11-2022

Reinforcement learning approach to coordinate real-world multi-agent dynamic routing and scheduling

JOE WALDY
Singapore Management University, waldy.joe.2018@phdcs.smu.edu.sg

REINFORCEMENT LEARNING APPROACH TO COORDINATE REAL-WORLD MULTI-AGENT DYNAMIC ROUTING AND SCHEDULING

JOE WALDY

SINGAPORE MANAGEMENT UNIVERSITY
2022

Reinforcement Learning Approach to Coordinate Real-World Multi-Agent Dynamic Routing and Scheduling

Joe Waldy

Submitted to School of Computing and Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee:

LAU Hoong Chuin (Supervisor / Chair)
Professor of Computer Science
Singapore Management University

CHENG Shih-Fen
Associate Professor of Computer Science
Singapore Management University

MAI Anh Tien
Assistant Professor of Information Systems
Singapore Management University

Arunesh SINHA
Assistant Professor of Management Science & Information Systems
Rutgers Business School

Singapore Management University
2022

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the dissertation.

This PhD dissertation has also not been submitted for any degree in any university previously.

Joe Waldy

7 November 2022

Abstract

Reinforcement Learning Approach to Coordinate Real-World Multi-Agent Dynamic Routing and Scheduling

by

Joe Waldy

Doctor of Philosophy in Computer Science

Singapore Management University

In this thesis, we study new variants of routing and scheduling problems motivated by real-world problems from the urban logistics and law enforcement domains. In particular, we focus on two key aspects: *dynamic* and *multi-agent*. While routing problems such as the Vehicle Routing Problem (VRP) is well-studied in the Operations Research (OR) community, we know that in real-world route planning today, initially-planned route plans and schedules may be disrupted by dynamically-occurring events. In addition, routing and scheduling plans cannot be done in silos due to the presence of other agents which may be independent and self-interested. These requirements create great opportunities for synergized efforts between OR and Artificial Intelligence (AI), particularly Reinforcement Learning (RL) and Distributed AI such as Multi-Agent Systems (MAS).

The fundamental research question for dynamic routing and scheduling is: *How to make optimal decision within a short period of time?*. Routing and scheduling decisions are complex because they are multi-dimensional; consisting of spatial and temporal components. Meanwhile, each occurrence of dynamic event requires an *event-handling* action and a *re-planning* action such as assignment/dispatch and rerouting/rescheduling actions respectively. Current approaches are either time consuming or not sample efficient. Meanwhile, to address complex action, most current RL approaches either decompose the problem or action into multiple stages. In this thesis, we propose an RL-based approach that combines Value Function Approximation (VFA) and routing/scheduling heuristic to learn *event-handling* and *re-planning* policies jointly without the need for any decomposition step. We show that our approach is faster than sampling-based approaches and more sample efficient

than current offline methods for moderately-sized problem instances. We also show experimentally that our joint learning approach outperforms the commonly-used two-stage approach especially when the problem scenario becomes more complex.

Multi-agent routing and scheduling problems in real-world context go beyond the classical definition of multi-agent in academic literature which usually takes the form of multiple vehicles or multiple machines. In this thesis, we refer an *agent* as a independent, higher-order decision-making entity that is capable of executing complex action and usually consists of multiple sub-agents such as Logistics Service Providers (LSPs) where each LSP consists of multiple vehicles. Existing works on multi-agent VRP assume collaboration amongst agents. However, in real-world context, agents may not necessarily be collaborative. One such instance is a problem of coordinating Business-to-Business (B2B) pickup-delivery operations involving multiple LSPs. To address this gap, we formulate such problem as a strategic game and propose a scalable, decentralized, coordinated planning approach based on iterative best response to coordinate multi-agent routing and scheduling. Our proposed approach is able to ensure that there are enough incentives for agents to adopt a coordinated plan rather than planning independently. Our approach is also scalable as it decomposes a multi-agent problem into multiple single-agent problems allowing existing single-agent planning algorithms to be applied to a smaller problem.

Most current Multi-Agent RL (MARL) approaches solve dynamic routing and scheduling problems in which an agent is still defined as low-order entity such as vehicle or machine. Moreover, there is no prior work that addresses dynamic routing and scheduling problems where there exist multiple independent higher-order agents capable of making complex decision directly without decomposing the problem or the action. Therefore, in the final contribution of this thesis, we present a pioneering effort on a cooperative MARL approach to solve multi-agent dynamic routing and scheduling problem directly without any decomposition step. This contribution extends our earlier proposed VFA method to address multi-agent setting and incorporates an iterative best response procedure as a decentralized optimization heuristic and an explicit coordination mechanism. We evaluate our approach against a realistic multi-agent dynamic police patrol problem and through a series of ablation studies, ascertain the effectiveness of our proposed learning and coordination mechanisms.

This thesis opens up many opportunities for future research, some of which are presented in the concluding chapter, specifically those that represent aspects of RL approach that are peculiar to real-world problem settings.

Contents

Acknowledgments

First and foremost, I would like to thank God for His Grace and faithfulness that have been seen throughout my life especially in my academic journey and my career. Choosing to pursue a PhD has never crossed my mind nor something that I have ever dreamt of doing. He somehow paved the way for me and I can see how He has orchestrated everything for me to reach where I am right now.

I would like to take this opportunity to appreciate and thank my supervisor, Prof. Lau Hoong Chuin who has been a great and supportive supervisor, mentor, co-worker and friend. He was the one who picked up my PhD application and took the initiative to contact me. We share many common research interests and even life values. He not only provided me with precious advice pertaining to research work but he did so in such an encouraging and nurturing way. I am so thankful that he took the first step to reach out to me and the rest is history.

I would like to thank the members of my Dissertation Committee for availing their time to evaluate and provide constructive feedback to my research work. Prof. Shih-Fen, for his invaluable advice especially during the initial part of my PhD journey, Prof. Arunesh, for deciding to stay on in the committee despite having to relocate to somewhere faraway and Prof. Tien, for agreeing to evaluate my work even at such a late stage of my PhD candidacy.

I would also like to thank my fellow classmates, my seniors and juniors for the friendship, guidance and the comforting reminder that we are not alone in this journey. Their names are not mentioned here but they would know who they are.

Last but not least, I would like to thank my wife, Faith for her ever-present support, words of wisdom and encouragement. When I doubt myself and feel inadequate, she always believes in me and spurs me on. Never would I have imagined completing my PhD programme with the arrivals of not only one but two newborns! I would never be able to do so without her. She is the one truly deserving of this PhD.

To my Ebenezer, the Lord who has helped me thus far,
To my dearest wife, Faith who is the love of my life,
To my lovely children, Elise and Nate who are God's
precious gifts to me.

List of Figures

List of Tables

List of Algorithms

Acronyms

COPs Combinatorial Optimization Problems.

DPRP Dynamic Bi-Objective Police Patrol Dispatching and Rescheduling Problem.

DRLSA Deep Reinforcement Learning with Simulated Annealing.

DVRP Dynamic Vehicle Routing Problem.

FIP Finite Improvement Property.

LSP Logistics Service Provider.

MADPRP Multi-Agent DPRP.

MAP Multi-Agent Planning.

MARL Multi-Agent RL.

MAS Multi-Agent System.

MDP Markov Decision Process.

ML-VRPLC VRP with Pickup and Delivery, Time Windows and Location Congestion with Multiple LSPs.

OR Operations Research.

RL Reinforcement Learning.

TD Temporal-Difference.

VFA Value Function Approximation.

VRP Vehicle Routing Problem.

Chapter 1

Introduction

Routing and scheduling problems are classical Combinatorial Optimization Problems (COPs) that are well-studied and well-researched in the Operations Research (OR) community [51]. New and more complex variants of these problems have increasingly been introduced and studied in the literature. These variants are usually driven and inspired by real-world applications [20]. Meanwhile, the advancement in the field of Artificial Intelligence (AI) particularly in the sub-fields of Machine Learning (ML) such as Deep Learning (DL) and Reinforcement Learning (RL), and Distributed AI such as Multi-Agent System (MAS), brings about greater opportunities for synergized effort between the two fields of study (AI and OR) to address real-world routing and scheduling problems that are increasingly more challenging because of the following two key reasons:

- *Dynamic.* In most classical routing and scheduling problems, inputs to the problems are known beforehand and they can be either deterministic or stochastic. In the dynamic version of those problems, inputs to the problem may change over time and they are typically exogenous. For example, in the context of Dynamic Vehicle Routing Problem (DVRP), the exogenous changes include additional or cancellation requests from customers (stochastic customers), changes to the traffic conditions (stochastic travel times) and changes to the customers' demand (stochastic demand) [102]. The proliferation of data and the advancement of computing power have enabled researchers in the last couple of decades to study and develop state-of-the-art methodologies to solve DVRP [98]. The rise of online shopping and food delivery also heighten the research interest in the domain of dynamic routing and scheduling (see [123, 127]). In fact, the popular notion of *same-day delivery* that many of us

are familiar with today is essentially a DVRP.

- *Multi-Agent.* The notion of multi-agent here does not refer to the classical notion where an agent is usually defined as a lower-order entity like a vehicle, machine or a person. With the rise of e-commerce and the increased complexity of urban logistics, solving Vehicle Routing Problem (VRP) in real-world environment involves coordinating multiple logistics entities [41] and even synchronization of routing with scheduling due to shared resource constraints such as limited docking capacity and availability of equipment or personnel (see [37, 64, 65]). In other words, routing and scheduling decisions cannot be done in silos due to the presence of multiple independent and sometimes self-interested agents.

Motivated by the increasingly complex nature of the operational problems highlighted above, this thesis aims to develop new methodologies drawing from relevant techniques in AI (RL and MAS more precisely) and OR to address the gaps in current works in solving *multi-agent* and/or *dynamic* routing and scheduling problems.

1.1 Motivating Domains

The research work in this thesis is specifically motivated by two real-world application domains namely urban logistics and law enforcement. Routing and scheduling in the urban logistics context are not new and are well-studied in the literature. However, in this thesis, we address problem variants that are either new and/or have potential for AI applications. Meanwhile, routing and scheduling in the context of law enforcement operation (specifically police patrol) may not be as well-studied as those in the logistics context but it presents a huge potential for further research given that police patrol is done on a daily basis and optimal routing and scheduling of police patrol is necessary in view of scarce police resources [36].

1.1.1 Urban Logistics

This thesis looks into the two aspects of routing and scheduling in urban logistics that have huge potential for further research namely *dynamic* and *multi-agent*.

Dynamic. Pick-up and delivery tasks in real-life may be subjected to changes in the environment such as additional or cancellation of requests, changes to the traffic conditions or changes to the customers' demand. These changes or commonly referred to as dynamic events, may happen to disrupt the initially-planned routes and schedules. Thus, there is a need to make rerouting or rescheduling decision as and when these dynamic events happen and decision needs to be made rather quickly in real-time. The most direct approach to solve this problem is to reoptimize the routes or schedules with the new information. This is akin to solving a static variant of the problem again and again whenever there are new information available. However, such an approach is myopic because it does not take into consideration future dynamic events and most importantly, it may be time-consuming. The research challenge that this thesis is addressing will be whether a policy governing such rerouting or rescheduling decision can be learnt offline and be applied online. This is exactly what ML and RL are all about; making prediction or decision by learning from data beforehand.

Multi-Agent. Route planning and scheduling in urban logistics may be subjected to the presence of other logistics entities also known as Logistic Service Providers (LSPs) in the environment. One such scenario will be a pick-up and delivery problem where there are limited shared resources in each of the locations. For example, in urban, congested cities where space is scarce, each pick-up or delivery location may have limited parking space or loading/unloading bays. Congestion may take place if each LSP adopts selfish, uncoordinated planning. This thesis addresses such a problem of coordinating routing and scheduling in the presence of multiple, independent entities or agents vying for shared limited resources.

1.1.2 Law Enforcement

Similar to urban logistics, this thesis addresses both *dynamic* and *multi-agent* aspects of police patrol problem.

Dynamic. In policing, occurrences of unexpected incidents often disrupt the execution of an existing plan. Police patrol agent needs to be dispatched and its patrol schedule needs to be adapted to respond to dynamically-occurring incidents

while still fulfilling the two often conflicting objectives of projecting police presence (*proactive* patrol) and responding to incidents in a timely manner (*reactive* patrol). In addition, such complex decisions need to be made rather quickly and sometimes instantaneously. The key research challenge that this thesis addresses is whether ML and RL can be exploited to compute such decisions quickly and intelligently in order to inform human decision-makers.

Multi-Agent. In most countries, police operations are usually divided into several sectors or districts or jurisdictions to balance the workload [107]. However, due to limited police resources and increased workload in terms of number of incidents, there may be a need to coordinate the schedules of the different police entities so as to ensure that the common goal of maintaining law and order is being achieved. The most straightforward approach to address this problem will be to have a centralized agent which plans the schedules of all police entities. However, this approach is not practical given that scalability can be an issue and it also goes against the established operation and reporting structures that are already in place. The research question will be how to coordinate such an operation involving multiple independent, possibly selfish agents even in the presence of dynamically-occurring incidents and time pressure.

1.2 Contributions

The main contributions in this thesis are two-fold. Firstly, we define and formulate new variants of routing and scheduling problems which incorporate real-world operational characteristics. Secondly, we propose new solution approaches that incorporate techniques from outside the OR domain specifically RL and Multi-Agent Planning (MAP) to address those new variants of the problems which typically are *dynamic* and/or *multi-agent* in nature.

Figure 1.1 provides an overview of the contributions of this thesis and how each contribution fits into the overall theme of coordinating multi-agent dynamic routing and scheduling. References to the relevant chapters and sections are provided for a clearer view on the structure and flow of this thesis.

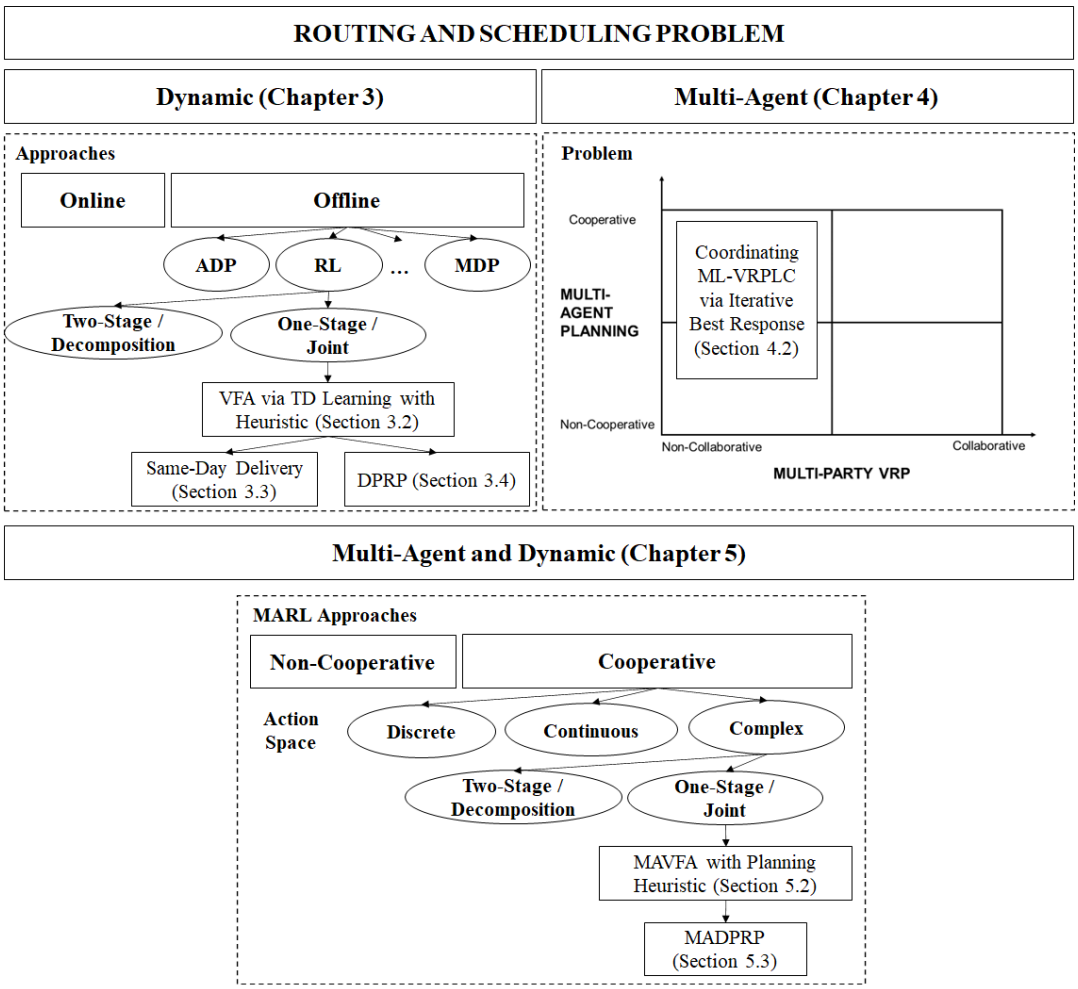


Figure 1.1: An overview of the contributions and structure of this thesis.

1.2.1 New Problem Variants

Urban Logistics. We define a new variant of VRP called VRP with Pickup and Delivery, Time Windows and Location Congestion with Multiple LSPs (or ML-VRPLC in short) and formulate the problem as an n-player strategic game [57]. This new problem variant is motivated by real-world problem of coordinating Business-to-Business (B2B) pickup-delivery operations to and from commercial or retail locations involving multiple LSPs with limited parking bays at each of the location.

Law Enforcement. We define a new variant of DVRP with stochastic customers that incorporates elements of university time-tabling problem called Dynamic Bi-

Objective Police Patrol Dispatching and Rescheduling Problem (DPRP) and formulate it as a route-based Markov Decision Process (MDP) [58]. We extend DPRP to a multi-agent setting involving multiple police entities which can be represented by different police sectors or jurisdictions and refer this problem as Multi-Agent DPRP or MADPRP in short.

1.2.2 New Solution Approaches

Addressing Dynamicity. We propose a new solution approach that combines Deep RL (specifically neural networks-based Temporal-Difference (TD) learning with experience replay) to approximate value function and a routing/scheduling heuristic to solve dynamic routing and scheduling problems. The proposed solution called Value Function Approximation (VFA) via TD Learning with Heuristic, learns the assignment/dispatch and rerouting/rescheduling policies jointly. This is in contrast with many existing approaches in the literature that adopt a two-stage approach (first stage to address the assignment/dispatch decision and second stage to address the rerouting/rescheduling decision). This proposed offline approach allows decisions to be computed quickly and almost instantaneously during run-time unlike many OR approaches which are sampling-based. We apply this approach to solve two problems namely DVRP with stochastic customers [56] and DPRP [58]. There are several works in the literature that build upon our proposed solution approach (see [14, 92, 114]). In addition, this research contribution represents one of the first few works in the literature that leverages on RL to address DVRP with route constraints and full state space [50].

Addressing Multi-Agent. We propose a scalable, decentralized, coordinated planning approach based on iterative best response to solve the above-mentioned ML-VRPLC [57]. We formulate and show that this problem is a *finite ordinal potential game* with Finite Improvement Property (FIP). This approach iteratively improves on a initial joint schedule by exploring multiple improvement paths through best response procedure until an approximate equilibrium is reached. In other words, we represent each LSP as a player and its schedule as a strategy, and through an iterative best response procedure, each LSP adjusts its own strategy until no LSP is

able to improve its own payoff.

Addressing Both Dynamicity and Multi-Agent. We extend our proposed VFA approach to address multi-agent setting. We propose a Multi-Agent RL (MARL) approach that combines Multi-Agent Value Function Approximation (MAVFA) with planning heuristic to solve multi-agent dynamic routing and scheduling problems directly without the need to decompose the action or the problem into multiple stages. In our proposed approach, the learned value function is utilized by the heuristic to search for better rerouting or rescheduling decision. Our proposed approach incorporates iterative best response procedure to serve as a scalable, decentralized optimization heuristic and an explicit coordination mechanism for a more coordinated decision-making amongst multiple agents. We evaluate our approach on an MADPRP and show experimentally that our approach outperforms the commonly used two-stage approach and through a series of ablation studies, ascertain the effectiveness of our proposed learning and coordination mechanisms

1.3 Structure of the Dissertation

This thesis is organized as follows:

- In Chapter 2, we present the background information and related works pertaining to dynamic routing and scheduling problems and RL approaches to solve such a problem. In this chapter, we also provide more detailed descriptions of the motivating problems and their related works.
- In Chapter 3, we propose a new RL-based approach to solve dynamic routing and scheduling problems and evaluate our proposed approach against two application domains: urban logistics and law enforcement. The research work in this chapter has been published in [56, 58].
- In Chapter 4, we study a new problem variant that involves coordinating the routing and scheduling of multiple agents in view of shared limited resources. We propose a scalable, decentralized coordinated planning approach based on iterative best response. This work has been published in [57].

- In Chapter 5, we build upon the works in the earlier two chapters and propose a cooperative MARL approach that solves multi-agent dynamic routing and scheduling problems directly and evaluate our proposed approach on a real-world problem that incorporate both aspects of *dynamicity* and *multi-agent* in the law enforcement domain. The manuscript of the research work done in this chapter is currently under review.
- Lastly, in Chapter 6, we conclude this thesis by providing concluding thoughts on the key contributions of this thesis and discussing future research opportunities that can spring out of the works done in this thesis.

Chapter 2

Background

In this chapter, we first define the key terminologies and notations used to facilitate the reading and understanding of this thesis. After which, we present the background information and related works pertaining to dynamic routing and scheduling problem and Reinforcement Learning (RL) approaches to solve such problem. Lastly, we provide more detailed descriptions of the motivating problems and their related works.

2.1 Terminologies

For greater clarity and focus in reading and understanding the works presented in this thesis, the definitions of key terminologies used are presented and discussed in the following paragraphs.

Routing and Scheduling. We define routing and scheduling problem as a variant or extension of Vehicle Routing Problem (VRP). VRP was first introduced in [32]. We refer the readers to [118] for a comprehensive overview of VRP, its problem variations and solution approaches. For generality, the word *vehicle* is omitted since in our problem setting, vehicle can be generalized as an agent to represent a person, a group of people or even robots or drones. Meanwhile, schedule is essentially an extension of a route where route is a sequence of tasks typically represented by geographical locations while schedule refers to the time and duration that are attached to each component of the route. We simply use the term *plan* interchangeably to refer to either route or schedule or both.

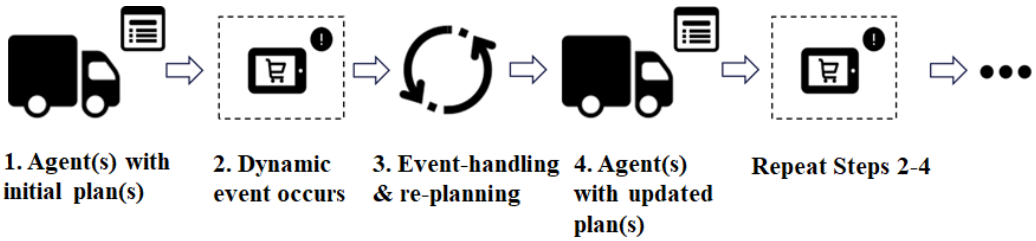


Figure 2.1: An illustration of the general structure of a dynamic routing and scheduling problem.

