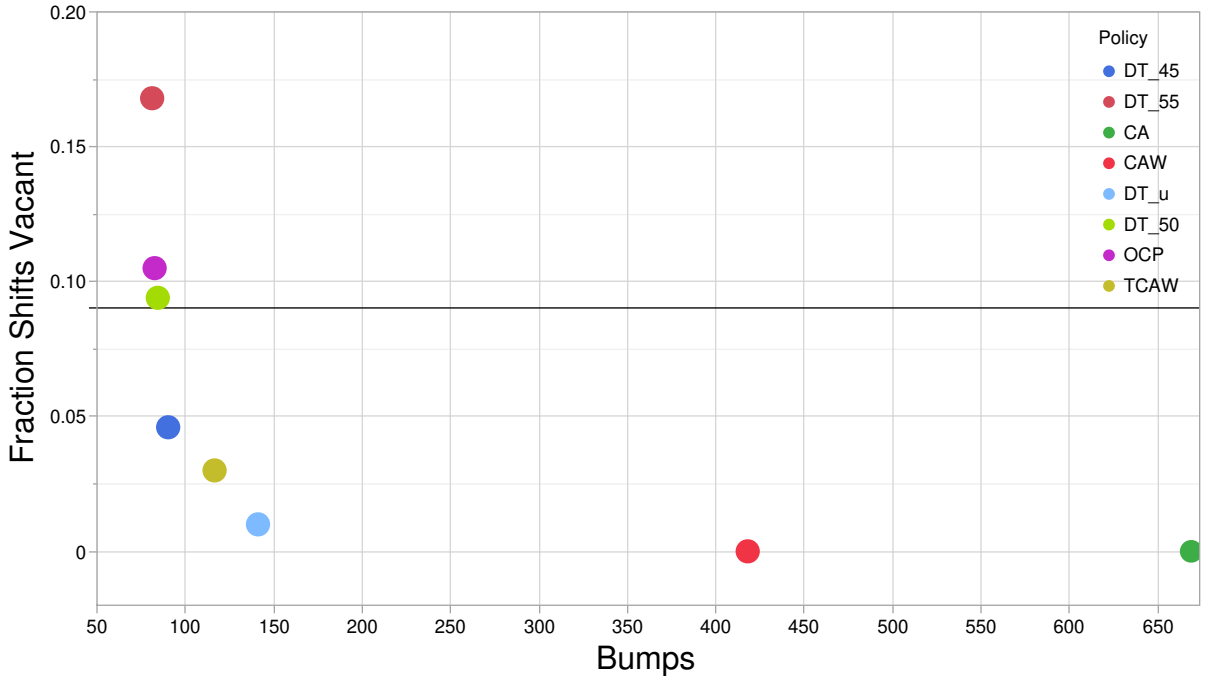


Figure A.1 Bumps vs Vacancy for all policies

A.3.3 Effect of number of scenarios

Next, we also study the effect of the number of scenarios (n) on the learning. We try 3 different settings for $n = \{1, 5, 9\}$ and present the results obtained from the best policies. We see that the performance in five scenarios is not as bad as compared to 9 scenarios. However, having only 1 scenario means there may be vacant shifts. As noted in the previous subsection, the value of stochastic information is high, and as a result, having 1 scenario completely fails to ensure sufficient shift occupancy. The results show that the greater the number of scenarios, the better the performance in terms of cost.

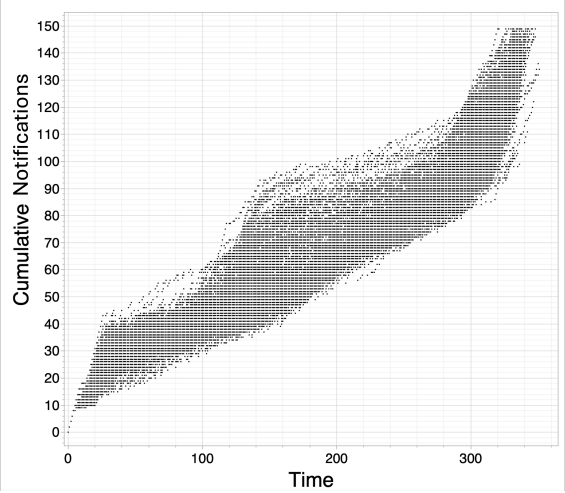
A.3.4 Notifying Profiles

Figure A.2a illustrates the distribution of the cumulative number of notifications sent at each discrete point in time. Upon comparison with Figure 6.1, it becomes evident that the distributions are generally similar. The distribution appears tightly packed, indicating