Dynamic Problem. For greater clarity and focus in the discussion, we formulate dynamic routing and scheduling problem as a sequential decision problem [96, 97]. Further discussion on sequential decision problem can be found in Section 2.3. The general structure of such dynamic problem is illustrated in Figure 2.1. Step 3 represents the key research focus of this thesis: *how do the agent(s) handle the dynamic event when it occurs (event-handling) and how to amend the initial plan(s) that have been disrupted due to the occurrence of dynamic event and in the presence of other agent(s) (re-planning)?.*

Multi-Agent. An *agent* can refer to an entity (usually with intelligence) or a computer software that acts independently within an environment and is driven by certain objectives [134]. Meanwhile, Multi-Agent System (MAS) refers to a system where multiple such agents exist and interact to achieve those objectives [39]. Within MAS, there exist variations in terms of the characteristics of agents involved, the degrees of interaction amongst agents and the environment itself [133]. For example, agents can be homogeneous or heterogeneous, the degree of their autonomy can range from low to high, the level of interaction may vary from mere observing to exchanging of information or resources, the nature of interaction can be cooperative or competitive and the availability of resources in the environment may be limited.

In this thesis, the word *agent* does not refer to an agent-based approach where different components of a solution approach or algorithm act as agents who work together to solve the problem (see [4, 82, 110, 128]). Instead, we refer an *agent* as a higher-order decision-making entity that is capable of executing complex action. These agents can be independent and can consist of multiple sub-agents. This is unlike the classical examples of multi-agent setting which come in the form of multiple

vehicles or multiple machines. The attributes of the agents are problem-specific and are motivated by the real-world domains that we described earlier (the exact attributes will be described in the later chapters). Meanwhile, the word *coordinate* implies the presence of a central agent or authority with limited information and autonomy over the agents. For example, in the context of an auction, the auctioneer is the central authority that facilitates the auction process but it does not influence or dictate agents' decisions. In our problem context, agents can refer to Logistics Service Providers (LSPs) or local police entities (belonging to a certain jurisdiction or sector) while the central coordinator can refer to a logistics platform owner or a government agency or a central police HQ.

Complex Action. In the context of dynamic routing and scheduling problem, we define a complex action to include *event-handling* (how to handle dynamic event when it occurs) and post-event *re-planning* (how to amend the initial plan that has been disrupted due to the occurrence of dynamic event). For example, in a police patrol problem, the complex action includes assigning an incident to a specific patrol team (*event-handling*); and rescheduling patrol schedule(s) of the assigned patrol team or even the other teams (*re-planning*). In addition, *re-planning* action in itself is complex because rerouting and/or rescheduling actions include both spatial and temporal dimensions.

2.2 Notations

Table 2.1 provides the set of common notations and the corresponding descriptions used in the model to describe multi-agent dynamic routing and scheduling problem in this thesis. Meanwhile, problem-specific notations are described separately in the relevant section corresponding to the specific problem.

2.3 Sequential Decision Problem

Sequential decision problem has been studied by various research communities ranging from stochastic optimization, optimal control, Markov Decision Process (MDP), RL, approximate dynamic programming and robust optimization to name a

Table 2.1: Set of common notations used in this thesis.

Notation	Description
k	Decision epoch.
K	Terminal decision epoch.
S_k	State at decision epoch k.
S_K	Terminal state or state at the end of planning horizon.
x_k	Joint action at decision epoch k
S_k^x	Post-decision state, state after taking action x.
ω_k	Realization of a dynamic event at decision epoch k.
$\hat{V}$	Value function approximate.
θ	Parameters of a value function or network.
γ	Discount factor.
$R(S_k, x)$	Reward function for taking action x at a given state S_k.
$x_{k,i}$	Action by agent i at decision epoch k.
$x_{k,-i}$	Joint action by all agents except agent i at decision epoch k.
$S_{k,i}$	Local pre-decision state of agent i at decision epoch k.
$S_{k,i}^{x,i}$	Local post-decision state of agent i at decision epoch k
t_k	Time period where decision epoch k occurs.
$\delta_i(k)$	A plan of agent i at decision epoch k.
$\delta(k)$	A joint plan of all agents at decision epoch k where $\delta(k) = (\delta_i(k))_{i \in I}$.
$\delta_{-i}(k)$	A joint plan of all agents except for agent i at decision epoch k.
$(\delta_i(k), \delta_{-i}(k))$	A joint plan where agent i follows plan $\delta_i(k)$ while the rest follows a joint plan $\delta_{-i}(k)$.
$\delta^x(k)$	A joint plan of all agents after executing action x at decision epoch k.
$u^i(\delta(k))$	Payoff/utility of agent i when all agents follow a joint plan $\delta(k)$.
$B_i(\delta_{-i}(k))$	Best response of agent i when all other agents follow a joint plan, $\delta_{-i}(k)$.

few [97]. As shown in Figure 2.2, a sequential decision problem consists of a sequence of states where in each state, a decision-making process is being triggered due to realization of new information [112].

Sequential decision problem consists of the following key elements namely:

- **State variables, S_k.** These refer to all the information needed to model the system and to make a decision.

- **Decision variables, x_k.** These refer to decisions made in response to realization of new information in the system and are usually governed by a policy π where π is a function that maps a state S_k to a decision $x_k = X^\pi(S_k)$. Decision can be discrete, continuous, categorical or multi-dimensional (i.e. complex).

- **Exogenous information, ω_k.** A realization of new information or changes to the current state variables that requires an action or decision to be taken.

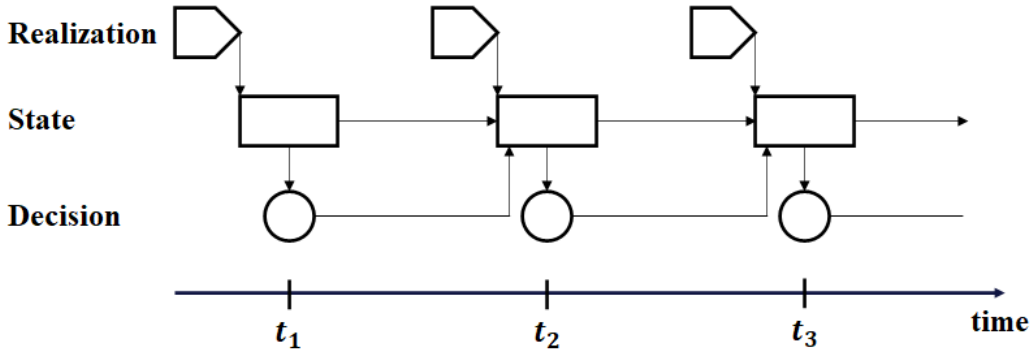


Figure 2.2: An illustration of sequential decision problem, adapted from [83]
.

- **Transition function,** P**.** This refers to a model that describes how the system moves from one state, S_k to next state, S_{k+1}. This function can be represented as follows: $S_{k+1} = P(S_k, x_k, \omega_{k+1})$. This function can be further broken down into two parts namely, from pre-decision state, S_k to post-decision state, S_k^x and then from post-decision state, S_k^x to the next pre-decision state S_{k+1}.

- **Objective function.** The goal of solving a sequential decision problem it to find an optimal policy π^* that maximizes the expected sum of rewards: $\max_{x_k \in X(S_k)} \Big\{ R(S_k, x_k) + \mathbb{E}\Big[\sum_{j=k+1}^{K} R(S_j, X^{\pi^*}(S_j)) | (S_k, x_k)\Big] \Big\}$. The second term of the Bellman equation is commonly referred to as a value function.

2.4 Markov Decision Process

MDP is a mathematical framework to represent sequential decision problems under uncertainty. MDP is typically defined as the following tuple:

$$\langle S, X, R, P, \gamma \rangle$$

where S represents a finite set of states, X a finite state of actions, R a reward function where $R(s, x)$ represents the reward of an agent who is in state s and takes action x; meanwhile, P represents the transition probability of going from state s to state s' after taking action x. The goal of solving an MDP is to find an optimal

policy π^* which indicates the optimal action x^* while agent is at a given state s. The key elements of an MDP are very much consistent with sequential decision problem.

The key challenge of representing a real-world sequential decision problem as an MDP is that the state space is rather huge. Therefore, factored MDP is a popular framework to represent large MDP in a compact manner [13]. Factored MDP relies on exploitation of problem structure to reduce the size of the state space. Factored MDP describes the set of states with a set of random variables, $S = \{S_1, S_2, ..., S_N\}$ [48]. This representation framework is in fact a kind of state aggregation method. In this framework, states are represented by a set of random variables which means that states that have the same values across the chosen set of random variables are aggregated and reduced to a single state.

For the multi-agent version of sequential decision problem, the MDP model (or MMDP [12] to be exact) is defined by the same tuple as above but with the following additional elements and definitions:

- I. a set of agents interacting in the system.
- S. a joint state space which can be decomposed into individual agents' local states, $s_i \in S_i$.
- X. a joint action space which can be decomposed into individual agents' actions, $x_i \in X_i$.

2.5 DVRP

Dynamic routing and scheduling problem is essentially a variant of Dynamic VRP (DVRP). DVRP is a variant of VRP whereby inputs to the problem may change during execution. Research on DVRPs witnessed a surge in the recent decade [98]. This trend is likely to continue given increasing needs toward services like same-day delivery, availability and accessibility of data and increased computing power.

DVRP can be broadly categorized into dynamic-deterministic and dynamic-stochastic [95]. Even within dynamic-stochastic VRP, stochasticity arises in different aspects namely travel times, demands, customers or combinations of these stochastic aspects [102]. Research focus in the recent decade has been on dynamic-stochastic

VRP (or DVRP in short) as it models more closely the real-world environment. We refer the reader to [95, 98, 102] for more detailed discussions on DVRP, its variants and solution approaches.

2.5.1 Modelling DVRP

DVRP can be modelled as MDP because in DVRP, decisions need to be made in view of uncertainty and are done sequentially. Conventional MDP in DVRP literature identifies optimal decision as next customer to visit at every decision point. There are inconsistencies and disconnectedness between such modelling framework with the DVRP solution techniques [121]. This is because the solution of a DVRP is a route plan and not just the next customer/location to visit. Ulmer *et al.* [120] first proposed route-based MDP as a unifying modelling framework for DVRP where the action is not merely the next customer to visit but the remaining route to be assigned to a vehicle. This thesis adopts the route-based MDP as the main modelling framework for dynamic routing and scheduling problem.

Why Route-based MDP? Unlike the passenger ride-sharing problem where single-action MDP makes more sense with somewhat simple consideration of passengers' pickup and drop off locations, our motivating problems suffer much more restrictions and complications.

Firstly, the planning horizon is much longer (for e.g. entire day). In urban logistics context, the delivery time windows and meal breaks must be considered, so the decision of which vehicle and the sequence to serve a given request must anticipate future dynamic requests. This implies that the value of a state needs to take into account the projected route of the vehicle in anticipation of future new requests. The calculations of rewards take into account the time window violation and waiting time that may occur into the future and these calculations can only be derived if a route is available. Similarly, in police patrol context, we need to consider the total number of effective patrol time at each location in a given shift. Computation of reward will require the availability of the planned patrol schedule for the whole shift.

Secondly, the visibility of entire routes is important in both problem settings. Practically speaking, LSPs or police do not start off their workday/shift with an

empty schedule and build it incrementally as the day goes. In urban logistics context, planned routes are important to drivers from the point of view of execution planning, cargo loading, tracking and communication on the ground. Each delivery job is tied to its respective cargo that needs to be loaded to the assigned vehicle, and swapping cargo between vehicles is not viable operationally. Meanwhile, in police patrol context, planned routes are important to ensure that all mandatory patrol areas are being patrolled at least for a minimum required time. In police operations, there may be occasions where patrol agents need to be at certain location at certain time and this requirement can only be incorporated if a planned schedule exists.

More detailed discussion on route-based MDP and how it differs from conventional MDP can be found in [121].

2.5.2 Solution Approaches to DVRP

Based on the survey by Ritzinger *et al.* [102], there are two broad categories of approaches for solving DVRP namely offline or pre-processsed decision support and online decision. Lately, there have been works that try to combine both approaches (hybrid or offline-online).

Offline or Pre-processed Decision Support. Policies or values for decision-making are computed prior to execution of plan. Here, the problem is usually formulated and solved as an MDP. Unfortunately, MDP-based approaches (specifically tabular-based ones) fall into the curse of dimensionality and hence are not suitable for most real-world problems [95]. Approximate Dynamic Programming (ADP) approaches are commonly used to tackle the scalability issue, and one such ADP method for DVRP is Approximate Value Iteration (AVI) (see A.1 for a sample implementation of an AVI algorithm). Agussurja *et al.* [1] and Ulmer *et al.* [124] proposed AVI to solve DVRP as an MDP. Both papers proposed state aggregation and representation to further overcome the challenge of large state space. Another main challenge of ADP is to generate enough scenarios during the training phase so as to accurately assign a value to a state. AVI approximates the value function as a lookup table and may fail if certain state is not encountered during the training phase. Most works used AVI on single vehicle setting without time window constraints which may not extend well for more complex problems like those with multiple

vehicles and additional constraints such as time windows or capacity.

Online Decision. Unlike the pre-processed decision support approaches, online decisions do not compute optimal global policy; rather computations are performed during the execution of plan. Common approaches in this category are usually termed as lookahead strategy or rolling horizon procedures [96] such as rollout algorithms [10, 46, 108] and Multiple Scenario Approach (MSA) [9, 127] (see A.2 for a sample implementation of an MSA algorithm). They are mainly sampling-based approaches. The common feature of these methods is that they focus on the current state and instance. They do not consider the values of all possible states but only the relevant states at the decision point. At the decision point, the methods do a *roll-out* or *lookahead* to simulate what will happen in the future and use this information to guide decision-making. Bent and Van Hentenryck [8] introduced the consensus algorithm into MSA to solve online VRPs with stochastic customers. Consensus algorithm performs an offline optimization on the available and sampled requests once per scenario and returns the decision with the largest score or lowest cost. Online decisions approaches are suitable where there is no strict time constraint imposed when decision-making is required and work well with increasing degree of dynamism (DoD). Thus, the choice between online or offline methods will be very much dependent on how fast decisions need to be made during execution time.

Hybrid (Offline-Online). There have been attempts to combine both approaches to leverage the strengths of both approaches. Ulmer *et al.* [119] proposes offline-online approximate dynamic programming which embeds the offline VFA into the online roll-out algorithm. Ulmer *et al.* [119] only managed to run at most 16 *lookahead* samples for every decision instance due to resource constraint. The gain achieved through this hybrid approach may not be compelling enough given the vast compromise in terms of decision-making time. De Filippo *et al.* [34] proposed an integration of offline and online optimization by considering multi-stage optimization problems where the first phase requires offline decision and the subsequent phases require online decision. The proposed approach works on DVRP with stochastic travel time where customers are assigned offline but routes are optimized online. This, however, does not work for other types of DVRP such as DVRP with stochastic customers as

customer requests are not known beforehand.

Different authors propose different ways of categorizing the solution approaches. For example, Ulmer *et al.* [121] categorize the different solution approaches into 4 main types namely Reoptimization (RO), Policy Function Approximation (PFA), Lookahead Algorithm (LA) and Value Function Approximation (VFA). Meanwhile, Soeffker *et al.* [112] categorize the approaches into either predictive or prescriptive. Despite the differences in how the categorizations are done and how the terminologies are being used and defined, these proposed categorizations are quite coherent and consistent with one another.

2.6 Learning to Solve Routing and Scheduling Problem

Recent advancement in Deep Learning (DL) has enabled machines to outperform many classical approaches in computer vision and natural language processing by learning from data [68]. In addition, Deep RL is also able to outperform human experts in games like Atari [85] and Go [111]. The recent significant progress in DL has given rise to an increased interest in learning-based approaches to solve NP-hard Combinatorial Optimization Problems (COPs) (see [7, 126]).

Learning-based approaches to solve COPs are appealing because machines can be trained to discover their own heuristics and learn the optimal policy to solve different instances of the same problem. More often than not, NP-hard problems rely on on heuristics to solve and their performances are evaluated empirically [45]. Most traditional heuristics are designed by human and require human expertise. On the other hand, DL algorithms have shown to be able to learn from larger datasets and detect useful patterns and extract features that human may have missed out [75].

Routing and scheduling problem is an NP-Hard COP. In this section, we provide a literature review on related works that propose RL approaches to solve VRP, DVRP and their variants which come close to the dynamic routing and scheduling problem that this thesis seeks to address.

2.6.1 RL Approaches to Solve Routing and Scheduling Problem

There have been numerous works that proposed learning-based approaches especially RL-based ones to solve VRP. Existing RL-based approaches solve the problem either through constructing solution node-by-node (see [63, 88, 136]) or through improving an existing solution (see [27, 42, 80]). Li *et al.* [70] use the terms *end-to-end* and *step-by-step* approaches to differentiate the two approaches while Falkner and Schmidt-Thieme [38] use the terms *construction* and *improvement*.

The *end-to-end* or *construction* approach converts the routing problem to a vehicle tour generation problem. This approach is the more popular approach between the two. Although numerous authors have proposed newer and better models, the methodologies adopted have the following commonalities:

1. Encoder-decoder network architecture (for e.g. Pointer Networks [88], Transformer [63]) with attention mechanism to encode the input nodes and decode the output route node-by-node.

2. Embedding of input data such as the location of the nodes, demands at each node and vehicles' capacities in the encoder network.

3. Policy gradient algorithms (for e.g. REINFORCE [63, 88, 136] and Asychronous Advantage Actor Critic (A3C) [88]) to train the network.

Meanwhile, for the *step-by-step* or *improvement* approach, the methodology starts with an initial feasible solution and iteratively improve it through learning to choose local search operators [80] or to select nodes to remove and insert into the route [27, 42]. This type of approach combines RL and classical heuristics to solve routing problems. Various network architectures are proposed to represent the policy network such as Long Short-Term Memory (LSTM) [27], Graph Attention Network (GAN) [42] or Multilayer Perceptron (MLP) with Attention [80]. The policy network is then trained using either A3C [27], REINFORCE [80] or generic Actor-Critic with Proximal Policy Optimization (PPO) [42].

Either approaches have both pros and cons. *Construction* approach can compute solution quickly and more suitable for simpler problems like Travelling Salesman Problem (TSP) or single-vehicle VRP. This is because the input to the learning

model lacks contextual information that a complete solution can provide such as total waiting time or total time violations. In addition, since the resulting action at every juncture is next node to visit and no tentative complete solution is available, complex routing constraints cannot be integrated [50]. Meanwhile, *improvement* approach provides a better solution quality for more complex routing problems like VRP with Time Windows (VRPTW), Dial-A-Ride Problem (DARP) and multi-vehicle VRP. However, as with classical heuristics, *improvement* approach is time-consuming.

Many recent works on *construction* approach build upon the Attention Model (AM) introduced by Kool *et al.* [63] to further enhance the capability of this approach type to address more complex routing problems. For example, Multi-Agent Attention Model (MAAM) [138] and Joint Attention Model (JAMPR) [38] address multi-vehicle routing problems while Multi-Decoder Attention Model (MDAM) [136] extend the original AM to efficiently explore more diverse solutions. In addition, by leveraging more advanced deep learning architecture, Li *et al.* [71] introduces vehicle selection decoder to solve Heterogeneous VRP (HVRP). As of the time that this thesis is written, there are numerous other works either published or in the works that are within this popular area of research.

2.6.2 RL Approaches to Solve Dynamic Routing and Scheduling Problem

Most of the RL-based approaches mentioned above solve static routing problems such as TSP, Capacitated VRP (CVRP), Split Delivery VRP (SDVRP), VRPTW, Capacitated VRPTW (CVRPTW). Only Nazari *et al.* [88] evaluated its approach on a DVRP.

Increasingly, there also have been many recent works that addressed the variants of DVRP (see [25, 26, 28, 73]). Most of these works adopt a two-stage approach. Chen *et al.* [28] uses the terms *dispatching-level* and *routing-level* while Chen *et al.* [25] uses the terms *dispatch policy* and *matching policy* to describe each component of the two-stage approach. In a two-stage approach, the problem is decomposed into two stages and solved one after another. The first stage involves learning the assignment/dispatch policy to determine which agent or vehicle or machine to be selected. In this stage, several RL algorithms are proposed such as Actor-Critic[28],

Deep Q-Network (DQN) [26], Double DQN (DDQN) [73] or QMIX [25]. After which, routing/scheduling decision is executed based on the decision made in the first stage. In this second stage, the decision is either computed using an exact method or a heuristic. This is done to reduce the action space.

2.6.3 MARL Approaches to Solve Multi-Agent Dynamic Routing and Scheduling Problem

We focus our discussion on cooperative Multi-Agent RL since the nature of the problems that this thesis seeks to address are leaning more towards cooperative rather than adversarial. The research in cooperative MARL has been to address the following two main challenges namely partial observability and scalability. The concept of Centralized Training Decentralized Execution (CTDE) was introduced to address the partial observability and since have been leveraged by many popular MARL algorithms such as COMA [40] and MADDPG [79]. To further address scalability, many works such as Value Decomposition Network (VDN) [115], QMIX [100] and QTRAN [113] propose value function factorization based on Individual Global Max (IGM) assumption on top of CTDE to learn decentralized policies.

Current cooperative MARL approaches fall short in addressing sequential decision problems with complex action (such as routing and/or scheduling) directly. For instance, VDN, QMIX and COMA only solve problems with discrete actions while MADDPG addresses problems with continuous actions. Current MARL approaches solve routing and/or scheduling problem either by decomposing the complex action into two stages and learn the policy in one of the stages, defining the actions to be either discrete or continuous, or combining both approaches.

- **Two-stage approach.** Chen *et al.* [25] proposed DeepFreight, a model-free DRL-based approach to solve multi-transfer freight delivery. To solve the problem, the authors decompose the problem into two stages: truck-dispatch and request-matching and leverage on QMIX to learn the dispatch policy while implement a separate matching algorithm for the second stage. Similarly, Chen *et al.* [26] solve a same-day delivery problem with vehicles and drones by decomposing the problem into two stages: assignment and routing. The authors propose a deep Q-learning approach to learn the assignment policy

and update the routing via heuristic.

- **Redefine action.** To solve a dynamic train timetabling problem, Li and Ni [72] propose a Multi-Agent Asynchronous Actor Critic (A2C) approach. Instead of defining the action as complex action (i.e. rescheduling), the authors define a discrete action in the form of integer-typed dwell time i.e. how long a train stops at a certain station before moving on. In a similar fashion, Zhang *et al.* [137] solves job shop scheduling by defining the action as a dispatch action instead of defining a complex action (i.e. scheduling).

- **Combination.** Chen *et al.* [28] decomposes a dynamic courier dispatch problem into two stages namely dispatch and routing stage. The authors propose an MARL approach to learn the dispatch policy and define a discrete action space consisting of a cartesian product of the next grid to visit and the corresponding period of stay in the grid. Ma *et al.* [81] propose a hierarchical approach to solve dynamic pickup and delivery problems by introducing two levels of agent. The first agent learns a policy to decide which orders to be released while the second agent learns a policy to choose a local search operator to reroute the vehicles. Both stages involve discrete actions.

To coordinate amongst agents, various works focus on the aspect of learning to communicate in cooperative setting such as when communication is needed [54], what and who to communicate with [33] and how to represent the communication network [11, 53].

However, communication alone does not guarantee coordination especially when agents act simultaneously [106]. Acting simultaneously in the context of multi-agent dynamic routing and scheduling problem is not ideal because it may cause uncoordinated actions resulting in poor event-handling decision. For example, in patrol scheduling problem, there may be scenarios where no agent or more than one agent respond(s) to an incident and this would result in an unattended incident or delays as agents may be waiting to attend incidents that have already been responded by other agent.

2.7 Motivating Problems

This section provides detailed descriptions of the motivating problems and discusses some related works pertaining to solution approaches to such problems.