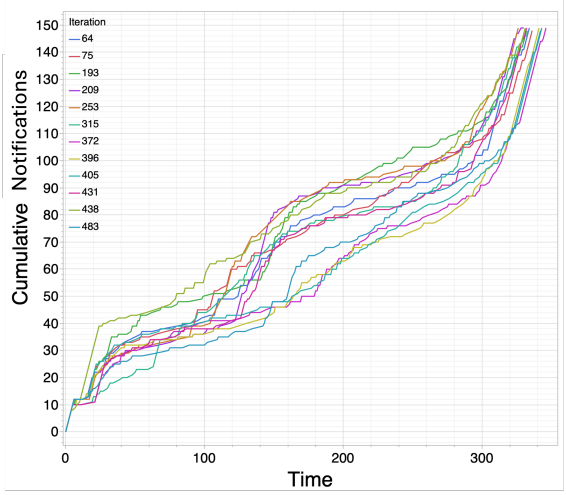
Table A.2 Cost, Bumps, and Vacancy across different scenarios

	Scenarios		
	1	**5**	**9**
Cost	275.9	119.5	99.4
Bumps	65.9	106.5	90.2
Vacancy	1.05	0.065	0.046

the absence of divergent trajectories. Furthermore, Figure A.2b displays the same profiles across 12 episodes featuring empty shifts out of the 500 test scenarios. The trajectory reveals that the model indeed attempted to notify all 150 employees. However, due to inadequate responses within the allotted time, some shifts remained vacant.



(a) Distribution of cumulative Notifications for 500 simulations



(b) Notification profile over 12 simulation runs with empty shifts

Figure A.2 Notification profiles over the horizon for DT_{45}

A.3.5 Feature importance

The graph in Figure A.3 displays the feature importance in the model.

A.4 Using multiple experts to control trajectories

In the previous subsection, we examined the use of different imperfect experts in the learning process. The imperfection in these experts stems from their reliance on scenario sampling from available data. Since the target action is determined based on sampled scenarios, the presence of an outlier scenario can lead to an outlier action for a given state. This deviation

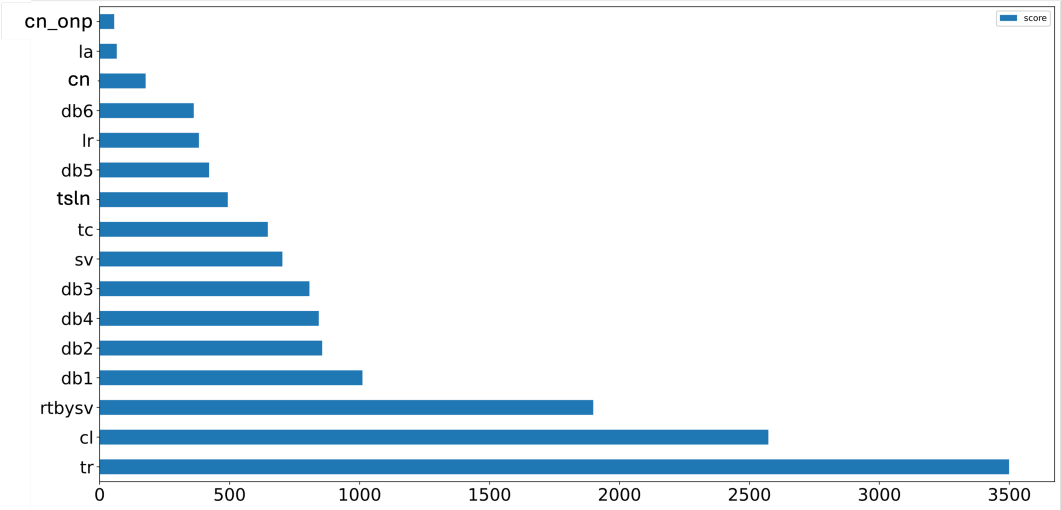


Figure A.3 Feature Importance

not only distorts the observed trajectory but may also introduce states that lie outside the true expert's distribution. Consequently, the learning process may be hindered, as the policy must adapt to handling these previously unseen or unrealistic states rather than refining its decisions within the intended distribution.