2.7.1 Same-Day Delivery Routing Problem

We consider a real-time planning problem in urban logistics in which changes to the routes and tasks are required in response to dynamic events. For example, customers may add or cancel requests throughout the day; travel or service times may change drastically due to traffic congestion; order amounts or demands may need to be modified. To ensure customers' demands are met, logistics service providers need to dynamically respond to these changes quickly.

More precisely, we consider a dynamic multi-vehicle same-day delivery routing problem with time windows and both known (i.e. fixed) and stochastic customers. This problem is essentially a DVRP with stochastic customers. Azi *et al.* [3] were the first to formulate same-day delivery problem as a DVRP. Since then, there were many other works that study such variant of DVRP (see [62, 124, 127]).

2.7.2 Dynamic Bi-Objective Police Patrol Dispatching and Rescheduling Problem

In policing, occurrences of unexpected incidents often disrupt the execution of an existing plan. Police patrol agents need to be dispatched and patrol schedules need to be adapted to respond to dynamically-occurring incidents while fulfilling the two often conflicting objectives of projecting police presence (*proactive* patrol) and responding to incidents in a timely manner (*reactive* patrol). In addition, such complex decisions need to be made rather quickly and sometimes instantaneously. To add to the complexity and practicality to the problem, changes to the existing schedules need to be kept as minimal as possible.

We define this real-world problem as a Dynamic Bi-Objective Police Patrol Dispatching and Rescheduling Problem (abbrev. DPRP) which is a variant of DVRP with some elements of a university time-tabling problem.

2.7.2.1 Police Patrol Routing Problem

DPRP belongs to a larger class of problem called police patrol problem. There are many aspects within the scope of police patrol problem ranging from designing of patrol district, resource allocation within a district, route design to combinations of any of these elements [107]. The discussion in this thesis will be mainly on the routing and scheduling aspects of the problem.

Many existing works on police patrol problem adopt the hotspot patrol strategy which mainly addresses the *reactive* patrol aspect. Such approaches typically begin by predicting crime hotspots spatially and temporally based on past data and then allocating police resources accordingly (see [23, 60, 69, 87]). However, there is a gap in current literature in addressing *proactive* patrol. One such work that addresses both aspects is [129]. The authors addressed the dual objectives by decomposing the problem into two sub-problems.

Meanwhile, most existing works on police patrol routing or scheduling assume routes and schedules are fixed and none takes into account the disruption to the existing routes or schedules after incident response. Therefore, this thesis serves as the first work in the literature to address the dynamic version of the police patrolling problem while taking into consideration both *proactive* and *reactive* patrolling.

There is also another stream of works that represents the police patrol routing problem as Stackelberg Security Games (SSG) [18, 125]. However, Rosenfeld and Kraus [105] pointed that SSG may not always be applicable to all patrol context. SSG assumes the existence of an attacker-defender relationship which may not always be present in many problem settings. Police handles a variety of incidents ranging from traffic incidents, maintaining order and crowd control or providing general assistance to public. This paper assumes a problem setting where incidents happen rather randomly instead of being triggered by a strategic intent to outwit the police.

2.7.2.2 Multi-Agent DPRP

Multi-Agent DPRP or MADPRP in short is a multi-agent extension to DPRP. In MADPRP, there are multiple police sectors consisting of multiple police patrol teams within each sector. We define these sectors as agents. Each police sector is in-charge of patrolling their local patrol areas throughout a given shift. At the

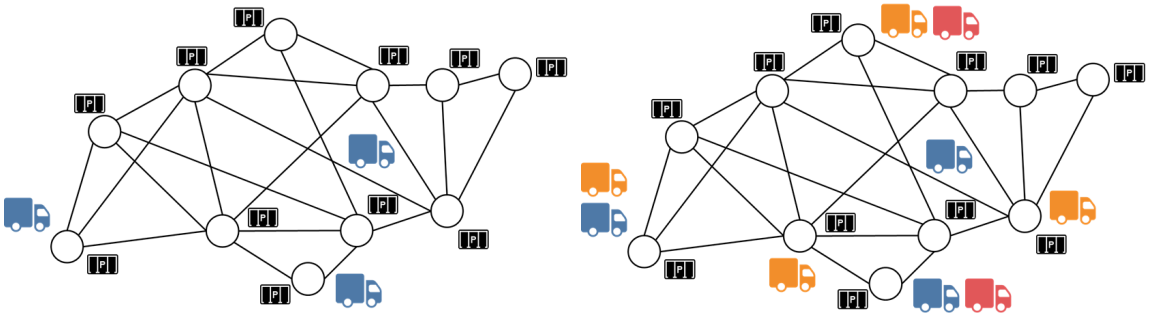
start of the shift, each patrol team in each sector is assigned to an initial patrol schedule. Throughout the shift, incidents occur dynamically across the different patrol areas across different sectors. Patrol team needs to be dispatched to respond to each incident resulting in disruption to the initial schedules. Coordination is crucial in this problem as patrol teams can cross over to other sectors to respond to an incident and perform routine patrol so as to ensure that incidents are responded within target time and all patrol areas are sufficiently patrolled.

This is a new multi-agent variant of a DVRP without any prior works. Existing approaches to solve multi-agent version of DVRP are very specific to transportation or logistics problem scenarios (see [78, 130]).

2.7.3 Multi-Party Vehicle Routing Problem with Location Congestion

Business-to-Business (B2B) pickup-delivery operations to and from commercial or retail locations involving multiple parties, commonly referred to as LSPs, more often than not cannot be done in silos. Resource constraints at these locations such as limited parking bays can cause congestion if each LSP adopts an uncoordinated, selfish planning. Thus, some form of coordination is needed to deconflict the schedules of these LSPs to minimize congestion thereby maximizing logistics efficiency. This research is motivated by a real-world problem of improving logistics efficiency in shopping malls involving multiple independent LSPs making B2B pickups and deliveries to these locations in small, congested cities where space is scarce.

Collaborative planning for vehicle routing is an active area of research and had been shown to improve efficiency, service level and sustainability [41]. However, collaborative planning assumes that various LSPs are willing to collaborate with each other by forming coalitions, exchanging of information and/or sharing of resources to achieve a common objective. Ideally if we have one single agent who can control the routes and schedules of multiple LSPs with complete information and collaboration amongst the LSPs, we may achieve some form of system optimality. However, an unintended outcome is that some LSPs may suffer more loss than if they adopt their own planning independently. Moreover, such centralized approach is not scalable and not meaningful in solving the real-world problems, since LSPs may not always



(a) There are limited resources at each of the location.

(b) There exist multiple independent LSPs in the environment. The limited resources at each location are shared among the multiple LSPs.

Figure 2.3: Single-LSP VRP with Location Congestion and the multi-LSP version of the problem.

be willing to collaborate with one another. In a non-collaborative context, LSPs are independent entities who can only make decision locally in response to other LSPs' decisions and they do not interact directly with each other to collaborate or make joint decision.

The underlying problem can be seen as a Vehicle Routing Problem with Pickup and Delivery, Time Windows and Location Congestion with multiple LSPs (or ML-VRPLC in short) (see Figures 2.3a and 2.3b).

2.7.3.1 VRP with Location Congestion

VRP with Location Congestion (VRPLC) is essentially a variant of a classical VRP with Pickup and Delivery, and Time Windows (VRPPDTW) but with cumulative resource constraint at each location [64]. Resources can be in the form of parking bays, cargo storage spaces or special equipment such as forklifts (see Figure 2.3a). VRPLC can be considered as a VRP with resource synchronization [37].

In VRPLC, there are temporal dependencies between routes and schedules that do not exist in classical VRPs. In classical VRPs, arrival times of vehicles are merely used to ensure time window feasibility. In VRPLC, changes to the time schedule of one route may affect the time schedule of another routes in the form of wait time or time window violation. Many existing approaches to VRP do not take into consideration this relationship between routes and schedules.

Lam and Van Hentenryck [64] proposed a branch-and-price-and-check (BPC)

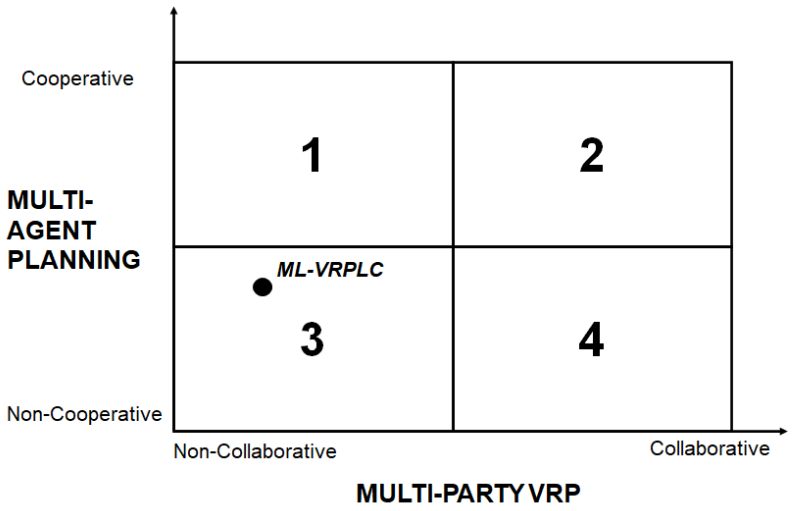


Figure 2.4: ML-VRPLC as a Multi-Party VRP and Multi-Agent Planning Problem.

approach to solve a single-LSP VRPLC. It is inspired by a branch-and-cut-and-price method for VRPPDTW [103] and combines it with a constraint programming subproblem to check the VRPPDTW solutions against the resource constraints. However, BPC approach can only find feasible solutions for instances up to 150 pickup-delivery requests and proves optimality for up to 80 requests given a time limit of 2 hours. Therefore, this approach is not scalable when applied directly to solve ML-VPRLC since pickup-delivery requests are usually in the region of hundreds per LSP and for our problem setting, solution is expected within a region of 1 hour due to operational requirement. In addition, a direct application of BPC to ML-VRPLC assumes a fully centralized, collaborative planning approach which we have concluded earlier that it may not be practical and not meaningful.

2.7.3.2 ML-VRPLC

ML-VRPLC can be considered as a problem belonging to an intersection between two main, well-studied research areas namely **Multi-Party VRP** and **Multi-Agent Planning** (MAP). Existing approaches to Multi-Party VRP and MAP can broadly be categorized based on the degrees of collaboration and cooperation respectively. Based on our understanding of our problem setting, approaches to ML-VRPLC should fall within Quadrant 3 (see Figure 2.4). In the following paragraphs, we discuss existing works that are relevant to solving ML-VRPLC.

ML-VRPLC as a Multi-Party VRP. To solve VRPs involving multiple parties similar to ML-VRPLC, many existing works in the literature focus on collaborative planning approaches. Gansterer and Hartl [41] coined the term collaborative vehicle routing and it is a big area of research on its own. Collaborative vehicle routing can be classified into centralized and decentralized collaborative planning. The extent of collaboration ranges from forming of alliances or coalitions (for e.g. [30, 47]) to sharing of resources such as sharing of vehicles or exchanging of requests through auction (for e.g. [31, 131]). Many of the existing works in forming of coalitions and auction mechanism fall within Quadrants 2 and 4 respectively. However, we have established earlier that existing works in this area are not directly applicable to our problem due to the non-collaborative nature of the LSPs.

ML-VRPLC as an MAP Problem. MAP is simply planning in an environment where there exist multiple agents with concurrent actions. Approaches to MAP can be further categorized into cooperative and non-cooperative domains although most MAP problems lie in between the two domains.

1. **Cooperative Domain.** Cooperative MAP involves agents that are not self-interested and are working together to form a joint plan for a common goal [117]. Thus, existing approaches in this category fall within Quadrants 1 and 2. Brafman and Domshlak [15] introduced MA-STRIPS, a multi-agent planning model on which many cooperative MAP solvers are based on. Nissim *et al.* [91] proposed a two-step approach consisting of centralized planner to produce local plan for each agent followed by solving a distributed constraint satisfaction problem to obtain a global plan. Meanwhile, Brafman *et al.* [16] introduced the concept of planning games and propose two models namely coalition-planning games and auction-planning games. Those two models assume agents collaborate with each other through forming of coalitions or through an auction mechanism; similar to the approaches within the collaborative vehicle routing domain. In general, the approaches in this domain essentially assume cooperative agents working together to achieve a common goal.

2. **Non-Cooperative Domain.** Planning in the context of multiple self-interested agents where agents do not fully cooperate or collaborate falls

into the domain of non-cooperative game theory. MAP problem can be formulated as strategic game where agents interact with one another to increase their individual payoffs.

Lambert Iii *et al.* [67] proposed a sampled fictitious play algorithm as an optimization heuristic to solve large-scale optimization problems. Optimization problem can be formulated as a n-player game where every pure-strategy equilibrium of a game is a local optimum since no player can change its strategy to improve the objective function. Fictitious play is an iterative procedure in which at each step, players compute their best replies based on the assumption that other players' actions follow a probability distribution based on their past decisions [17]. This approach had been applied to various multi-agent optimization problems where resources are shared and limited such as dynamic traffic network routing [43], mobile units situation awareness problem [66], power management in sensor network [21] and multi-agent orienteering problem [24].

Meanwhile, Jonsson and Rovatsos [59] proposed a best-response planning method to scale up existing multi-agent planning algorithms. The authors used existing single-agent planning algorithm to compute best response of each agent to iteratively improve the initial solution derived from an MAP algorithm. It is scalable compared to applying the MAP algorithm directly to an MAP planning problem. However, the authors evaluated their proposed approach only on standard benchmark problems such as those found in the International Planning Competition (IPC) domains. On the other hand, De Nijs *et al.* [35] applied a similar best-response planning approach to a real-world power management problem.

Chapter 3

RL Approach to Solve Dynamic Routing and Scheduling Problem

In this chapter, we introduce a new RL-based solution approach that combines value function approximation step with heuristic to solve dynamic routing and scheduling problems. Our proposed approach addresses the multi-dimensional nature of routing/scheduling action without decomposing the action into two stages namely dispatch/assign and reroute/reschedule. We evaluated this proposed solution approach against two real-world problems of different domains with slightly different problem settings, namely:

1. Dynamic multi-vehicle same-day delivery routing problem with time windows and stochastic customers. This problem is essentially a DVRP with stochastic customers.

2. Dynamic Bi-Objective Police Patrol Dispatching and Rescheduling Problem (abbrev. DPRP). This problem is a variant of DVRP with stochastic customers with dual objectives and some elements of a university time-tabling problem.

3.1 Motivation

Current DVRP approaches in the literature fall short when comes to dealing with real-world dynamic problem settings. Rerouting and rescheduling decisions need to be computed quickly and sometimes instantaneously. Current approaches in the OR community are mainly sampling-based/online approaches which reportedly take the order of 100 seconds per decision [127]. On the other hand, most offline, MDP-based approaches that adopted Value Function Approximation (VFA) method

only managed to solve single vehicle problems without any additional constraints like time windows or capacity. Approximate Value Iteration (AVI), the most common offline approach in OR community, relies on Monte Carlo method to update the learned value function which is not sample efficient and require training episodes in the magnitude of millions [122, 124]. In addition, look-up based AVI may not extend well for more complex problems due to larger state space.

RL is a natural fit to solve DVRP because DVRP is essentially a sequential decision-making problem and can be formulated as MDP. In addition, RL is model-free and learning is done via interacting with environment without the need to know or model the statistical distribution of dynamic event which is unknown in real-world context. RL facilitates offline learning of policies and values, which are computed beforehand and can be quickly executed during run-time for for instantaneous decision-making.

Current state-of-the-art RL approaches to solve routing problem can only handle static problems with simple routing constraints. None takes into consideration the temporal elements of a route (i.e. schedule). Meanwhile, most RL-based approaches dealing with dynamic variant of routing and scheduling problems decompose the problems into two stages (see Section 2.6.2). None of the work directly addresses the routing/scheduling action which is multi-dimensional in nature. Meanwhile, solution approaches that combine heuristic and learning to solve COPs are not new (see [116]). However, existing works that combines RL with heuristic mainly solve static routing and scheduling problems [27, 42, 80].

The proposed solution approach in this chapter of the thesis aims to address the gaps mentioned above.

3.2 VFA via TD Learning with Heuristic

Our proposed solution approach combines Deep RL (specifically neural networks-based Temporal-Difference (TD) learning with experience replay) to approximate the value function and a heuristic to compute rerouting/rescheduling decision. Deep RL-based method uses neural networks to approximate the value function and the corresponding parameters of the networks are tuned based on the observed rewards over many training episodes. There is no need to make assumption on the

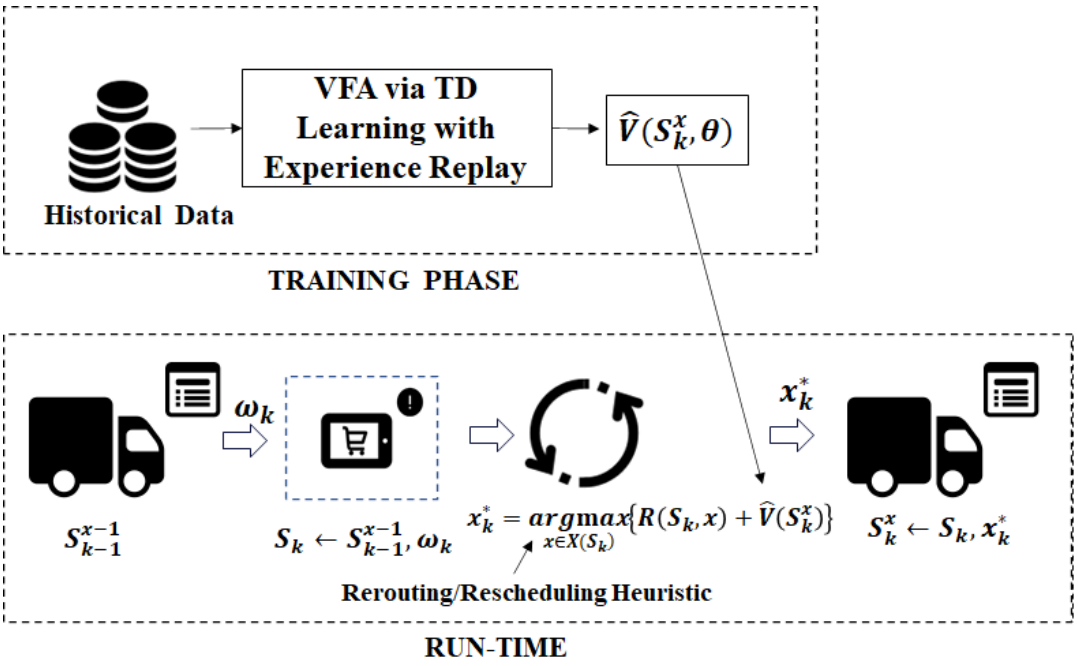


Figure 3.1: VFA via TD Learning with Heuristic.

underlying relationships between the features in the states. In addition, compared to lookup-based ones like AVI (see Section 2.5.2), VFA via non-parametric models such as neural networks is a popular choice for more complex problems because they are able to handle larger state space. As shown in Figure 3.1, our solution approach, comprises two phases namely training phase and run-time.

3.2.1 Training Phase: Value Function Approximation

We propose to approximate the value function for each post-decision state, $\hat{V}(S_k^x, \theta)$ with neural networks. To learn the parameter θ, we propose the use of on-policy TD learning with experience replay.

Algorithm 1 describes how our proposed neural networks-based TD learning with experience replay algorithm approximates the value function. This algorithm is adapted from the vanilla version of Deep Q-Network (DQN) with experience replay [84] with $V(S^x, \theta)$ replacing $Q(S, x, \theta)$ at lines 10, 15, 19 and 20 since we are approximating the value function instead of the state-action value or Q-function. Similar to DQN, two value function networks are used to deal with non-stationarity of the target network and experience replay to ensure that randomly selected samples are independent. However, another difference is that this proposed algorithm is

on-policy while DQN is off-policy (see difference in line 15). Mini-batch gradient descent is used to update the parameter of target network as shown in line 19. In addition, a routing or scheduling heuristic is used in line 10 to choose the optimal rerouting or rescheduling decision since computing the 'true' arg max involves brute force enumerating of all possible rerouting or rescheduling sequences.

Unlike the two-stage approach that is commonly adopted in the literature, this approach learns the dispatch/assignment and rerouting/rescheduling policies jointly because the value function represents the value of a state after executing both decisions. The advantage is that the learned value function is utilized by the heuristic to search for better decision. Thus, the heuristic is also learning and adapting. Meanwhile, in two-stage approach, the heuristic only provides the reward signal to learn the dispatch/assignment policy but the learned policy does not inform how the heuristic works. Thus, there is in fact no learning in the second stage. However, the downside of this proposed approach is that it is more time-consuming than the two-stage approach. Nevertheless, we show in our experiments that our joint approach results in better quality solution within operationally realistic time limit for moderately-sized problems.

This proposed solution approach provides a general framework that is flexible to accommodate any learning algorithms (must be value-based) and heuristics.

Why Approximate Value Function? We chose an algorithm that approximates the value function instead of Q-function. This is deliberately chosen to fit the context of our problem. In dynamic routing and scheduling problem, a decision involves a rerouting/rescheduling of the remaining routes or schedule which does NOT consist of an explicit action per se. As shown in Figure 3.2, in the context of DVRP with stochastic customers, the decision in response to new customer 6 is not merely choosing the position for insertion but it can also include swapping of existing customers' positions. This means that the action space is exponentially large with respect to number of customers, since every scenario consists of different possible actions due to different possible lengths of routes and different possible customer locations. Zhang and Dietterich [139] concluded that Q-function is not suitable in problem settings where the actions are largely dependent on the current states like job scheduling problems. Specifying the Q-value of each action is not

Algorithm 1: VFA via TD Learning with Experience Replay

Input : No. Simulation Runs N, Replay Memory D, Initial Value Function V with random weights θ, Initial Target Value Function $\hat{V}$ with random weights θ^-

Output : θ

$i = 1$
while $i \leq N$ **do**
  Initialise S_0 with the initial route/schedule
  $k = 1$
  while $S_k \neq S_K$ **do**
    if *new event = True* **then**
      $S_k \leftarrow (S^x_{k-1}, \omega)$
      With probability ϵ select a random decision x_k
      Otherwise
      // use heuristic to find the optimal feasible decision x_k
      $x_k \leftarrow \arg\max_{x \in X(S_k)} \{R(S_k, x) + \gamma V(S^x_k, \theta)\}$
      // proceed with the updated route/schedule
      $S^x_k \leftarrow (S_k, x_k)$
      Store transition $(S_k, S^x_k, R(S_k, x_k))$ in D
      Sample random minibatch of transitions $(S_j, S^x_j, R(S_j, x_j))$ from D
      if $S^x_k \neq S_K$ **then**
        $y_j \leftarrow R(S_j, x_j) + \gamma \hat{V}(S^{x_j}_j, \theta^-)$
      else
        $y_j \leftarrow R(S_j, x_j)$
      end
      Perform a gradient descent on $(y_j - V(S^{x_{j-1}}_{j-1}, \theta))^2$ with respect to parameter θ
      Reset $\hat{V} = V$ for every C steps
    else
      // proceed with the existing route/schedule
      $S^x_k \leftarrow S_k$
    end
    $k \leftarrow k + 1$
  end
  $i \leftarrow i + 1$
end
return θ

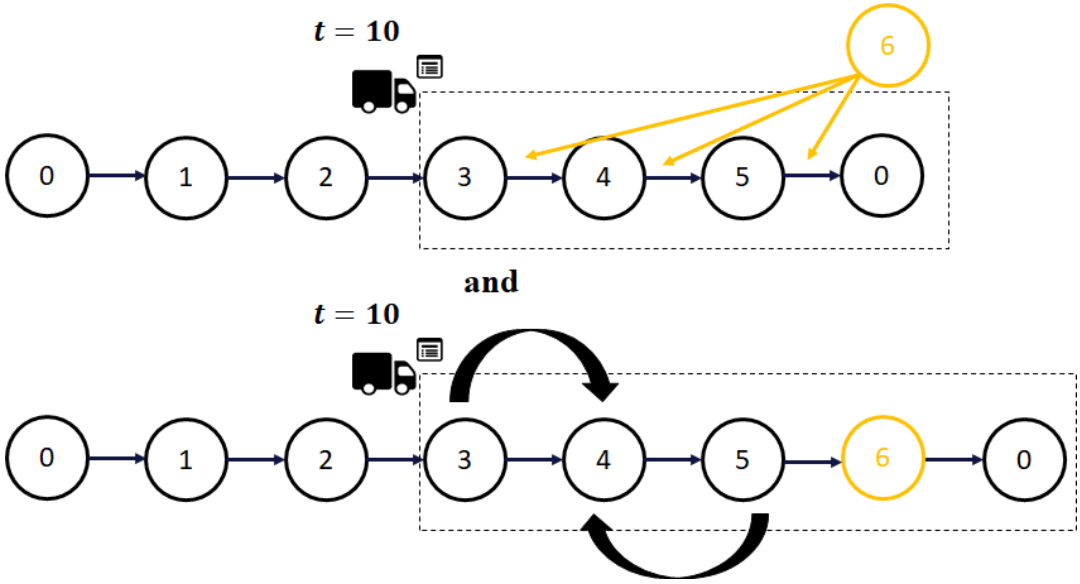


Figure 3.2: A rerouting decision is more than insertion of new customer and may include swapping of existing customers.

possible since action space changes depending on the current state. This also means that off-policy TD learning method like Q-learning and policy gradient method may not be applicable directly since the action space changes depending on the current state. Meanwhile, Powell [96] proposed the use of VFA around the post-decision state and showed that VFA using the post-decision state is equivalent to Q-learning. However, the advantage of VFA using the post-decision state is that it can handle multi-dimensional decision variables. The exact proof and explanation can be found in Chapter 4 of [96]. Thus, we conclude that approximating the value of a post-decision state using on-policy TD learning is more suitable for this problem setting where the action is multi-dimensional.

3.2.2 Run-Time: Rerouting/Rescheduling Step

The learned value function is subsequently utilized by the rerouting/rescheduling heuristic to generate the revised routes or schedules during run-time. Heuristics are chosen for fast computation and real-time decision-making. As shown in Figure 3.1, a routing or scheduling heuristic is used to choose the optimal rerouting/rescheduling decision since computing the 'true' arg max involves brute force enumerating of all possible rerouting/rescheduling sequences.

3.2.3 State Representation

As mentioned in Section 2.5.2, similar to ADP, value-based RL approach face curse of dimensionality in terms of the state space. Due to the large state space, the post-decision states are usually aggregated or represented based on certain handcrafted features. It is important to find features that exploit the structure of the problem and are able to sufficiently differentiate between two distinct states which may require two different policies. Another way to reduce the state space is is to build an encoder network to transform input data to low-dimensional vectors so as to allow machine to directly learn the key features. The choice is problem-dependent and this thesis proposes a hybrid approach to leverage the strength of machine while not ignoring human expert inputs completely.

3.3 Application 1: Same-Day Delivery Routing Problem

We first evaluate this proposed approach on a dynamic multi-vehicle same-day delivery routing problem with time windows and stochastic customers. This problem is essentially a DVRP with stochastic customers.

3.3.1 Problem Description and Model

Table 3.1 provides the set of key notations and the corresponding descriptions used in this problem.

3.3.1.1 Problem Description

We are given a fleet of M identical vehicles initially located at the depot at the start of the day. Each vehicle has an initial route $\beta_m(0)$ consisting of a sequence of customer orders, C_0 to fulfill for that particular day. Every route starts and end at the depot. Every order has a delivery time window $[e_n, l_n]$. In addition, there is a lunch hour when no delivery can be made. A waiting time is incurred if the vehicle either arrives early or during the lunch hour. Throughout the day, new orders arrive from a set of stochastic order, C_r. An action/decision x_k is selected to modify a route assigned with the new order(s). Delivery later than the time window upper bound incurs a penalty cost per unit time violated. The objective is

Table 3.1: Set of notations used in the same-day delivery routing problem.

Notation	Description
M	Set of identical vehicles, $M \in \{1, ..., M\}$ Depot is set as location 0.
$\delta_m(k)$	A route or sequence of remaining locations to visit by vehicle m at decision epoch k.
$\delta_m^x(k)$	A route or sequence of remaining locations to visit by vehicle m after executing decision x at decision epoch k.
$\beta_m(k)$	A route or sequence of locations to visit and visited by vehicle m updated as of decision epoch k.
$\beta_m^x(k)$	A route or sequence of locations to visit and visited by vehicle m after executing decision x at decision epoch k.
$vloc_m(k)$	Location of vehicle m at decision epoch k.
$t(k)$	Time at decision epoch k.
C_k	Set of realized orders updated as of decision epoch k.
C_r	Set of stochastic orders.
$CS_n(k)$	Order status of customer order n at decision epoch k.
$[e_n, l_n]$	Delivery time window at customer location for order n.
$\tau(\delta_m(k))$	Total travel time of vehicle m when following route $\delta_m(k)$.
$\tau(\beta_m(k))$	Total travel time of vehicle m when following route $\beta_m(k)$.
$wait(\delta_m(k))$	Total waiting time of vehicle m when following route $\delta_m(k)$.
$wait(\beta_m(k))$	Total waiting time of vehicle m when following route $\beta_m(k)$.
$pen(\delta_m(k))$	Total penalty cost for time windows violation of vehicle m when following route $\delta_m(k)$.
$pen(\beta_m(k))$	Total penalty cost for time windows violation of vehicle m when following route $\beta_m(k)$.

to minimize the sum of total travel and waiting times of all vehicles and penalty cost for time window violations. In this problem, we focus on insertion of new orders rather than cancellation since both warrant similar approach and insertion is more challenging and interesting than cancellation. In addition, in order to simplify discussion, we assume that a new request can be served without considering its pickup. Incorporating pickup is fairly straightforward, since we are making route changes within a single vehicle.

3.3.1.2 Model Formulation

We model this dynamic multi-vehicle same-day delivery routing problem with time windows and stochastic customers as a route-based MDP.

Decision Epoch. A decision epoch or decision point k occurs at every time step $t(k)$. This means that dynamic event can take place at any time throughout the time horizon. Thus, by this definition, k is equal to $t(k)$. This definition of decision epoch is chosen to simulate real-world environment where dynamic events can take

place at any time and also to facilitate modelling the arrival rate of these events.

State. A state of the MDP consists of two parts, pre-decision state S_k and post-decision state S_k^x. S_k captures the necessary information required such as the current time, locations of all the vehicles, the remaining routes of all the vehicles and the statuses of all the realized orders. S_k is represented as the following tuple:

$$S_k = \langle t(k), vloc(k), \delta(k), CS(k) \rangle \tag{3.1}$$

where the locations of all vehicles, $vloc(k) = (vloc_m(k))_{m \in M}$, the remaining routes of all vehicles, $\delta(k) = (\delta_m(k))_{m \in M}$ and the order statuses of all orders, $CS(k) = (CS_n(k))_{n \in C_k}$. The post-decision state S_k^x captures the changes to the state upon executing a decision. $t(k)$ remains the same while the other three components are updated depending on the decision taken.

Action/Decision. x_k at decision epoch k is the action of updating the remaining route of the vehicle which is assigned to serve the new order. The route of the chosen vehicle m, $\delta_m(k)$ is revised to $\delta_m^x(k)$ after executing $x_k \in X(S_k)$ where all orders should be delivered within $[e_n, l_n]$. Note that the time window is a soft constraint while lunch hour period is a hard constraint. We also assume that there is no swapping of customer orders among vehicles as the load picked up for a particular customer order must be delivered to the respective customer.

Transition. There are two main transitions in the model namely, from pre-decision state, S_k to post-decision S_k^x and from S_k^x to the next pre-decision state, S_{k+1}. The transition from S_k to S_k^x has been mentioned in earlier. Meanwhile, during transition from S_k^x to S_{k+1}, a realization of new order, ω takes place, and C_{k+1} is updated by adding the new order to C_k and $S_{k+1} = (S_k^x, \omega)$ where $\omega \in C_r$.

Reward Function. The *reward* function, $R(S_k, x_k)$ is defined as the incremental increase in total cost of being in state S_k and choosing decision x. $Cost(S_k, x)$ is defined as the total cost (i.e. travel plus wait plus time window violation) when choosing decision x at state S_k. Figure 3.3 illustrates how this reward function is

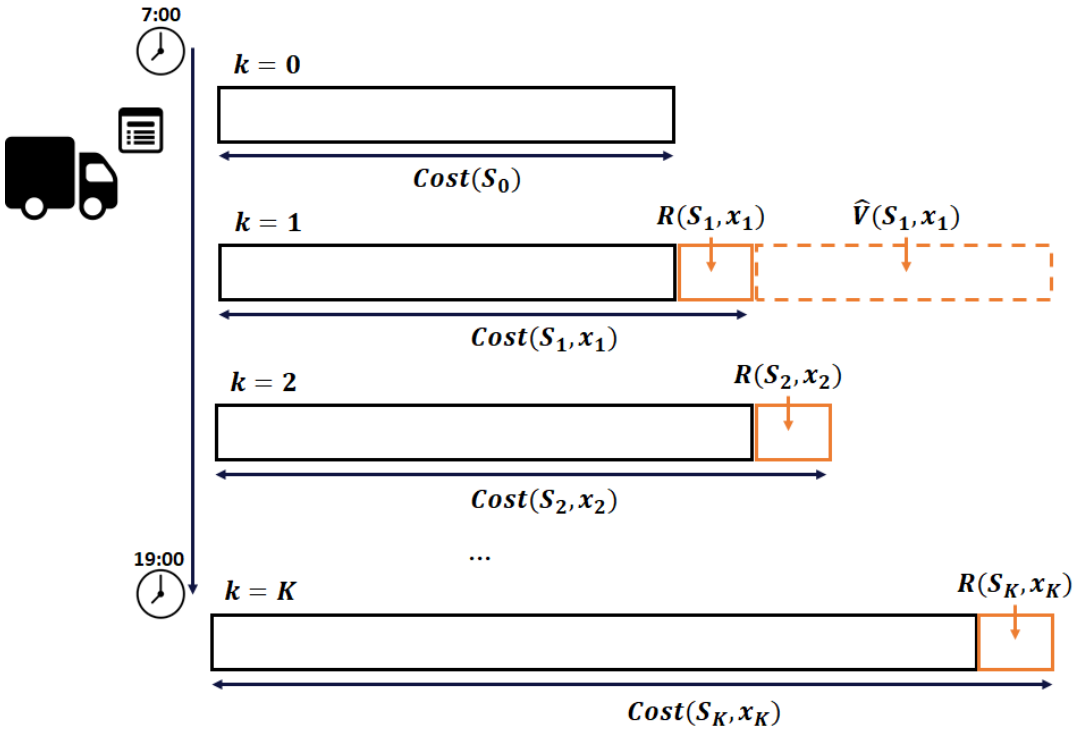


Figure 3.3: Illustration on how the reward function and approximate value function are derived.

derived.

$$Cost(S_k, x) = \sum_{m=1}^{M} \tau(\beta_m^x(k)) + wait(\beta_m^x(k)) + pen(\beta_m^x(k)) \tag{3.2}$$

$$R(S_k, x_k) = Cost(S_k, x_k) - Cost(S_{k-1}, x_{k-1}) \tag{3.3}$$

Value Function. The value function equation, $V(S_k^{x^*})$ can be approximated to the following Bellman Equation:

$$\hat{V}(S_k^{x^*}) = min_{x \in X(S_k)}\{R(S_k, x) + \gamma \hat{V}(S_k^x)\} \tag{3.4}$$

The goal is to minimize the expected future cost over the planning horizon. In other words, given a state S_k, select decision x that returns the minimum $\hat{V}(S_k^{x^*})$. As shown in Figure 3.3, the approximated value function (for e.g. $\hat{V}(S_1, x_1)$) would take into consideration expected the future rewards from yet-to-realized dynamic events.

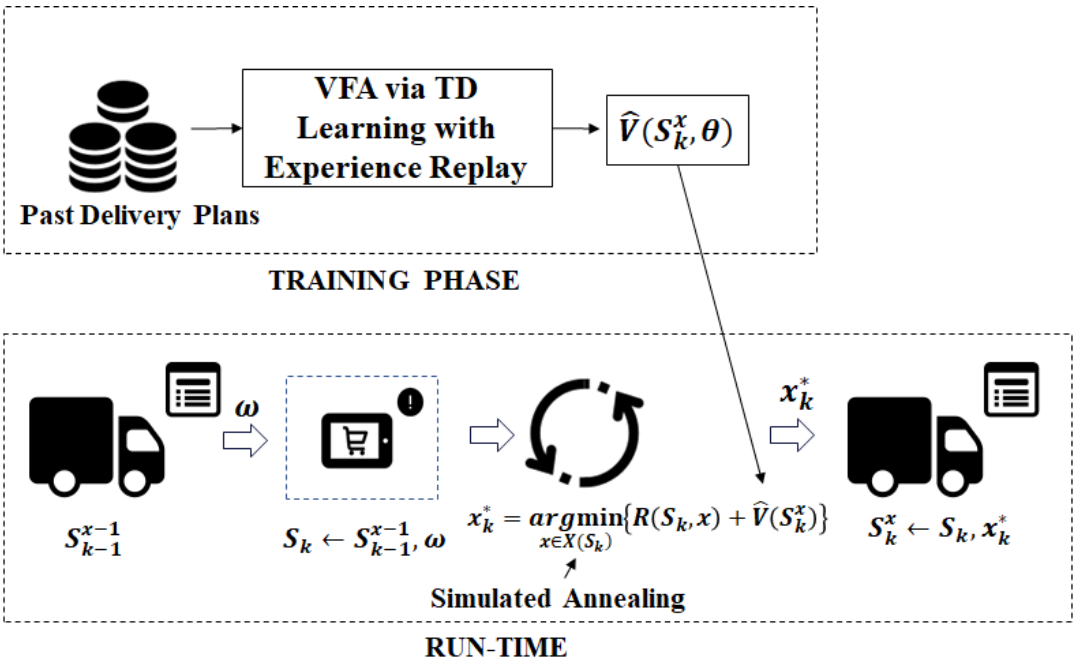


Figure 3.4: Framework of the proposed approach, DRLSA with SA to compute rerouting decision.

3.3.2 Solution Approach: DRLSA

We refer our proposed solution approach as Deep Reinforcement Learning with Simulated Annealing (DRLSA) as we adopt SA to be the rerouting heuristic (see Figure 3.4). The learnt value function approximate is subsequently utilized by a SA algorithm to generate the revised routes during run-time. Due to large state space, a state representation based on the cost of the remaining routes of vehicles and current time is used to capture the spatial and temporal attributes of the state.

3.3.2.1 Value Function Approximation

During the training phase, we use a set of historical delivery plans to simulate the initial delivery plans and, depending on the value of Degree of Dynamism (DoD), a percentage of the orders in the plan is randomly removed and these orders are added subsequently as realizations of new orders. This is done to simulate dynamic events. This offline simulation outputs the approximated value function for each post-decision state, $\hat{V}(S_k^x, \theta)$ in a form of neural networks (see Figure 3.5). The bulk of the computational time of this approach is during this training phase; and during execution, the run time is spent on SA to generate the changes to the routes.

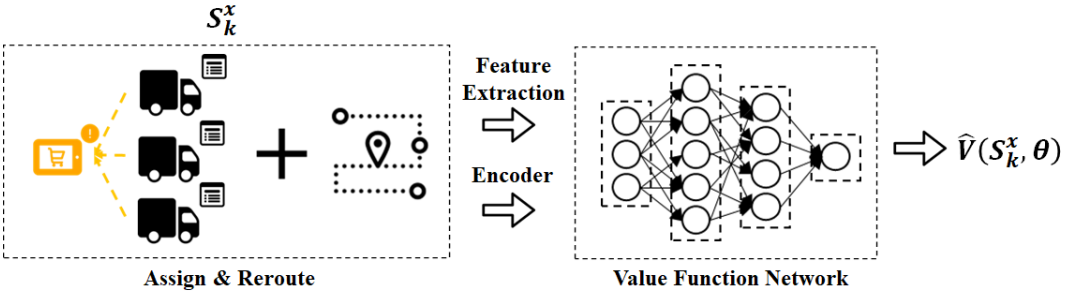


Figure 3.5: Approximate value function that incorporates both assignment and routing decisions.

3.3.2.2 Routing Optimization via SA

During the run-time phase, a set of scenarios of daily delivery plans and dynamic realizations of stochastic new orders are presented. Our approach utilizes the approximate value function from the training phase to compute value of each decision explored during the SA search. As shown in Figure 3.4, x_k^* is modified using SA taking into account both immediate cost plus expected future reward computed by the approximate value function, $\hat{V}(S_k^x)$. We note that many routing heuristics can be used to optimize the routes. In this work, we picked SA due to its efficiency and effectiveness to provide fast quality solution for vehicle routing problems [29, 93]. The focus of the work is not to find the best heuristics for routing, but a flexible solution framework which enables different heuristics to be used in optimizing the routes.

3.3.2.3 State Representation

To exploit the structure of this problem, we propose a state representation based on the total cost of the remaining routes of the vehicles and the current time at the point of decision-making. We show that the cost of the remaining routes of vehicles can serve as proxy to the sequence of the routes and time window requirements. This state representation captures both spatial and temporal features which impact decision-making.

Our proposed state representation consists of the current time and the cost of the remaining routes of the vehicles at decision epoch k which includes the penalty cost for time window violations for each vehicle. The proposed state representation is shown below:

$$S'^{x}_{k} = \langle t(k), Cost_{remain}(S_k, x), Cost_{penalty}(S_k, x) \rangle \tag{3.5}$$
$$\text{where } Cost_{remain}(S_k, x) = (Cost_{remain,m}(S_x, k))_{m \in M}$$
$$Cost_{(remain,m)}(S_k, x) = \tau(\delta^x_m(k)) + wait(\delta^x_m(k))$$
$$Cost_{penalty}(S_k, x) = (Cost_{penalty,m}(S_k, x))_{m \in M}$$
$$Cost_{(penalty,m)}(S_k, x) = pen(\delta^x_m(k))$$

Time. $t(k)$ is an important temporal feature in the problem because the earlier the decision point during run-time, the more likely there will be new stochastic requests in the future and thus, the anticipatory value for future rewards is higher. The same principle applies when $t(k)$ happens later during run-time.

Cost of Remaining Routes. The rerouting decision to the existing route does not depend on the previous visited locations prior to decision point k. This is because the immediate and future rewards only depend on the remaining routes.

Specifying the exact remaining routes of the vehicles, $\delta(k)$ may result in a very large state space. The proposed state representation uses the cost of remaining routes, $Cost_{remain}(S_k, x)$ as a proxy to for the routes. For example, the cost of route $[3, 4, 5, 0]$ is different from $[3, 5, 4, 0]$ and this sufficiently differentiate the routes. There may be scenarios where the cost of two routes may be the same even though the exact sequences of the routes are different. However, this is not a concern since rerouting decision is dependent on the immediate reward and the post-decision state which are different in both scenarios. For example, inserting Customer 6 at the second position (i.e. $[3, \mathbf{6}, 4, 5, 0]$ and $[3, \mathbf{6}, 5, 4, 0]$ respectively) will result in different immediate rewards and post-decision states in both scenarios.

Penalty Cost. State representation also needs to capture the time window requirements. Assuming two different new orders with the same location but different time windows requirements, inserting these two orders at the same exact position in a route may result in different reward and post-decision state. This is because the penalty costs for the time window violations are different. Thus, the total penalty cost can serve as a proxy for the time window requirement.

3.3.3 Experiments

The objective of the experiment is to evaluate the performance of DRLSA against other existing algorithms, both offline and online. We evaluate these approaches based on how much improvement they achieve over the pure re-optimization method i.e. myopic approach.

3.3.3.1 Benchmark Algorithms

Approximate Value Iteration. Algorithm 7 in Appendix A.1 shows the AVI algorithm adapted from [124] for the problem setting of this problem. Cumulative observed rewards are used to approximate the future expected rewards of a state (lines 12 and 16). We use SA as routing heuristic to compute the arg min in line 7 and same state representation for a fairer comparison.

Multiple Scenario Approach. MSA with consensus algorithm used in this experiment is adapted from [8, 9]. MSA does not have training phase and is directly applied during the run-time. The main idea of this algorithm is to find the optimal route at every decision point by sampling lookahead scenarios and compute the decision that returns the lowest average total cost across the samples.

Algorithm 8 in Appendix A.2 details how this algorithm is applied during the run-time. At every decision point k, J samples of future new requests are collected (line 8). For every sample, an optimal route is calculated assuming that all the future new requests until $k + H$ are known at k (line 9). We also use SA as the routing heuristic to compute the optimal route. The resulting optimal route from each sample is stored with the sampled future new orders removed (line 10). This will represent one possible rerouting decision. Across many samples, unique rerouting decisions are stored and the average total cost is calculated for every unique decision or route (line 11). Route with the lowest average total cost will be selected as the best decision (line 14).

Myopic Approach. This approach is simply choosing a decision that gives the minimal immediate total rewards. SA is used here to compute the rerouting decision.

3.3.3.2 Experiment Design

We use 2-month's worth of historical delivery data from a local Logistics Service Provider (LSP) containing 48 customer locations. The data contains the daily delivery plans generated by an optimization algorithm that minimizes travel time, wait time, make span (the amount of time the vehicle is out during the delivery) and penalty cost due to time window violations. The delivery data is split into training (34 days) and test sets (10 days). 2 vehicles are used for this experiment with an average of 22 daily orders.

For every test scenario, a random daily delivery plan from the test set is picked as the initial routes of the vehicles. Depending on DoD, a percentage of the orders in the initial plan is randomly removed and are added subsequently during the simulation; following a Poisson process with $\lambda = 1.75$ as the rate of occurrences of dynamic orders per hour. The spatial distribution of customers requesting these new orders are based on probability distribution derived from the historical data. We need to specify this distribution in running MSA and not for DRLSA or AVI as these two methods will learn the underlying distribution during training.

Performance Measure. We use the percentage improvement in terms of the final total cost that the approaches (DRLSA or AVI or MSA) can achieve over myopic as the performance measure.

$$\%improve_{DRLSA} = \frac{Cost_{DRLSA}(S_K) - Cost_{myopic}(S_K)}{Cost_{myopic}(S_K)} \times 100\% \tag{3.6}$$

By this definition, the tested approaches must achieve a negative % improvement if they are to result in lower cost. For simplicity's sake and consistency with other works in the literature, we report reduction in cost as positive value and increase in cost as negative value.

Experiment Setup. The experiment consists of 3 phases. The 3 phases of experiment are as follow:

- **DRLSA vs. AVI.** We use 3 different values of DoD, $0.3, 0.5$ and 0.7. DoD refers to the percentage of total orders that are dynamic. For each DoD, we ran 20 experiment runs. Each experiment run consists of 50 daily test scenarios.

- **DRLSA vs. MSA.** We use only $DoD = 0.7$ with 100 test scenarios and sample size, $J = 50$ and 100.
- **Further Experimentations on DRLSA.** We test DRLSA with larger DoDs and larger problem scale.