To mitigate this issue, we introduce a **behavior policy** as a secondary expert to guide the algorithm away from unsafe or anomalous actions. While this behavior policy is not necessarily optimal, it is designed to perform reliably under standard conditions. Its primary function is to monitor the actions suggested by the primary expert and intervene when necessary. If the primary expert recommends an action that significantly deviates from expected behavior, the secondary expert provides an alternative, more stable action. This mechanism enhances robustness by reducing the impact of outlier actions and ensuring safer decision-making in uncertain environments.

To formalize this approach, we introduce a modified decision rule, referred to as the **Expert-Safe Decision Rule**, which is detailed in Algorithm 16. The key distinction from the Vanilla DAgger decision rule, highlighted in blue, lies in how the expert's guidance is utilized. Specifically, we compare the imperfect primary expert action, denoted as a_k^*, against the secondary expert action, a_k^b. A discrepancy measure f is then computed to quantify the difference between these two actions in the current state. If the discrepancy exceeds a predefined threshold τ, the behavior action is executed; otherwise, we default to the expert action, assuming it remains safe. The discrepancy measure f can be any function that evaluates the difference between the two actions. A simple example is the absolute difference $|a_k^* - a_k^b|$, though more sophisticated measures could incorporate additional state-dependent factors to

refine the decision-making process.

In the context of the **Notification Timing Problem (NTP)**, the behavior action a_k^b can be represented by the **Offline Notification Policy (ONP)**—a well-established policy with a proven track record of reliability, as discussed in Chapter 4. By leveraging this behavior policy as a secondary expert, the algorithm promotes more stable trajectories while gradually integrating the learned model's decisions. The key difference of the Expert-Safe DR as compared to Safe DR in Algorithm 6 is we do check on the expert action and not the model action.

Algorithm 16: EXPERT-SAFE DAgger Decision Rule

Function `Expert-Safe DAgger`$(x_k, i, \beta_0, \lambda)$**:**
  $a_k \leftarrow \pi_i(x_k)$;
  $a_k^b \leftarrow \pi_i^b(x_k)$;
  $a_k^* \leftarrow \pi^a(x_k)$;
  $\beta_i \leftarrow \lambda^i \beta_0$;
  $z \sim \text{Uniform}(0, 1)$;
  if $f(x_k, a_k^*, a_k^b) \geq \tau$ **then**
    $a_k^* \leftarrow a_k^b$;
  end
  if $z \leq \beta_i$ **then**
    return a_k^*;
  else
    return a_k;
  end
end

Table A.3 presents the algorithm within the taxonomy framework, highlighting its key distinctions from the Vanilla DAgger algorithm discussed in the previous subsection. The main differences lie in the incorporation of multiple experts. Unlike Vanilla DAgger, which relies solely on a single expert, this approach introduces a behavior policy as a secondary expert. This secondary expert is a heuristic behavior policy. This behavior policy provides alternative actions when the primary expert suggests an outlier and is also meant to play a crucial role in guiding the trajectory, ensuring more stable and reliable learning dynamics.

- **Expert Type**: The experts used are a combination of the optimization-based expert and the heuristic.
- **Optimality**: Since we use a combination of experts, the action suggested will be optimal if the optimization model was used; else it would be an approximation if the heuristic was used.

- **Expert Usage**: ADE differs in how it utilizes expert guidance. Instead of relying on a single deterministic expert, it acts as a proxy expert by querying the DE multiple times and selecting a target action based on an aggregation criterion. This makes ADE more robust to variations in deterministic expert decisions and helps smooth out inconsistencies. The FIE, DE, and 2SSE all are used just once to provide a demonstration.
- **Number of Experts**: All algorithms use multiple (2) distinct experts.

Table A.3 Expert-Safe DAgger Algorithm Classification

	Expert Used		
	DE + Heuristic	**2SSE + Heuristic**	**ADE + Heuristic**
Policy Type	Dynamic	Dynamic	Dynamic
Policy Variable	Direct Decision	Direct Decision	Direct Decision
Optimality	Sub-Optimal/Approximate	Sub-Optimal/Approximate	Approximate
Information Used	Limited	Limited	Limited
Reproducibility	Inconsistent	Inconsistent	Inconsistent
Expert Usage	Single Complete	Single Complete	Multiple/Single Complete
Number of Experts	Mutiple	Mutiple	Mutiple
Learning Control	Regressive	Regressive	Regressive
Expert Evolution	Static	Static	Static