The implementation codes are written in Python while PyTorch is used to build and train the neural networks. The experiments are run on a local machine with the following configurations: Ubuntu 18.04 with 6 CPU Cores and 16GB RAM. The detailed descriptions of the simulator used to run the model can be found in Appendix A.5.

Size of State Space. Based on the state representation in Equation 3.3.2.3, the size of the state space based on the experiment settings can be estimated as follows:

- *Time.* Each planning horizon spans from 7am in the morning to 7pm midnight. This translates to 720 minutes.
- *Cost of Remaining Routes.* Assuming a vehicle travel non-stop for the whole of planning horizon, a route has a maximum cost of 720 minutes corresponding to the duration of the planning horizon.
- *Penalty Cost.* Similarly, the total penalty cost represented by the total waiting time and time violation has an upper bound value of 720 minutes.

Thus, the size of state space in our experiment can be estimated to be $720^{1+2\times|M|}$. However, the exact size of state space will be smaller since the cost of remaining routes and penalty cost are not mutually exclusive and must add up to 720.

Model and Training Setup. DRLSA is a fully-connected neural network with 2 hidden layers with 64 nodes and 32 nodes respectively with a total of 2497 trainable parameters (see Figure 3.6 for the detailed breakdown). Meanwhile, the dimension of the input features are 5 assuming 2 vehicles. The input dimension size of 5 corresponds to the following state representation:

$$S_k'^x = \langle t(k), Cost_{remain,1}(S_k, x), Cost_{penalty,1}(S_k, x), \\ Cost_{remain,2}(S_k, x), Cost_{penalty,2}(S_k, x)\rangle \tag{3.7}$$

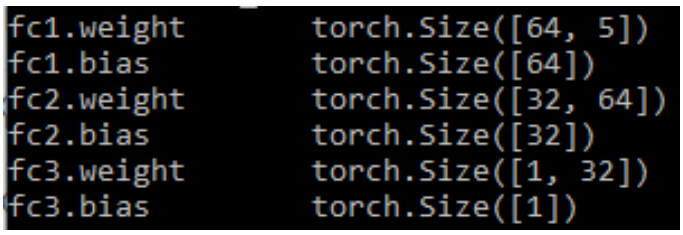


Figure 3.6: The detailed breakdown of the sizes of the trainable weights at each layer. *fc* denotes a fully-connected layer. The sum of all the weights is 2497.

We use ReLU as activation function and Adam optimizer to train the networks. We experimented on various ranges of parameters such as number of hidden layers, nodes, batch size, update and learn frequencies and learning rate. We empirically evaluated and found that the following parameters resulted in the best performance in terms of average total cost: batch size = 20, learn and update frequencies of once in every 5 steps, learning rate = 0.001 and $\gamma = 0.99$. The value of ϵ in the ϵ-greedy step is set to be decreasing as the number of training episode increases. Stochastic Gradient Descent is used as the optimizer. For AVI, we use a constant value of $\alpha = 0.1$. For MSA, we set H as large number to simulate that all future requests are known. SA is the default routing heuristic with the following dynamic cooling function, $T = T_{max} \times e^{-f(\frac{step_{current}}{step_{max}})}$ where $step_{max} = 2500$ and $T_{max} = 25000$.

VFA via non-parametric function has an advantage over the table-based ones because it may not need to observe all possible states and approximate the function based on the observed ones. We observe that for DRLSA, the average % improvement over myopic in the last 100 episodes begins to stabilizes after 2500 training episodes (see Figure 3.7). Thus, we set the number of training episodes for DRLSA to be 3500 and 35000 for AVI. We show that even with less training episodes, DRLSA is able to outperform AVI.

Assumptions. There are several assumptions used in modelling the DVRP environment. There is no capacity constraint, no new order after certain designated time and no swapping of loads among vehicles once assigned. New orders can be loaded from another customer locations so no depot returns are required. Vehicle which is on the way to the next customer order needs to fulfill the order first. Travel and service times are assumed to be static.

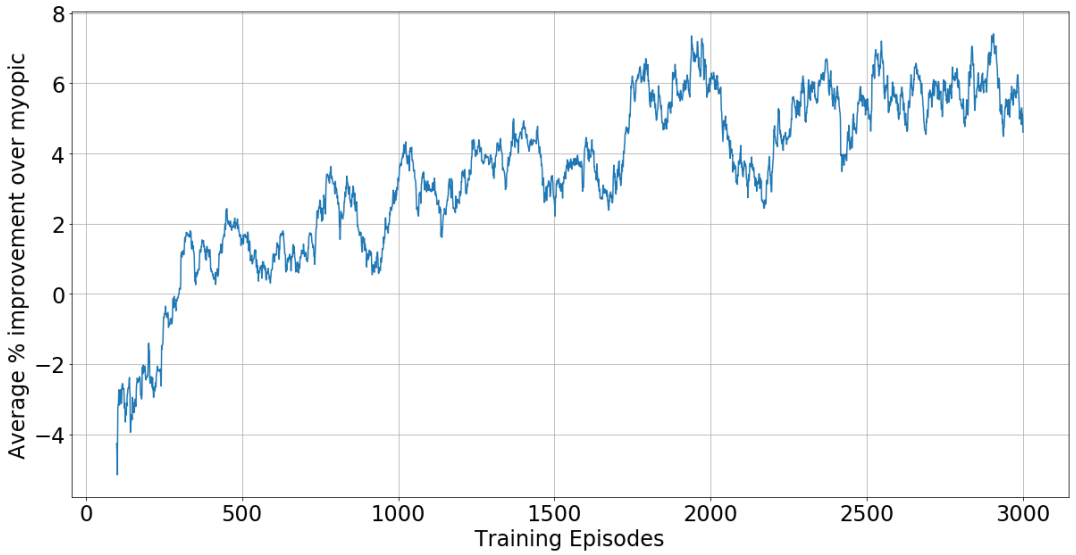


Figure 3.7: Average % improvement over myopic in last 100 training episodes.

Table 3.2: Average % of improvement of DRLSA vs. AVI over different DoDs.

	Method	**0.3**	**0.5**	**0.7**
Average% of improvement	DRLSA	-1.95%	-0.98%	11.90%
over 20 experiment runs	AVI	0.24%	-1.25%	-1.51%

3.3.3.3 Experiment Results

DRLSA vs. AVI. Table 3.2 provides the summary of the phase one results. The proposed approach outperforms AVI for $DoD = 0.7$, achieving on average 12% improvement over myopic. Figure 3.8 shows the average performances of DRLSA and AVI over myopic over 20 experiment runs. Although the training time of DRLSA is 4 times longer than AVI (DRLSA takes about 36 seconds/episode compared to AVI which takes about 8 seconds), the computation time during run-time is almost instantaneous ($<$ 10 seconds/decision) for both.

We observe that AVI performs quite poorly for every DoD. This can be due to the fact that the training episodes may not be sufficient for the algorithm to approximate most or all possible state representations. AVI approximates value of states that have never been encountered during training as 0 and this is equivalent to myopic approach. It is no surprise that AVI results in near 0% improvement. DRLSA does not perform better than myopic when $DoD \leq 0.5$. This is expected as lower DoD means new stochastic orders are less likely to occur and anticipating a

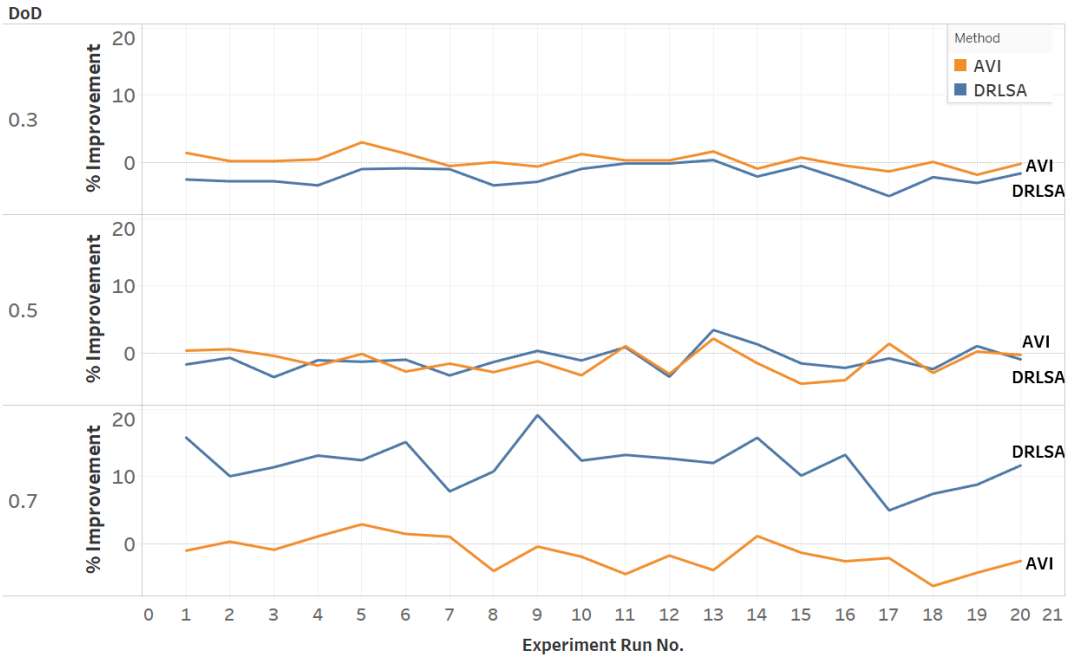


Figure 3.8: Average % of improvement of DRLSA vs. AVI over different experiment runs.

rarely occurring event becomes more challenging (which perhaps cannot be learnt). Nevertheless, this experiment is able to show that DRLSA is able to achieve good result even with small training episodes and with higher DoDs.

DRLSA vs. MSA. DRLSA achieves an average of 9.6% improvement over 100 test scenarios, outperforming MSA even with $J = 100$ and it is more than 10 times faster in terms of computation time (see Table 3.3). MSA needs a very large sample size in order to evaluate the rerouting decisions. To sample the instances of future new requests in the next time horizon H, it needs to take into consideration a combination of the locations of those requests, the arrival time of the requests and the corresponding time window requirements. Therefore, the performance of online approach like MSA depends on how good the stochastic model is in simulating the future events. On the other hand, offline, learning-based approach such as DRLSA is able to learn the underlying probability distribution of the dynamic events based on historical data.

To our knowledge, there is no study in the literature that compares performance between offline and online decisions approaches and no set of benchmark test

Table 3.3: Average % of improvement and computation time per scenario for DRLSA vs. MSA.

	Average% of Improvement 100 Test Scenarios	**Average Computation Time per Scenario (in mins)**
DRLSA	9.61%	< 1
MSA ($J = 50$)	1.73%	3
MSA ($J = 100$)	2.20%	10

Table 3.4: Performances of DRLSA for increasing *DoD*s.

	Method	**0.6**	**0.7**	**0.8**
Average % of improvement over 20 experiment runs	DRLSA	1.92%	11.90%	12.72%

parameters available. Nevertheless, based on our experiment, to outperform DRLSA, MSA needs $J >> 100$ and even longer computation time.

Further Experimentation on DRLSA. We evaluate the performance of DRLSA over higher *DoD*s and are able to show that DRLSA indeed performs better with increasing *DoD*s (see Table 3.4).

To evaluate its scalability, we did further experiments with 3 and 4 vehicles with average total daily orders of 30 and 40 respectively, The available datasets only allow us to scale up to 4 vehicles with 40 total daily orders. Based on 20 experiment runs, DRLSA achieves an average of 15.71% and 20.88% improvements over myopic respectively.

3.3.3.4 Experiment Discussion

Ritzinger *et al.* [102] summarized the performances of various successful pre-processed decision methods in the literature. For larger-sized customers (> 30), most approaches managed to achieve improvements in the region of $5\% - 10\%$ over myopic. Our experiment shows that the performance of DRLSA is comparable with those cited in that study. However, we note that the experiment setups across the papers may be different.

This experiment also shows that our proposed approach, DRLSA is able to outperform both AVI and MSA even with a relatively small number of training episodes. AVI's poor performance also reiterates our hypothesis that AVI may not work well in complex problems with large state spaces (due to multi-vehicle and multi-

constraints settings) even with state aggregation as a very large number of training episodes (in the magnitude of millions) are required to reasonably approximate the value function of most of the states. The experiment also shows that offline approach like DRLSA is more suited than the online counterparts like MSA for problem setting that requires instantaneous decision-making such as same-day delivery operations.

3.4 Application 2: Dynamic Bi-Objective Police Patrol Dispatching and Rescheduling Problem

DPRP, although similar to the first problem, poses slightly different and additional challenges in a different application domain. The problem deals beyond mere routes but schedules of the routes while managing dual, conflicting objectives.

3.4.1 Problem Description and Model

Table 3.5 provides the set of key notations and the corresponding descriptions used in this problem.

3.4.1.1 Problem Description

In a given police sector, there are $|I|$ patrol agents in-charge of patrolling $|J|$ patrol areas in a shift with a duration of $|T|$ time periods. At the start of the shift, each patrol agent is assigned to an initial patrol schedule. Throughout the shift, incidents occur dynamically and a patrol agent is dispatched to respond to each incident which results in the need to reschedule the initial schedules so as to still fulfill both objectives of projecting patrol presence and meeting the response time target. We can observe that this problem shares many similarities with DVRP [36].

Decision Epoch. A decision epoch k occurs whenever an incident occurs and decisions to dispatch agent to respond to the incident and to reschedule the existing schedules are required. $k = 0$ indicates the start of the shift.

Schedule. Each patrol schedule includes the sequence of patrol areas to visit (routes) and when and how long to patrol each areas (schedule). A sample joint patrol schedule can be found in Figure 3.9. It is similar to a university time-table

Table 3.5: Set of key notations used in DPRP.

Notation	Description
I	A set of patrol agents, $I \in \{1, 2, ..., \|I\|\}$.
J	A set of patrol areas, $J \in \{1, 2, ..., \|J\|\}$.
T	A set of time periods in a shift, $T \in \{1, 2, ..., \|T\|\}$.
k	Decision epoch
t_k	Time period in a shift where decision epoch k occurs, $t_k \in T$.
$\delta_i(k)$	A schedule of patrol agent i at decision epoch k where $\delta_i(k) = (j_1, j_2, ..., j_t, ..., j_{\|T\|})$ where $j_t \in J \cup \{-1\}, t \in T$.
$\delta(k)$	A joint schedule of all patrol agents at decision epoch k where $\delta(k) = (\delta_1(k), \delta_2(k), ..., \delta_{\|I\|}(k))$.
$\delta_{-i}(k)$	A joint schedule of all patrol agents except for agent i at decision epoch k where $\delta_{-i}(k) = (\delta_1(k), ..., \delta_{i-1}(k), \delta_{i+1(k)}, ..., \delta_{\|I\|}(k))$.
$\delta_i^x(k)$	A schedule of patrol agent i after executing action x at decision epoch k.
$\delta^x(k)$	A joint schedule of all patrol agent after executing action x at decision epoch k.
τ_{target}	A response time target.
τ_{max}	A maximum tolerable buffer time from the moment incident occurs to the moment an agent acts upon the dispatch call.
τ_k	Actual response time to incident at epoch k.
$D_h(\delta', \delta)$	Hamming distance (in %) between schedules δ' and δ.
P_{max}	Maximum allowable perturbation (in %) from the initial schedule.
$d(j, j')$	Travel time from patrol area j to another patrol area j'.
Q_j	Minimum patrol time (in terms of time period) for patrol area j.
$\sigma(k)$	Patrol statuses of each other patrol area in terms of the ratio of the effective patrol time over Q_j where $\sigma(k) = (\sigma_j(k))_{j \in J}$.

with an additional key constraint whereby in between two different patrol areas, there must be sufficient time periods to cater for travel time. At the start of the shift, each patrol agent is assigned to an initial patrol schedule, $\delta_i(0)$ that forms a joint schedule, $\delta(0)$. In this paper, we assume that the initial joint schedule is available and computed independently.

Incident. A dynamic incident, ω_k occurs at decision epoch k and is described as the following tuple: $\langle \omega_k^j, \omega_k^t, \omega_k^s \rangle$ where $\omega_k^j \in J$ refers to the location of the incident, $\omega_k^t \in T$ refers to the time period when the incident occurs and ω_k^s refers to the number of time periods needed to resolve the incident. We assume deterministic resolution time.

	Time Period							
Agent	**1**	**2**	...	$t-1$	t	$t+1$	...	$\|T\|$
1	*1*		...		*5*		...	*9*
2	*3*		...	*6*			...	*2*
3	*8*		...	*4*			...	*1*

	Travelling / Unavailable
<Patrol Area j ∈ J>	**Patrolling**

Figure 3.9: A sample joint schedule, $\delta(k)$ assuming $|I| = 3$.

Patrol Presence. We define patrol presence in terms of the number of time periods each patrol area is being patrolled in a shift. Each patrol area j needs to be patrolled for at least Q_j time periods in a given shift. We define a fitness function, $f_p(\delta(k))$ to quantify the goodness of a given schedule $\delta(k)$ in terms of its ability to project presence. $f_p(\delta(k))$ consists of two components namely patrol utilization (ratio of effective patrol time over total shift time) and penalty cost for failure to meet the minimum patrol time. Thus, a schedule is deemed to have good patrol presence if agents spend most of the time patrolling rather than travelling between patrol areas and each patrol area is being patrolled for at least the required number of time periods.

$$f_p(\delta) = \frac{\sum_{i \in I, t \in T} 1_J(\delta_{i,t}) - \sum_{j \in J}(Q_j(1 - min(\sigma_j, 1)))}{|T| \times |I|} \tag{3.8}$$

$$\text{where } 1_J(x) = \begin{cases} 1, x \in J \\ 0, x \notin J \end{cases}$$

Response Time. The response time to an incident at decision epoch k, τ_k is computed as the time taken by the assigned agent, x_k^i to act upon the dispatch call from the point where incident occurs $(x_k^t - \omega_k^t)$ plus the travel time from its current location to the incident location. A successful incident response happens when $\tau_k \leq \tau_{target}$. We assume that any dispatch call must be acted upon within

τ_{max}.

$$\tau_k = (x_k^t - \omega_k^t) + d(j_{t'}, \omega_k^j) \quad (3.9)$$
$$\text{where } x_k^t - \omega_k^t \leq \tau_{max}, t' = x_k^t$$

Problem Objective. The objective of the problem is to make dispatching and rescheduling decisions at every epoch that maximize the number of successful incident responses while minimizing the reduction in patrol presence.

3.4.1.2 MDP Formulation

We model the problem as a route-based MDP. This modelling framework suits our problem since a rescheduling decision must include the updated patrol schedules instead of simply the next patrol area to visit.

State. A state of this MDP consists of two parts, pre-decision state S_k and post-decision state S_k^x. S_k captures the necessary information required to make the dispatching and rescheduling decisions. S_k is represented as the following tuple: $\langle t_k, \delta(k), \sigma(k), \omega_k \rangle$. Meanwhile, the post-decision state S_k^x captures the changes to the state upon executing a decision. t_k remains the same while the other three components are updated depending on the decision taken.

Action/Decision. x_k is the action of dispatching an agent to an incident and updating the joint schedule at decision epoch k. x_k is represented as the following tuple: $\langle x_k^i, x_k^t, \delta^x(k) \rangle$ where $x_k^i \in I$ is the agent assigned to respond to the incident, $x_k^t \in T$ the time period which the assigned agent starts to act upon the dispatch call and $\delta^x(k)$ the resulting joint schedule after executing action x_k.

In a real-world operational setting, the disruption to the initially-planned schedule must be minimized. We propose the use of Hamming distance to quantify the extent of disruption to the original schedule and this distance must be within a given threshold, P_{max}.

$$D_h(\delta^x(k), \delta(0)) < P_{max} \quad (3.10)$$

Transition. There are two main transitions in the model namely, from pre-decision state, S_k to post-decision S_k^x and from S_k^x to the next pre-decision state, S_{k+1}. The

transition from S_k to S_k^x takes place after executing action x_k. Meanwhile, during transition from S_k^x to S_{k+1}, a realization of a dynamic incident, ω_{k+1} takes place and $S_{k+1} = (S_k^x, \omega_{k+1})$.

Reward Function. The reward function, $R(S_k, x_k)$ is designed in such a way that high reward is given to a successful incident response while minimizing the reduction in patrol presence at the same time. We introduce $f_r(x_k)$ to quantify the success of an incident response when executing x_k.

$$R(S_k, x_k) = f_r(x_k) \times f_p(\delta^x(k)) - f_p(\delta(k)) \tag{3.11}$$

$$f_r(x_k) = \begin{cases} 1, \tau_k \leq \tau_{target} \\ 0.5, \tau_k > \tau_{target} \\ 0, \tau_k = \varnothing \end{cases} \tag{3.12}$$

The last case in Equation 3.12 indicates that the incident cannot be responded to due to unavailability of agents within τ_{max} or the existing schedule simply cannot be disrupted beyond P_{max}. As this is beyond the scope of our work, we make the assumption that additional resources may be activated to ensure that all incidents are responded to.

Our proposed reward function implicitly maximizes the success rate of responding to an incident within τ_{target} while minimizing the impact to the patrol presence. Unlike the commonly adopted linear scalarization method to address multiple objectives, there is no need to manually determine a set of weights attached to each objective.

Objective Function. The objective at every decision epoch k is to select action x_k^* which maximizes the immediate reward and the expected future reward from yet-to-realized dynamic events which is represented by the approximated value function, $\hat{V}(S_k^x)$.

$$x_k^* = argmax_{x_k \in X(S_k)} \{R(S_k, x) + \gamma \hat{V}(S_k^x)\} \tag{3.13}$$

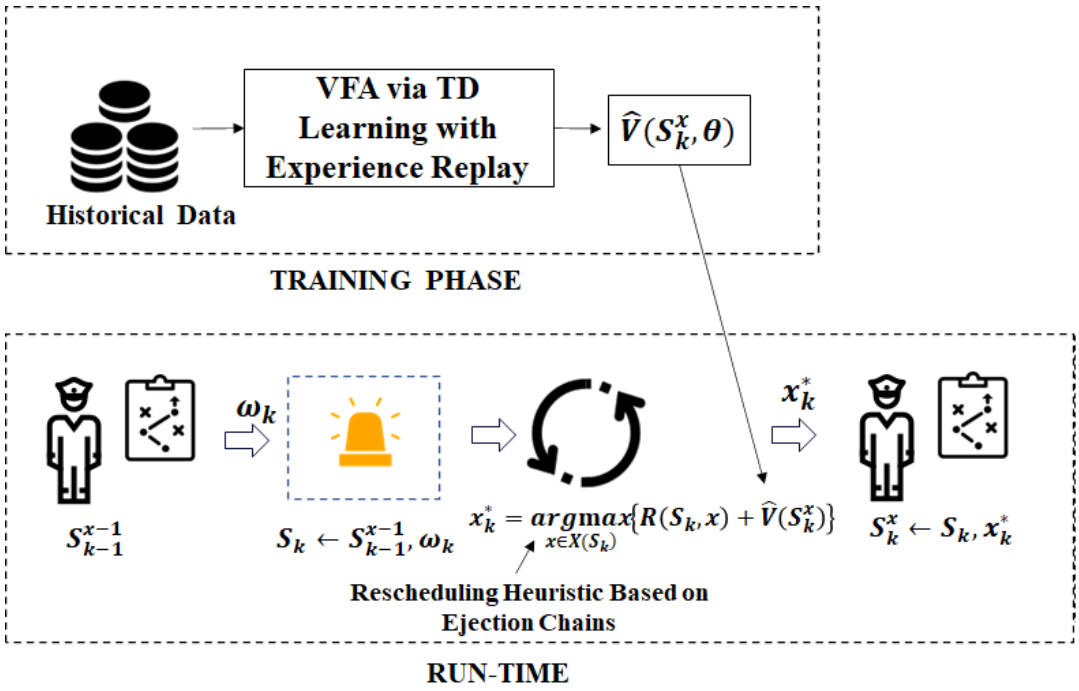


Figure 3.10: The learned value function approximation is utilised by the rescheduling heuristic to compute rescheduling decisions during run-time.

3.4.2 Solution Approach

We do not reduce the action space by decomposing the problem into two stages or two sub-problems; instead we learn the dispatching and rescheduling policies jointly by combining VFA with a rescheduling heuristic. As shown in Figure 3.10, we propose the use of rescheduling heuristic based on ejection chains to compute the rescheduling decision. The learnt value function approximate is used to guide this rescheduling heuristic.

3.4.2.1 Value Function Approximation

As shown in Figure 3.11, we approximate the value function for each post-decision state, $\hat{V}(S_k^x, \theta)$ with neural networks. The value function represents the value of a state after executing both dispatch and rescheduling decision. The proposed reward function ensures that the learned values take into account the need to maximize both objectives. In other words, given a state and an incident, we need to choose an action that results in a schedule with better incident response capability and patrol presence.

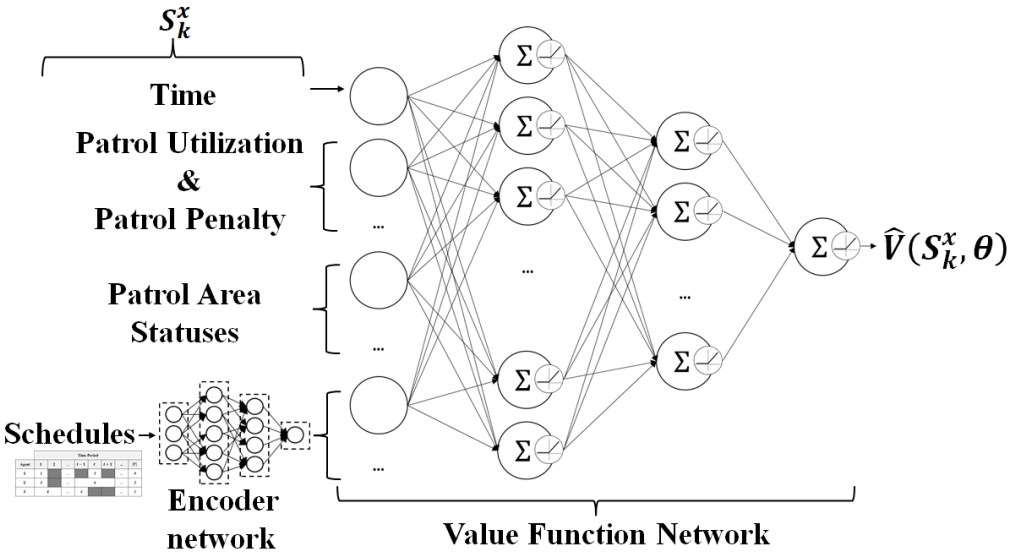


Figure 3.11: Neural network is used to approximate the value function of a post-decision state. Handcrafted features and encoded schedules are used to reduce the state space.

State Representation. To reduce the state space, we represent the joint schedule by extracting its key features and also encoding it into low-dimensional vector representations. Our encoder network is a multilayer perceptron which takes in the one-hot vector encoding of each agent's schedule. In addition, we introduce the following handcrafted key features to describe a joint schedule:

- **Patrol Utilization.** The ratio of effective patrol time over the total shift time of each agent's schedule.
- **Patrol Penalty.** Penalty cost for failure to meet the minimum patrol time.

Figure 3.11 shows how our proposed state representation enhances the learning process by supplementing handcrafted features with encoded schedules as inputs to the network.

3.4.2.2 Rescheduling Heuristic Based on Ejection Chains

To compute the rescheduling decision almost instantaneously, we propose a rescheduling heuristic based on ejection chains. Ejection chain is a complex neighbourhood structure which was first introduced in [44] and are commonly used as perturbation operator to escape local optimum. Although originally designed for TSP, ejection

chain moves have been used by many authors for various optimization problems [19, 61, 94, 101].

Ejection chain move consists of a sequence of operators that forms a chain reaction. In our problem, the ejection chain consists of a sequence of defect-checking and repair operations. Insertion of an incident into the schedule potentially introduces a defect to the schedule (see Figure 3.12). Repairing a defect at one part of a schedule may introduce a defect in another part. Thus, a chain of $CHECK$ and $REPAIR$ operations is formed until termination condition is met or until no defect is present. Two types of defects exist in our problem:

- **Type 1: Patrol Consecutiveness.** There must be sufficient time periods in between two different patrol areas in a schedule. The time periods must be at least able to cater to the travel time between the consecutive patrol areas. This is a hard constraint since the existence of this defect deems the schedule to be infeasible. There are two types of cases for this defect type namely *Insufficient* case and *Excess* case. *Insufficient* case refers to a case where there are insufficient time periods to cater for the required travel time while *Excess* refers to a case where the time periods in between two different patrol areas are more than the required travel time periods.

- **Type 2: Minimum Patrol Presence.** Each patrol area must be patrolled for at least a minimum required amount of time periods, Q_j. This is a soft constraint since the existence of this defect results in a penalty cost that reduces the goodness of a schedule.

Algorithm 2 describes how this rescheduling heuristic works for every (x_k^i, x_k^t) pair. A tabu list is introduced to avoid cycling (lines 4-9). Maximum disruption check is incorporated in the feasibility check in line 5. To speed up the heuristic, we propose to explore only one ejection chain instead of exploring multiple ejection chains at the same time with priority given to repair Type 1 defect (line 17). This simplified approach results in 5x speed-up on average with minimal impact to the solution quality.

REPAIR Operation Each ejection chain consists of a series of $CHECK$ and $REPAIR$ operations (lines 4-19). We omit the implementation details for $CHECK$

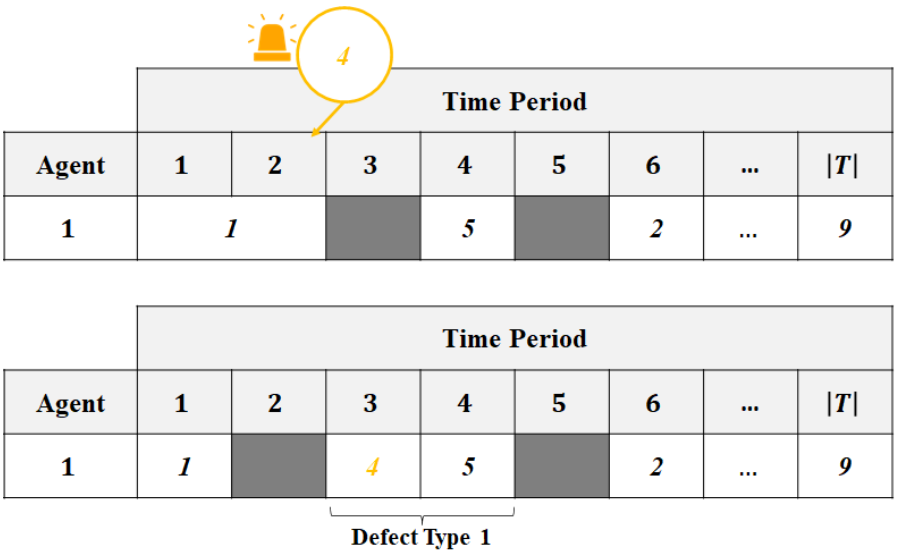


Figure 3.12: Assuming Agent 1 is dispatched to respond to an incident at patrol area 4 at time period 2 and it takes 1 time period to travel from patrol area 1 to 4 and a resolution time of 1 time unit, direct insertion of the incident into the existing schedule will create a defect to the schedule. This is unlike in the classical VRP where additional stop can be directly inserted into a route.

to simplify the discussion as it is fairly straightforward to implement it. Meanwhile, Algorithm 3 describes how the $REPAIR$ operation works. For each type of defect, we introduce corresponding $repairOperator$.

- **Repair operators for Type 1 defect.** We introduce two repair operators namely *Insert* and *Replace*. Figure 3.13 illustrates how these operators repair a schedule with Type 1 Defect (*Insufficient* case). Meanwhile, for the *Excess* case, the *Insert* operator will insert either the origin or destination patrol area to the excess time period (the illustration can be found in Appendix A.4).

- **Repair operator for Type 2 defect.** We introduce *Replace* operator which simply selects patrol areas that have extra patrol time and replace them with patrol areas that require more patrol time.

Each repair operator is akin to a local neighbourhood search which explores the neighbouring schedules by repairing the defect and return the neighbouring schedule with the highest $getScore$ value (line 7). However, we introduce an ϵ-greedy policy to explore chain that results from possibly selecting a poorer solution so as to escape possible local optimum (line 6). The learned value function is being utilized in

Algorithm 2: Rescheduling Heuristic Based on Ejection Chains

Input : Joint schedule $\delta(k)$, incident ω_k, agent x_k^i, action time x_k^t
Output : Post-decision joint schedule $\delta^x(k)$

$\delta_i^x(k)$:= Insert incident ω_k^j into agent x_k^i's schedule at time period x_k^t
$\delta^x(k) := (\delta_i^x(k), \delta_{-i}(k))$
while $\delta^x(k)$ *is defective* **do**
  if $\delta^x(k)$ *in TabuList* **then**
    if $\delta^x(k)$ *is feasible* **then**
      return $\delta^x(k)$
    else
      return None
    end
  end
  Add $\delta^x(k)$ into TabuList
  $defects := CHECK(\delta^x(k))$
  if $D_h(\delta^x(k), \delta(0)) > P_{max}$ **then**
    return None
  end
  if *defects is not empty* **then**
    Select a defect, d from $defects$
    $\delta^x(k) := REPAIR(\delta^x(k), d)$
  end
end
return $\delta^x(k)$

Algorithm 3: $REPAIR$

Input : Joint schedule $\delta^x(k)$, defect d

Create a set of neighbouring schedules, $\mathbf{N} := \{\}$
for *each repair operator* **do**
  $n := repairOperator(\delta^x(k), d)$
  $\mathbf{N} \cup \{n\}$
end
With probability ϵ, $bestSolution \sim U(\mathbf{N})$
Otherwise, $bestSolution := argmax_{n \in N} getScore(n)$
return *bestSolution*

getScore function to determine the exact neighbourhood move to take. In fact, *getScore* is equivalent to Equation 3.11 plus the learned value function and a penalty term for infeasible solution since the neighbouring schedule may still contain some defects. For example, in Figure 3.13, only n_3 has zero penalty term since n_1 and n_2 still contain Type 1 defect.

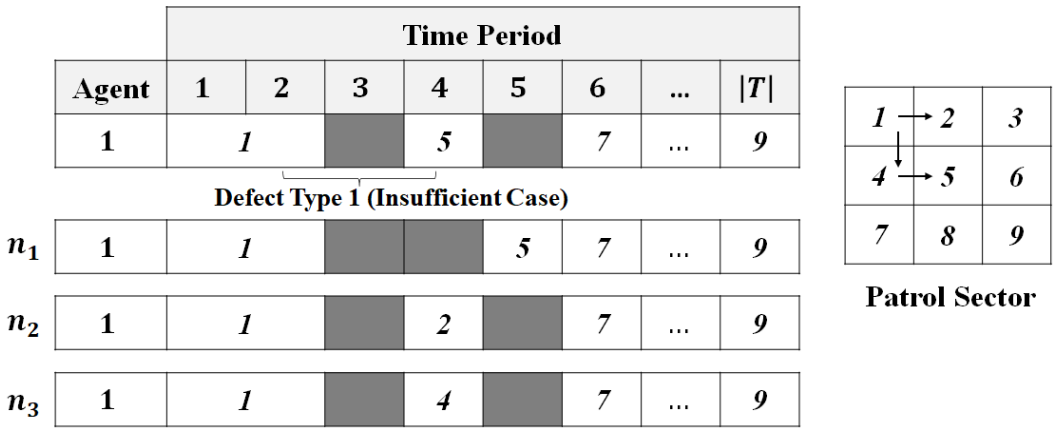


Figure 3.13: Assuming a patrol sector in the form of 3x3 grid, *Insert* operator inserts the necessary travel time into the schedule (see n_1). Meanwhile, *Replace* operator replaces the infeasible destination with the feasible ones (see n_2 and n_3).

3.4.3 Experiments

The objective of the experiment is to evaluate the solution quality and the computational time of our proposed approach against a real-world problem setting. While we try to mimic the real-world problem setting as close as possible, we evaluate our proposed approach with synthetically-generated data due to classified nature of the data.

3.4.3.1 Experimental Setup

Environment Setup. Hexagonal grids of diameter 2.22 km each are drawn over the local police sectors (see Figure 3.14). Each grid represents a patrol area. We assume that patrol agents have the flexibility to plan their own routes within a patrol area.

To evaluate the robustness of our approach, we consider 4 patrol sectors with different parameter settings and profiles to represent different problem complexities (see Table 3.6 and Figure 3.15). Sectors A and B represent a slightly less complex problem with most of the patrol areas having relatively homogeneous patrol density while C and D represent a more complex problem with more diverse patrol areas. We define patrol density as the number of minimum patrol time required in a given hexagonal grid. Meanwhile, B and D have added complexity of lower agent-to-area ratio.

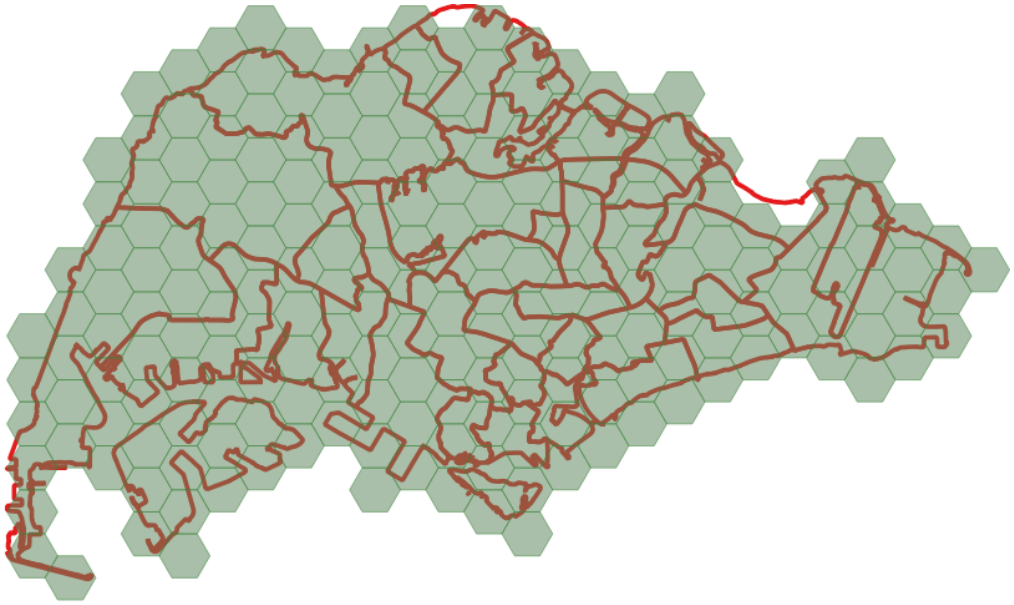


Figure 3.14: Each hexagonal grid represents a patrol area.

Table 3.6: Different patrol sectors representing different problem structures and complexities.

Sector	Parameter	Description
A	$\|I\| = 7$,	High agent-to-area ratio,
	$\|J\| = 14$	relatively homogeneous patrol densities (medium)
B	$\|I\| = 4$,	Low agent-to-area ratio,
	$\|J\| = 23$	relatively homogeneous patrol densities (low)
C	$\|I\| = 3$,	High agent-to-area ratio,
	$\|J\| = 6$	more diverse patrol densities (low to high)
D	$\|I\| = 7$,	Low agent-to-area ratio,
	$\|J\| = 23$	more diverse patrol densities low to high)

We define T as a 12-hour shift discretized to 10-minute time periods. We model the inter-arrival time of the dynamic incident using a Poisson process with $\lambda = 2$ as the rate of occurrences of dynamic incidents per hour.

Training & Testing. In the learning phase, we synthetically generate 100 samples of initial joint schedules. Each sample represents a joint schedule of a single shift in a given day. 75% of the samples are used as training data while 25% withheld as testing data. We derive the initial schedule samples by formulating and solving a mathematical model based on the static version of the problem (the model formulation can be found in Appendix A.3). During the testing phase, we run 30

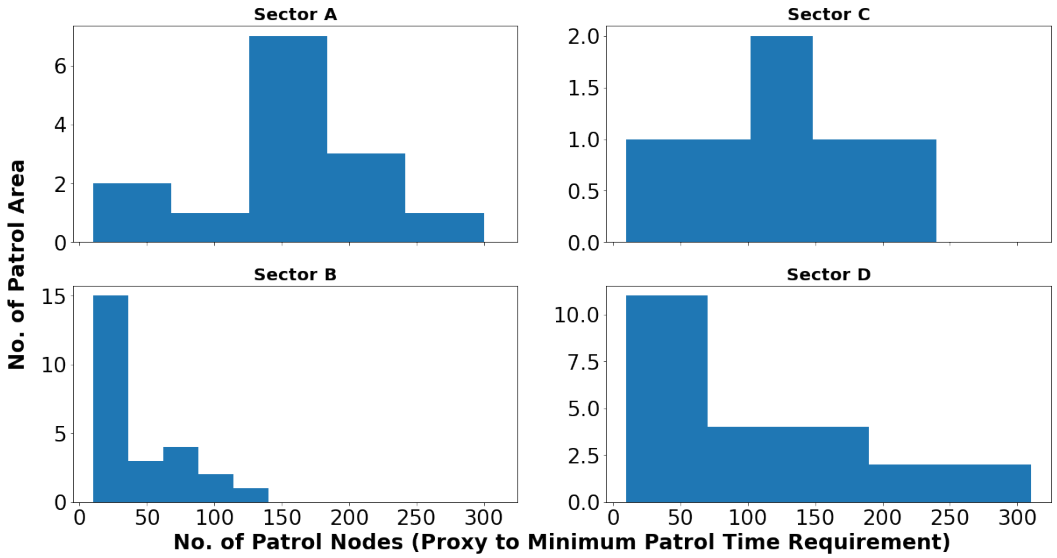


Figure 3.15: Histogram showing the structure of each patrol sector in terms of the distribution of its patrol areas by their patrol density.

experiments to simulate a month's worth of patrol schedules. For each experiment, we run 20 different realizations of dynamic incidents to simulate different scenarios that may take place in a given day.

Performance Measure. We propose to evaluate the approach based on the % improvement over myopic in terms of the success rate i.e. number of incidents that are successfully responded within τ_{target} and patrol presence in terms of its $f_p(\delta)$ value. The myopic approach is simply choosing a decision that gives the maximal immediate reward using the proposed heuristic. Instead of presenting mean value as a point estimate to represent the solution quality, we also use 95% Confidence Interval (CI) of the mean since the problem is stochastic in nature.

Baseline Models. Given that there is no prior work in solving DPRP, we propose to the following two common approaches in solving similar problem as baseline models:

- **Two-Stage.** This corresponds to the earlier-mentioned technique commonly used in many learning-based approaches. We implemented DQN as the learning algorithm to learn the dispatch policy with slight differences in the state representation since DQN approximates Q-value around S_k rather than

S_k^x.

- **Greedy.** A commonly adopted dispatching policy by human decision-makers is to dispatch the nearest available agent to an incident.

The same rescheduling heuristic and reward function are used for fairer comparison.

Size of State Space. Based on the state representation shown in Section 3.4.2.1, the size of the state space based on the experiment settings can be estimated as follows:

- *Time.* Based on our definition of T being in multiple of 10-minute time periods, the value of time ranges from 0 to 71.
- *Patrol Utilization.* For a given patrol agent, its patrol utilization values ranges from $0/|T|$ to $72/|T|$ where 0 represents a scenario whereby an agent spends all the time travelling rather than patrolling while 72 represents another extreme scenario whereby an agent remains in the same patrol area throughout the whole duration of the shift.
- *Penalty Cost.* The penalty cost for not meeting the minimum patrol time for a patrol area has an upper bound of 72 since Q_j can take up to 72 time periods.
- *Encoded Schedule.* Each agent's schedule is encoded into a vector with a dimension of 5 where each element can take a value between 0 to 1.

Thus, the size of state space in our experiment can be estimated to be $72^3 \times 10^{5\times|I|}$ assuming each element of the encoded schedule is rounded off to one decimal place.

Model Parameters. The value function network is represented as fully-connected neural network with 2 hidden layers with 64 nodes and 32 nodes respectively. Meanwhile, the encoder network is also represented as fully-connected neural network with 2 hidden layers with 128 and 64 nodes respectively. We use ReLU as activation function and Adam optimizer to train the networks. There are a total of 22806 trainable parameters for Sector A problem (see Figure 3.16 for the detailed breakdown). Here are the values of some key hyperparameters used in the model: batch size = 64, learn and update frequencies of once in every 10 steps, learning rate

```
embedding.weight          torch.Size([1, 16])
encoder.fc1.weight        torch.Size([128, 73])
encoder.fc1.bias          torch.Size([128])
encoder.fc2.weight        torch.Size([64, 128])
encoder.fc2.bias          torch.Size([64])
encoder.fc3.weight        torch.Size([5, 64])
encoder.fc3.bias          torch.Size([5])
fc1.weight        torch.Size([64, 40])
fc1.bias          torch.Size([64])
fc2.weight        torch.Size([32, 64])
fc2.bias          torch.Size([32])
fc3.weight        torch.Size([1, 32])
fc3.bias          torch.Size([1])
```

Figure 3.16: The detailed breakdown of the sizes of the trainable weights at each layer. *fc* denotes a fully-connected layer. The sum of all the weights is 22806.

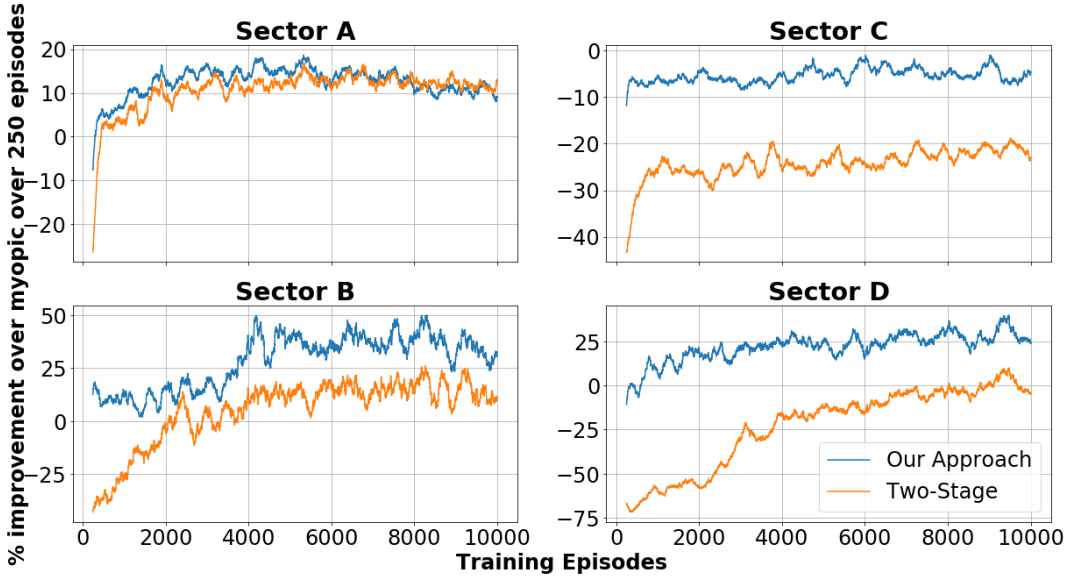


Figure 3.17: Average cumulative rewards over last 250 training episodes.

$= 0.005$, $\gamma = 0.99$ and $\epsilon = 0.3$. We set the number of training episodes for both our approach and DQN to be 10000 episodes. We observe that the cumulative rewards (represented as % improvement over myopic) stabilize after around 5000 training episodes (see Figure 3.17). The training time for our approach in on average < 15 seconds/episode which translates to about 40 hours for 10,000 episodes.

Implementation. The implementation codes are written in Python while PyTorch is used to build and train the neural networks. To generate the initial schedules (see Section A.3), Python API of any CPLEX version that is compatible to the corresponding Python version is required. The experiments are run on a local

machine with the following configurations: Ubuntu 18.04 with 6 CPU Cores and 16GB RAM. The detailed descriptions of the simulator used to run the model can be found in Appendix A.5.

3.4.3.2 Experiment Results and Analysis

Solution Quality. There are 3 main observations that we can gather from the experiments.

1. *Our proposed approach is statistically able to produce decisions that result in higher success rate as compared to the other two baseline models in 3 out of 4 sectors* (see Table 3.7). However, our approach performs poorer than myopic in Sector C. This is because C represents a very small police sector with very small numbers of agent and patrol areas. Thus, a myopic approach will suffice given that the decision space is rather limited.

2. *Our proposed approach outperforms the Two-Stage approach in more complex problems.* For ease of illustration, we intentionally present the results as line charts in Figure 3.18. We observe that when the problem scenarios are less complex (A and B), the performances of both approaches do not differ much. However, the performance gap widens with our approach outperforming the Two-Stage approach when it comes to more complex problems (C and D). Furthermore, our approach has slight edge over Two-Stage approach when the agent-to-area ratio is low (B and D). In fact, the Two-Stage approach performs rather poorly and even worse off than myopic when the problem is both diverse and the agent-to-area ratio is low (D). This may be due to the fact that in our joint learning approach, the rescheduling heuristic is also learning and adapting while, in Two-Stage approach, the rescheduling heuristic provides the reward signal to learn the dispatch policy but the learned policy does not inform how the heuristic works. Thus, there is in fact no learning in the second stage. The joint learning mechanism in our approach may be crucial in addressing more complex problem scenarios.

3. *Greedy approach is outperformed by the other models in almost all the scenarios except for B.* This shows that the common notion of dispatching the nearest

Table 3.7: Our approach statistically outperforms the other two baseline models in terms of % improvement (success rate) over myopic across 30 experiments in most sectors.

Sector	Our Approach	Two-Stage	Greedy
A	$\mathbf{18.4 \pm 1.9}\%$	$16.0 \pm 1.8\%$	$-11.7 \pm 1.7\%$
B	$36.6 \pm 9.1\%$	$24.1 \pm 10.6\%$	$\mathbf{48.5 \pm 10.4}\%$
C	$\mathbf{-5.1 \pm 1.7}\%$	$-22.4 \pm 2.5\%$	$-6.2 \pm 2.0\%$
D	$\mathbf{33.9 \pm 5.3}\%$	$0.8 \pm 4.0\%$	$10.1 \pm 5.6\%$

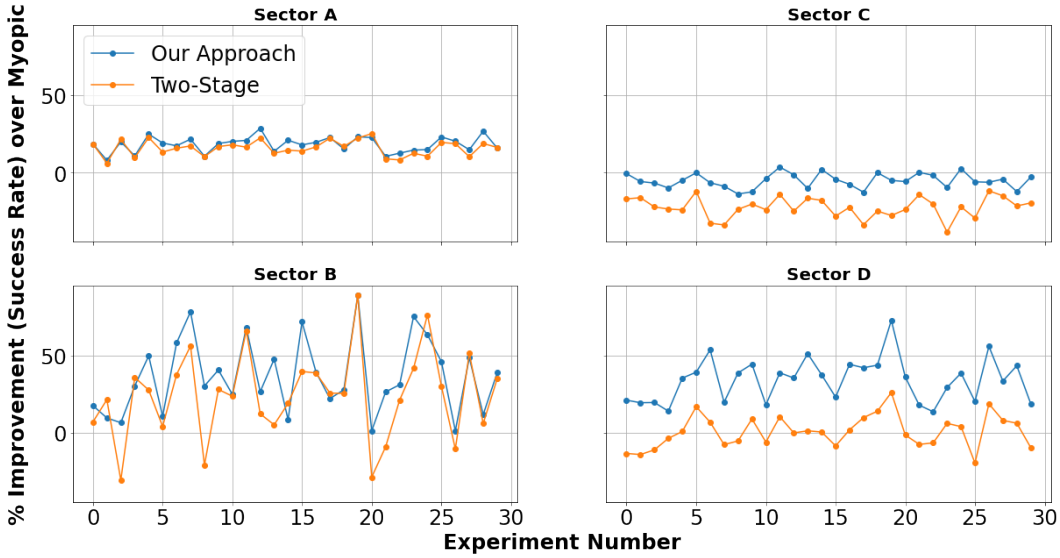


Figure 3.18: The gap in performance between our approach and the Two-Stage approach widens with more complex problem scenarios.

available agent may be sub-optimal for most cases except for some specific cases.

Computational Time. Although our proposed approach is slower than the two baseline models, it is still able to compute the decision within operationally realistic time of < 10 seconds on average across all sectors (see Figure 3.19).

The computational time may seem to be relatively long given the simplicity of the rescheduling heuristic. This is because the heuristic needs to be run for every (x_k^i, x_k^t) pair. We allow x_k^t to take values other than ω_k^t (see Equation 3.9) which means that agent may not need to respond to the incident immediately. The intuition behind this waiting strategy which myopically may not make sense, is that it allows agent to respond to another more urgent incident in the meantime or

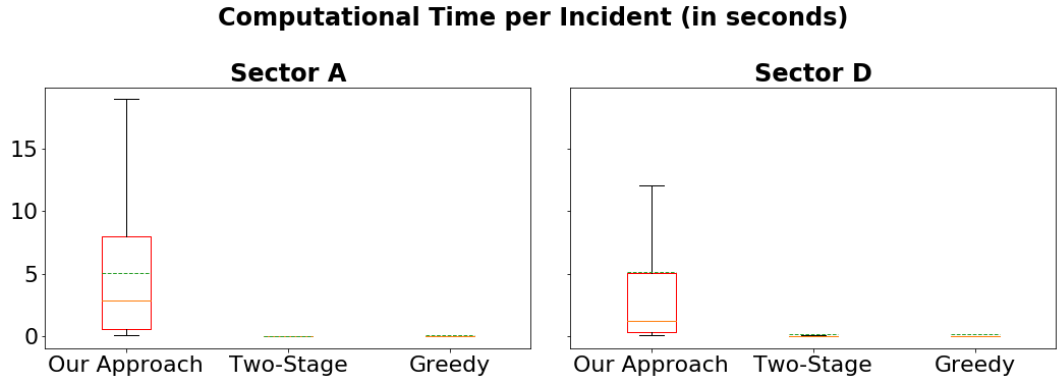


Figure 3.19: Our proposed approach, although slower, is still able to compute decision within seconds. Only results from 2 largest sectors, A and D are shown as results from the other sectors also exhibit similar trends.

by responding late to this current incident, it may result in more incidents being responded successfully later on.

Response vs. Presence. In all 3 approaches, we observe that an increase in success rate will correspondingly result in a reduction in patrol presence. To evaluate how our reward function performs in managing this trade-off, we run our approach on the same set of experiments data but with a modified reward function that is based on linear scalarization of the two objectives, $R'(S_k, x_k) = w_1 \times f_r(x_k) + w_2 \times f_p(\delta^x(k))$. In this experiment, we select two sets of weights $(w_1, w_2) = (0.5, 0.5)$ and $(0.7, 0.3)$ to represent the notions of "balance" and greater emphasis on incident response respectively; and Sectors A and D (2 largest sectors with one representing a more complex problem structure than the other).

Table 3.8 summarizes the experiment results. Comparing against the linear scalarization method, our reward function enables our proposed approach to respond to 2 and 4 more incidents while trading-off $< 5\%$ and $< 10\%$ less *proactive* patrol time in A and D respectively. This translates to an average of about 15 minutes less *proactive* patrol time per agent for 1 more successful incident response. This represents a reasonable trade-off operationally. In addition, our proposed reward function avoids the difficulty of manually assigning weights which have been shown to produce differing outcomes depending on the problem structures.

Table 3.8: Comparison of our approach with proposed vs. modified reward functions over 30 experiments. Results presented are average values across 30 experiments.

Sector	Reward Function	No. of Successful Incident Response	Patrol Presence $\mathbf{f_p}$
A	R	**16.3**	0.77
	$R'(0.7, 0.3)$	14.2	0.80
	$R'(0.5, 0.5)$	14.0	**0.81**
D	R	**14.1**	0.71
	$R'(0.7, 0.3)$	13.1	0.74
	$R'(0.5, 0.5)$	9.5	**0.80**

3.4.3.3 Experiment Discussion

We have shown experimentally that our proposed joint learning approach outperforms the popular Two-Stage approach especially when the problem scenario becomes more complex within an operationally realistic time. We have also shown empirically how our reward function is able to implicitly maximize one objective while minimizing the loss in the other without the need to manually assign weights to each objective. More importantly, our proposed approach is performing better than the greedy and myopic policies which are commonly adopted by human decision-maker in most of the problem scenarios.

3.5 Conclusion

In this chapter, we introduce a new deep RL-based solution approach that combines value function approximation step with heuristic to solve dynamic routing and scheduling problem. Our proposed approach is able to handle multi-dimensional decision variables without the need to decompose the problem into two stages. By representing the value function as neural networks, our approach is able to address problems with more complex routing constraints and large state space which characterize real-world problems. This proposed offline method is able to compute decision quickly for moderately-sized problem instances based on small training episodes in the magnitude of thousands. Our approach is also oblivious to probability distributions of the dynamic events which cannot be assumed to be known in real-world context. In terms of evaluation, we tested this approach on

two different problem domains. This approach assumes *single-agent* setting whereby a central agent controls and computes the decision. One area of future work that we are looking into would be to extend this approach to address problems with *multi-agent* setting.

Chapter 4

Coordinating Multi-Agent Routing and Scheduling

In this chapter, we propose a scalable, decentralized, coordinated planning approach to solve multi-agent routing and scheduling problems with limited shared resources. In such problem, multiple independent, non-collaborative agents or entities need to plan their routes and schedules in an environment where there are limited resources to be shared among them. Thus, there is a need for coordination to deconflict the schedules so as to maximize individual reward functions. In our approach, the problem can be formulated as a potential game where every improvement path generated by a best response procedure will converge to an equilibrium.

4.1 Motivation

In this chapter, we are addressing Multi-LSP VRP with Location Congestion (ML-VRPLC), a new variant of multi-agent VRP where there is no prior work done. Given that the LSPs in ML-VRPLC are considered as *loosely-coupled* agents, the approach to solve ML-VRPLC will be somewhere in between cooperative and non-cooperative domains of Multi-Agent Planning (MAP), although it tends to lean more towards the non-cooperative domain since LSPs are still largely independent and self-interested. Our proposed approach includes certain elements that are discussed above such as fictitious play and best-response planning. Nevertheless, our work differs mainly from other existing works in that we apply techniques from other research domains (MAP and Game Theory) on a new variant of a well-studied optimization problem (VRP) with a real-world problem scale.

4.2 Coordinating Multi-Party Vehicle Routing with Location Congestion via Iterative Best Response

4.2.1 Problem Description

Multiple LSPs have to fulfill a list of pickup-delivery requests within a day. They have multiple vehicles which need to go to the pickup locations to load up the goods and deliver them to various commercial or retail locations such as warehouses and shopping malls. The vehicles need to return to their depot by a certain time and every request has a time window requirement. A wait time will be incurred if the vehicle arrives early and time violations if it serves the request late. In addition, every location has limited parking bays for loading and unloading, and a designated lunch hour break where no delivery is allowed. As such, further wait time and time window violations will be incurred if a vehicle arrives in a location where the parking bays are fully occupied or arrives during the designated lunch hour.

The objective of each LSP is to plan for a schedule that minimizes travel time, wait time and time window violations. Given that parking bays at every location are shared among the multiple LSPs, some sort of coordination is needed to deconflict their schedules to minimize congestion.

4.2.2 Model Formulation

We formulate ML-VRPLC as an n-player game $\Gamma_{ML-VRPLC}$ with LSPs represented as players $i \in N$ having a finite set of strategies S_i and sharing the same payoff function i.e. $u^1(s) = ... = u^n(s) = u(s)$. $s \in S_1 \times .... \times S_n$ is a finite set since S_i is finite. Table 4.1 provides the set of notations and the corresponding descriptions used in the model.

Strategy. In this chapter, we will use the terms *strategy*, *solution* and *schedule* interchangeably since a strategy of a player i.e. an LSP is represented in the form of a schedule. A schedule is a solution of a single-LSP VRPLC which consists of the routes (sequence of locations visited) of every vehicle and the corresponding time intervals (start and end service times) of every requests served by each vehicle. s_i is

Table 4.1: Set of notations used in $\Gamma_{ML-VRPLC}$.

Notation	Description
N	A set of LSPs, $N \in \{1, 2, ..., n\}$.
s_i	A schedule of LSP i, $i \in N, s_i \in S_i$.
s	A joint schedule of all LSP, $s = (s_1, s_2, ..., s_n), s \in S$.
s_{-i}	A joint schedule of all LSP except LSP i, $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_n)$.
(s_i, s_{-i})	A joint schedule where LSP i follows a schedule s_i while the rest follows a joint schedule, s_{-i}.
$u^i(s)$	Payoff of LSP i when all LSP follows a joint schedule, s.
$B_i(s_{-i})$	Best response of LSP i when all other LSPs follow a joint schedule, s_{-i}.

represented as the following tuple:

$$s_i = \langle s_i.routes, s_i.timeIntervals \rangle$$

Potential Function. We define a function, $P(s) = \sum_{i \in N} u^i(s)$ i.e. total weighted sum of travel times, wait times and time violations when all LSP follow a joint schedule s. In this paper, we define the payoff function, $u^i(s)$ as cost incurred (see Equation 4.7 for the full definition).

Lemma 1. *$P(s)$ is an* ordinal potential function *for $\Gamma_{ML-VRPLC}$ since for every $i \in N$ and for every $s_{-i} \in S_{-i}$*

$$u^i(s_i, s_{-i}) - u^i(s'_i, s_{-i}) > 0 \text{ iff}$$
$$P(s_i, s_{-i}) - P(s'_i, s_{-i}) > 0 \text{ for every } s_i, s'_i \in S_i. \tag{4.1}$$

Proof.

$$\begin{aligned} & P(s_i, s_{-i}) - P(s'_i, s_{-i}) > 0 \\ \Rightarrow u^i(s_i, s_{-i}) + \sum_{j \in -i} u^j(s_{-i}) - \Big(u^i(s'_i, s_{-i}) + \sum_{j \in -i} u^j(s_{-i}) \Big) > 0 & \\ \Rightarrow u^i(s_i, s_{-i}) - u^i(s'_i, s_{-i}) > 0 & \end{aligned} \tag{4.2}$$

□

Thus, $\Gamma_{ML-VRPLC}$ is a *finite ordinal potential game* and it possesses a pure-strategy equilibrium and has the Finite Improvement Property (FIP) [86]. Having the FIP means that every path generated by a best response procedure in $\Gamma_{ML-VRPLC}$ converges to an equilibrium. We are able to show conceptually and empirically that our approach converges into an equilibrium in the later sections.

Equilibrium and Local Optimality. $s' = (s'_i, s'_{-i})$ is an equilibrium if

$$u^i(s'_i, s'_{-i}) \leq u^i(s_i, s'_{-i}) \text{ for all } i \in N \text{ where } s_i \in B_i(s'_{-i}). \tag{4.3}$$

An equilibrium of $\Gamma_{ML-VRPLC}$ is a local optimum since no player can improve its payoff / reduce its cost by changing its individual schedule. Conversely, every optimal solution, s^* of $\Gamma_{ML-VRPLC}$ is an equilibrium since $u^i(s^*) \leq u^i(s_i, s^*_{-i})$ for all $i \in N$ where $s_i \in B_i(s^*_{-i})$.

Objective Function. The objective of this problem is to minimize the maximum payoff deviation of any one LSP from an ideal solution.

$$min_{s \in S} f(s) \tag{4.4}$$

$$f(s) = max_{i \in N} Deviation_{LB}(s, i) \tag{4.5}$$

$$Deviation_{LB}(s, i) = \frac{u^i(s) - u^i(s^{ideal})}{u^i(s^{ideal})} \times 100\% \tag{4.6}$$

where s^{ideal} is defined as the joint schedule where all other LSPs do not exist to compete for parking bays. s^{ideal} is a Lower Bound (LB) solution since it is a solution of a relaxed $\Gamma_{ML-VRPLC}$. We are essentially trying to search for solutions where each LSP's payoff is as close as possible to its corresponding LB solution.

We do not define the objective function as $min_{s \in S} \sum_{i \in N} u^i(s)$ because in this game, the players are not concerned about the system optimality (total payoffs of all players) but rather on how much benefit it can obtain by adopting a coordinated planning instead of planning independently.

4.2.3 Solution Approach

The key idea of our proposed approach is to improve a chosen joint schedule iteratively by computing the best responses of each player assuming the rest of the players adopt the chosen joint schedule until no improvement can be obtained to the resulting joint schedule or until a given time limit or maximum number of iterations has been reached. Our approach is decentralized in nature because each LSP is an independent agent which can compute its own route and schedule i.e. a central agent does not dictate how each player determine their decisions.

Given that we have established that our problem is a *potential game* and has an FIP, our approach will converge to an equilibrium which has been shown in

the previous section to be equivalent to a local optimal solution. Therefore, our approach seeks to explore multiple local optimal solutions until the terminating conditions are met and returns the best one found so far.

4.2.3.1 Iterative Best Response Algorithm

Algorithm 4 describes how the iterative best response algorithm works. At each iteration (lines 3-22), a joint schedule is chosen from a sampling pool of previously obtained improved joint schedules or from the current best joint schedule (line 7). We implement an epsilon greedy sampling policy to allow for exploration of multiple improvements paths (see Figure. 4.1 for an example of an improvement path) to search for best joint schedule. An improvement step consisting of $n-1$ best response computations is applied to the chosen joint schedule to obtain new improved joint schedules (line 10). If no further improvement can be made to the sampled joint schedule, we proceed to the next iteration (lines 11-13). We update the current best joint schedule if any of the new joint schedules has a lower $f(s)$ value than f_{min} (lines 15-16). Otherwise, we place the new improved joint schedules into the sampling pool for further improvement steps in the subsequent iterations (lines 17 and 19). We repeat the process until termination conditions are met. Then, we return the current best joint schedule as the final output.

Initial Solution, Lower Bound and Upper Bound Solutions. The initial joint schedule can be initialized to any random, feasible joint schedule. However, in this problem, we use the uncoordinated joint schedule as the initial solution to be improved by iterative best response algorithm. To compute the initial joint schedule, $s^{initial}$, we first compute the best schedules for each LSP independently assuming no other LSPs exist to compete for the limited resources. This is akin to solving a single-LSP VRPLC. The resulting joint schedule is in fact s^{ideal} and is the LB solution to $\Gamma_{ML-VRPLC}$. Next, a scheduler consisting of a Constraint Programming (CP) model that incorporates the resource capacity constraint at each location is solved for the combined routes of s^{ideal}. This forms an uncoordinated joint schedule, $s^{uncoord}$ which serves as an Upper Bound (UB) solution to $\Gamma_{ML-VRPLC}$ as any coordinated planning approaches must result in solutions that are better than an uncoordinated one. We use the LB and UB solutions in the experiments to

Algorithm 4: Iterative Best Response Algorithm to solve ML-VRPLC

Input : Initial joint schedule $s^{initial}$, maximum iteration K, time limit T
Output : Best found joint schedule s^{best}

$s^{best} := s^{initial}$, $f_{min} := f(s^{initial})$, $k = 0$
Create a sampling pool of joint schedules, $\mathbf{H} = \{s^{initial}\}$
while $k < K$ *and* $runTime < T$ *and* $\boldsymbol{H} \neq \{\emptyset\}$ **do**
  if $k = 0$ **then**
    $s^k := s^{initial}$
  else
    With probability ε, $s^k \sim U(\mathbf{H})$ otherwise $s^k := s^{best}$
  end
  Remove s^k from $\mathbf{H}$
  Find new joint schedules $\{s^{k,1}, s^{k,2}, ..., s^{k,n}\}$ where $s^{k,i} = (s_i^k, s_{-i}^k), u^i(s^{k,i}) < u^i(s^k)$ and $s_i^k \in B_i(s_{-i}^k)$
  if $u^i(s^k) \leq u^i(s^{k,i})$ *for all* $i \in N$ **then**
    $k+ = 1$
    continue
  end
  if $min_{i \in N} f(s^{k,i}) \leq f_{min}$ **then**
    $s^{best} := s^{k,i^*}$, $f_{min} := f(s^{k,i^*})$
    put $\{s^{k,i}\}_{i \in N \setminus \{i^*\}}$ in $\mathbf{H}$
  else
    put $\{s^{k,i}\}_{i \in N}$ in $\mathbf{H}$
  end
  $k+ = 1$
end
return s^{best}

evaluate the solution quality of our proposed approach.

Finite Improvement Paths and Convergence. Each improved joint schedule can be represented as a node in a directed tree. A series of nodes with parent-child relationship forms an improvement path as shown in Figure. 4.1 where $P(s^{k,i}) < P(s^{k-1,i'})$ for all $k \geq 1$ and $i, i' \in N$. Every improvement path is finite since S is a finite set. Every finite improvement path will converge to an equilibrium and every terminal point is a local optimum. However, since the best response is computed heuristically and there is no way to prove optimality, the resulting equilibrium is just an approximate. Nevertheless, we can show empirically in our experiments that our approach will converge to an approximated equilibrium solution after a certain

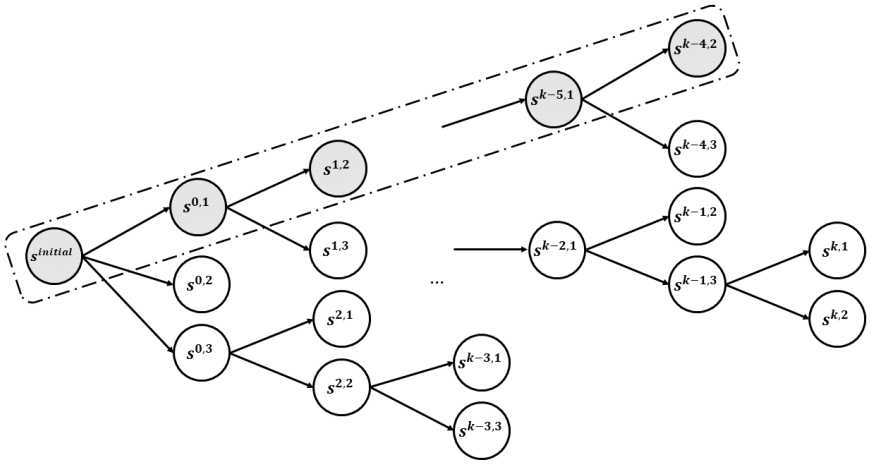


Figure 4.1: One example of an improvement path assuming $n = 3$.

number of iterations.

In short, our approach explore multiple improvement paths to search for joint schedule that return the best objective value, $f(s)$ with the lowest total payoffs, $P(s)$ as a secondary objective.

4.2.3.2 Best Response Computation

At every iteration, best response to a chosen joint schedule, s^k is computed for each LSP (line 10 of Algorithm 4). The best response computation of single LSP is equivalent to solving a single-LSP VRPLC where the resource constraint is determined by the resource utilization of each location by all other LSPs based on s^k_{-i}. Table 4.2 shows the notations used in this single-LSP VRPLC model.

We propose a heuristic consisting of Adaptive Large Neighbourhood Search (ALNS) as route optimizer and a scheduler based on a CP model to solve this single-LSP VRPLC. Heuristic is proposed as it is more scalable for a real-world problem setting. ALNS is used to search for better routes and the CP model based on the resulting routes is then solved to produce a schedule that meets the resource and time-related constraints. ALNS is chosen because it is probably the most effective metaheuristic for the VRPPDTW [74] and ALNS is widely used to solve large-scale problem [132]. Algorithm 5 details the proposed best response computation consisting of ALNS and CP model.

Table 4.2: Set of notations used in the single-LSP VRPLC model.

Notation	Description
V	A set of vehicles.
R	A set of all requests.
M	A set of all locations.
R_v	A set of requests served by vehicle v.
O_m	A set of requests at location $m \in M$.
$C_{m,t}$	Resource capacity at location m at time t.
$e_{r,v}$	Lower time window of request r served by vehicle v.
$l_{r,v}$	Upper time window of request r served by vehicle v.
$prev(r)$	Previous request served prior to request r, $prev(r), r \in R_v$.
$d_{x,y}$	Travel time from location of request x to location of request y.
$timeInterval_{r,v}$	Time interval when request r in vehicle v is being served, consisting of start and end time.
$\overline{T_0}, coolingRate$	Parameters for acceptance criteria in Simulated Annealing.

4.2.3.3 ALNS as Route Optimizer

The ALNS algorithm implemented in this paper is adapted from the vanilla version of ALNS proposed by Ropke and Pisinger [104] with differences in the choices of remove and insert operators and parameters used. However, the key difference in our ALNS implementation lies in line 7 of Algorithm 2. To compute the time intervals and the corresponding payoff of the updated solution, a CP model is solved. The detailed description of the CP model can be found in Section 4.2.3.4.

As the words *adaptive* and *large* in ALNS imply, ALNS explores large search space for new solutions by adaptively choosing remove and insert operators to remove a certain number of orders from existing solution and reinserting them back to other positions to form a new solution (lines 4-6). In our implementation, we use the following remove and insert operators.

Remove Operators We define the following 5 remove operators and each operator will select $10 - 15\%$ of orders uniformly to be removed from the current solution.

1. **Random removal** – This operator randomly selects orders from the current solution. Random removal allows exploration of larger search space even though the probability of finding a better solution is low.

2. **Worst removal** – This operator selects orders that result in the maximum increase in payoff/cost if removed from the current solution.

Algorithm 5: Best Response Computation

Input : Chosen solution s^k, initial temperature $\overline{T_0}$, *coolingRate*
Output : $B_i(s^k_{-i})$

$s_i^{best} := s_i^k, s_i := s_i^{input}, \overline{T} = \overline{T_0}$
while *termination criteria are not met* **do**
  $s'_i := s_i$
  Select removal and insert operators via roulette wheel mechanism
  Apply the selected removal operator to remove the requests from $s'_i.routes$
  Apply the selected insert operator to insert the orders into $s'_i.routes$
  Calculate the cost/payoff, $u^i(s'_i, s^k_{-i})$ and update $s'_i.timeIntervals$
  if $u^i(s'_i, s^k_{-i}) < u^i(s_i^{best}, s^k_{-i})$ **then**
    $s_i^{best} := s'_i, s_i := s'_i$
  else
    if $u^i(s'_i, s^k_{-i}) < u^i(s_i, s^k_{-i})$ **then**
      $s_i := s'_i$
    else
      $s_i := s'_i$ with probability, $\min\{1, e^{(u^i(s_i, s^k_{-i}) - u(s'_i, s^k_{-i}))/\overline{T}}\}$
    end
  end
  Update the weights and scores of the operators accordingly
  $\overline{T} := \overline{T} * coolingRate$
end
$B_i(s^k_{-i}) := s_i^{best}$
return $B_i(s^k_{-i})$

3. **Spatio-temporal distance removal** – This operator selects orders which has the highest sum of spatio-temporal distances with their adjacent orders. This operator tries to relocate orders that are more likely to incur higher travel cost and time window violations. The definition of spatio-temporal distance implemented was first introduced in [99].

4. **Time-violation removal** – This operator selects orders that result in highest time window violation. Similar to spatio-temporal distance removal, this operator tries to relocate orders that are "out of position" with respect to their time window requirements.

5. **Shaw removal** – This operator was first introduced in [109]. The basic idea of Shaw removal is to select orders that are similar to each other. The intuition is that removing and reinserting orders that are similar to each other is easier

and is more likely to create better solution. Removing and inserting orders that are vastly different from one another can be challenging and may result in unsuccessful reinsertions or poor insertion positions.

The above remove operators are selected to provide both exploitation and exploration in the search process. Randomization in the operator provides the exploration capability while removal operators like worst, spatio-temporal and time-violation are more exploitative in nature as they aim to improve the solution in a greedy manner. Shaw removal is essentially the opposite of worst removal and thus having both operators provides a diversification of search. Shaw removal select orders that are easier to remove and insert while worst removal select orders that are relatively harder to reinsert.

Insert Operators Prior to any insertion operation, orders are selected for removal using the selected remove operations as described in previous section. The selected orders can be removed in two ways namely remove all or remove one by one. Remove all indicates that the selected orders for removal are removed together prior to insertion while remove one by one indicates that order is removed one at the time prior to insertion. Meanwhile, insertion is done sequentially which means that for remove all, all orders are removed and reinserted one by one while for remove one by one, each order is removed and reinserted one at a time.

These are the 6 insert operators implemented:

1. **Remove all with greedy insertion** – All orders are removed prior to any insertion. Greedy insertion heuristics insert the order that returns the minimum increase in payoff/cost.

2. **Remove one by one with greedy insertion** – Similar to the first operator except that orders are removed and inserted one at a time.

3. **Remove all with greedy insertion with noise** – Similar to the first operator except that calculation of increase in payoff/cost includes a noise function.

4. **Remove one by one with greedy insertion with noise** – Similar to the second operator except that calculation of increase in payoff/cost includes a noise function.

5. **Remove all with regret-k insertion** – All orders are removed prior to any insertion. This operator inserts orders based on their regret values. Regret value is defined as the difference in the cost of inserting into its best route and its k^{th} best route.

6. **Remove one by one with regret-k insertion** – Similar to the fifth operator except that orders are removed and inserted one at a time.

Similar to the remove operators, the above insert operators are selected to provide both exploitation and exploration in the search process. Randomization in the operator via noise function provides the exploration capability while greedy insertion is more exploitative in nature. In addition, regret insertion provides a certain degree of lookahead capability since greedy insertion is more likely to cause the solution to be stuck at local optima.

On top of the exploration capability provided by the remove and insert operators, a similar mechanism used in Simulated Annealing is applied in choosing poorer solutions to further enlarge the search space and also to escape local optima (line 14). For more detailed discussions of ALNS algorithm such as on the roulette wheel mechanism (line 4) and the adaptive weight adjustment for the operators (line 17), we refer our readers to [104].

4.2.3.4 CP Model as Scheduler

The following CP model is solved to obtain the updated time intervals of the newly-found routes and the corresponding payoff of the updated schedules at every ALNS iteration in line 7 of Algorithm 5. The payoff is computed as follow:

$$u^i(s_i) = w_1 \times totalTravelTime(s_i.routes) +$$
$$minimize \sum_{v \in V} \left\{ w_2 \times \sum_{r \in R_v} waitTime_{r,v} + w_3 \times \sum_{r \in R_v} timeViolation_{r,v} \right\} \quad (4.7)$$

where

$$w_1, w_2, w_3 \text{ are predetermined set of weights,}$$
$$waitTime_{r,v} = min\{0, (start(timeInterval_{r,v}) - end(timeInterval_{prev(r),v}) - d_{prev(r),r})\},$$
$$timeViolation_{r,v} = min\{0, (end(timeInterval_{r,v}) - l_{r,v})\},$$
$$s_i.timeIntervals = \{timeInterval_{r,v}\}_{r \in R_v, v \in V}$$

The second term of Equation 4.7 is the objective function of the CP model with $\{timeInterval_{r,v}\}_{r \in R_v, v \in V}$ as the primary decision variables of the model. The key constraints of the CP model are as follow:

$$CUMULATIVE(\{timeInterval_{r,v} : v \in V, r \in R_v \cap O_m\}, 1, C_{m,t}), \forall m \in M \quad (4.8)$$
$$noOverlap(\{timeInterval_{r,v} : r \in R_v\}), \forall v \in V \quad (4.9)$$
$$start(timeInterval_{r,v}) \geq end(timeInterval_{prev(r),v}) + d_{prev(r),r}, \forall r \in R_v, v \in V \quad (4.10)$$
$$start(timeInterval_{r,v}) \geq e_{r,v}, \forall r \in R_v, v \in V \quad (4.11)$$

Constraint 4.8 is used to model the resource capacity constraint at each location at a given time t where $start(timeInterval_{r,v}) \leq t \leq end(timeInterval_{r,v})$ and $C_{m,t}$ is determined by the resource utilization of all other LSPs based on s^k_{-i}. Constraint 4.9 ensures that the time intervals of requests within a route do not overlap. Constraints 4.10 and 4.11 ensure that the start time of a request must at least be later than the end time of the previous request plus the corresponding travel time and it should not start before its lower time window. Other constraints relating to operational requirements such as no delivery within lunch hours, operating hours of the locations and vehicles are omitted to simplify the discussion as it is fairly straightforward to incorporate these constraints.

4.2.3.5 Solution Illustration

In this section, we provide a simple illustration on how our approach works using a simple toy example involving 2 LSPs and here, we assume each location has a capacity of one. Our approach begins by computing the initial solution, lower

bound and upper bound solutions as explained in Section 4.2.3.1. Each LSP plans independently assuming no other LSPs exist to compete for resource and the resulting joint schedule is as follows $s_1 = \langle [A, B, ...], [(5, 7), (9, 11), ...] \rangle$ and $s_2 = \langle [B, D, ...], [(4, 12), (15, 17), ...] \rangle$. This resulting joint schedule is in fact s^{ideal} which is a LB solution. However, it is observed that LSP 1 will need to wait at Location B because LSP 2 is still occupying Location B at time interval $(9, 11)$. Thus, through a CP-based scheduler, this initial joint schedule is revised to a feasible solution such as $s_1 = \langle [A, B, ...], [(5, 7), \mathbf{(13,15)}, ...] \rangle$ while s_2 remains. Due to the waiting time incurred in Location B, the cost of this initial solution is higher. This is a $s^{uncoord}$ since there is no coordination involved and is also an UB solution.

Through a iterative best response procedure, our approach tries to improve this initial solution. For instance, at a given iteration, we first assume s_2 to remain and compute the best response of LSP 1 by using the proposed heuristic (see Section 4.2.3.2). At the same time, we compute the best response of LSP 2 assuming s_1 remains. Each of the resulting new joint schedule is kept if its objective value is better than the f_{min} (see line 15-17 of Algorithm 4) or is better than the solution at the start of this iteration (lines 18-19). Assuming that only 1 resulting joint schedule, $s_1 = \langle [A, F, ..., B, ...], [(5, 7), (10, 13), ..., \mathbf{(20,22)}, ...] \rangle$ and $s_2 = \langle [B, D, ...], [(4, 12), (15, 17), ...] \rangle$ returns an improved objective value, we set this schedule as the current best solution, s^{best} and place it in the sampling pool H. This resulting joint schedule is an improved solution because a congestion is avoided at Location B as LSP 1 visits Location B at a later time interval, $(20, 22)$.

At the next iteration, since the sampling pool consists only 1 joint schedule and it is also the current best solution, another round of best response computations is done on this schedule. Assuming that the resulting new joint schedules do not return improved objective values, this current best solution is then returned as the local optimal solution which is also an approximate equilibrium solution since no one player can improve its own payoff. This example illustrates a terminating condition where the sampling pool is empty and no improvement can be made to the current best solution. Another instance of terminating condition will be when the computation time reaches a pre-determined time limit.

In this simple illustration, we simulate 1 iteration of iterative best response procedure. In practice, there would be multiple iterations and each iteration will

explore multiple improvement paths since the sampling pool would contain multiple improved joint schedules.

4.2.3.6 Scalability and Flexibility

Our approach is scalable because the best response computations for every LSP can be done in parallel since they are independent of each other (line 10 of Algorithm 4). In other words, explorations of multiple improvement paths as shown in Fig. 4.1 can be done concurrently. Our approach is also flexible as it also allows any other forms of solution approach to single-LSP VRPLC to be used to compute the best response.

4.2.4 Experiments

The objective of the experiment is twofold. First, we would like to empirically verify whether our approach converges to an equilibrium for our problem setting and second, to evaluate the solution quality produced by our proposed approach. For a more comprehensive evaluation of our proposed approach, we look into the following aspects in our experiments:

1. ***Exploration vs. Exploitation.*** We investigate the impact of implementing ε-greedy sampling policy to the solution quality.

2. ***Our Approach vs. Centralized.*** We compare our proposed decentralized solution approach with a centralized approach with respect to s^{ideal} (LB) and $s^{uncoord}$ (UB). Intuitively, our approach should return solutions with lower payoff/cost than UB solution and within a reasonable deviation from LB solution.

3. ***Sensitivity Analysis.*** We also investigate the impact of plan deviations by any of the LSPs to solution quality.

4.2.4.1 Experimental Setup

We synthetically generate 30 test instances to simulate a month's worth of pickup-delivery requests for 20 LSPs. These instances are generated based on existing datasets of our trials with several local LSPs. Each test instances consists of 100

requests per LSP and each LSP has 10 vehicles. To simulate congestion at the delivery locations, we narrow down the delivery locations to 15 unique shopping malls with maximum capacity of 4 parking bays per location. Our approach is implemented with K set at 300, $T = 60$ mins and $\varepsilon = 0.3$. The implementation codes are written in Java while CP Optimizer ver. 12.8 is used to solve the CP model. The experiments are run on a server with the following configurations: CentOS 8 with 24 CPU Cores and 32GB RAM.

Benchmark Algorithm. We chose a centralized, non-collaborative planning approach as a benchmark algorithm. It is centralized since all LSPs are treated as one single LSP and the central agent makes the routing and scheduling decision on behalf of the LSPs. It is non-collaborative as no exchange of requests or sharing of vehicles are allowed i.e. each vehicle can only serve requests from the LSP they belong to. We use a heuristic approach combining ALNS and CP model similar to the one used to compute best response to solve this single-LSP VRPLC. The initial solution is constructed via randomized Clarke-Wright Savings Heuristics adapted from [88]. The algorithm is run for 1 hour and 2 hours for each test instance.

Performance Measures. On top of $f(s)$, we introduce other performance measures to evaluate the two approaches. The other performance measures introduced are as follow:

1. **Maximum payoff deviation from an uncoordinated solution.** $f'(s)$ measures the payoff deviation of the worst performing LSP from the payoff if it follows a schedule based on an uncoordinated planning. A negative deviation value indicates reduction in cost and the lower the value, the higher the improvement gained from the UB solution.

$$f'(s) = max_{i \in N} Deviation_{UB}(s, i) \tag{4.12}$$

$$Deviation_{UB}(s, i) = \frac{u^i(s) - u^i(s^{uncoor})}{u^i(s^{uncoor})} \times 100\% \tag{4.13}$$

2. **Average payoff deviation from an ideal solution.** The lower the value, the closer the solution is to the LB solution.

$$g(s) = \frac{1}{n} \times \sum_{i \in N} Deviation_{LB}(s, i) \tag{4.14}$$

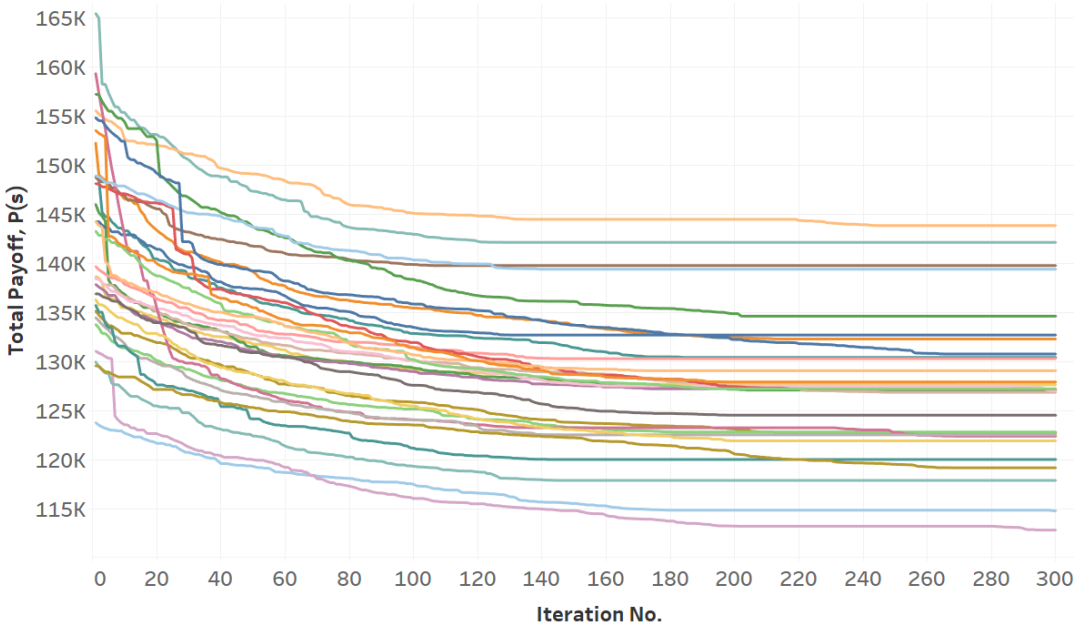


Figure 4.2: The total payoffs converge for all 30 test instances. Each coloured line represents the result of one test instance.

3. **Average payoff deviation from an uncoordinated solution.** Similar to Equation 4.12, a negative deviation value indicates reduction in cost.

$$g'(s) = \frac{1}{n} \times \sum_{i \in N} Deviation_{UB}(s, i) \tag{4.15}$$

We include results in terms of average and percentiles for a more extensive evaluation since the approaches being evaluated are heuristics and contain a certain degree of stochasticity. However, not all of the performance measures are being used in every experiment. At different parts of the experiments, we select only those which are relevant.

4.2.4.2 Experimental Results

Convergence. Figure. 4.2 shows that the total payoffs of all players converged after 200 iterations on average for all test instances. This supports our earlier deduction that $\Gamma_{ML-VRPLC}$ possesses an FIP and our proposed algorithm explores multiple improvement path that will converge to an approximated equilibrium. Meanwhile, the average run-time for 200 iterations is around 1 hour.

Table 4.3: The impact of ε to the solution quality in terms of the objective function, $f(s)$ and total payoff, $P(s)$ across 30 test instances.

Performance Measure		$\varepsilon = \mathbf{0}$	$\varepsilon = \mathbf{0.3}$	$\varepsilon = \mathbf{0.5}$
Max payoff	Q1	19.3%	17.1%	17.0%
deviation from LB	Q2	21.3%	21.1%	20.5%
$f(s)$	Q3	25.0%	24.2%	26.2%
	Avg	**31.8**%	**21.1**%	**26.5**%
Total payoff (in 1000s)	Q1	124.0	122.6	122.1
$P(s)$	Q2	126.5	127.2	128.4
	Q3	130.6	130.7	132.6
	Avg	**128.2**	**127.5**	**128.1**

Exploration vs. Exploitation. We investigate the impact of the value of ε to the solution quality in terms of the objective value, $f(s)$ and the total payoff, $P(s)$ as secondary objective function. As mentioned earlier, the value of ε determines the probability of exploring "poorer" improvement path at every best response iteration. $\varepsilon = 0$ implies full exploitation and no exploration, meaning that each best response procedure is done to improve only the current best solution. In this experiment, we run our solution approach with 3 different ε values $(0, 0.3$ and $0.5)$ against the 30 test instances.

As shown in Table 4.3, although the impact on the total payoff does not seem to be significant, our approach with $\varepsilon = 0.3$ returns solutions where the payoff of worst performing LSP is within 21.1% on average compared to 26.5% and 31.8% when $\varepsilon = 0.5$ and 0 respectively. Based on our experiment, our choice of $\varepsilon = 0.3$ provides a balance between exploration (high ε value) and exploitation (low ε value) and produces better quality solutions consistently across the 30 test instances.

Our Approach vs. Centralized. As shown in Figure 4.3, we intentionally present the results as a line chart and sort the test instances based on increasing total payoff of the ideal solution to better illustrate that our approach returns solutions whose total payoff are lower than the centralized approach and are well within the UB and LB solutions in all 30 test instances.

Table 4.4 shows that our approach outperforms the centralized approach on every performance measure even when the run-time for the centralized approach is

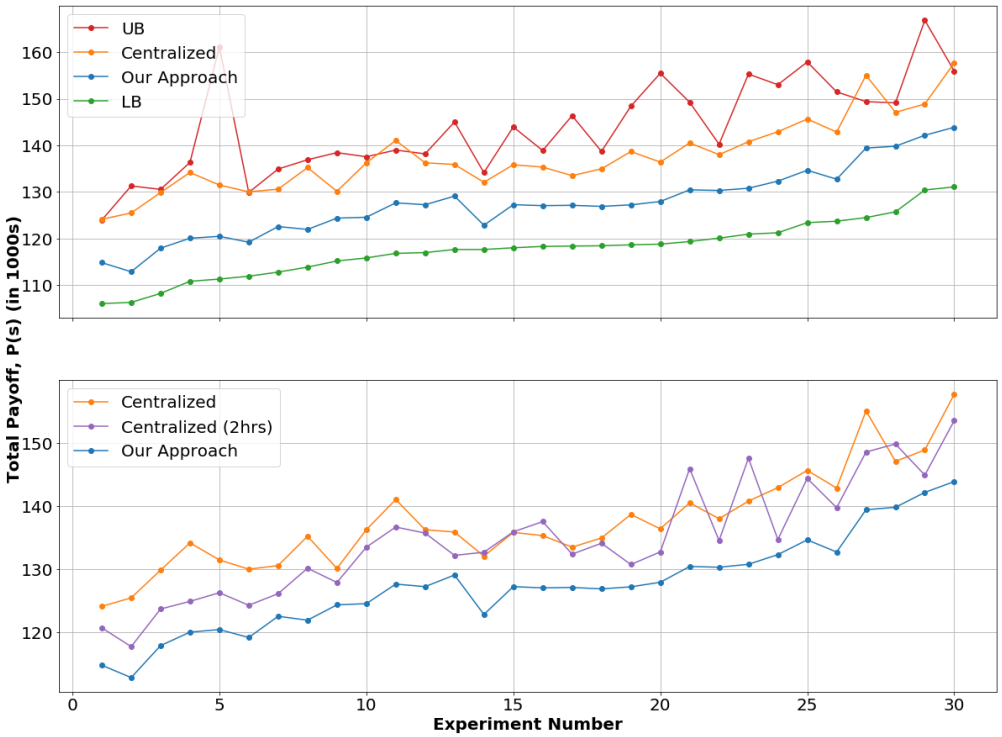


Figure 4.3: Our proposed approach outperforms the centralized approach (even when the run-time is doubled) and its solutions are well within the LB and UB solutions in terms of total payoff.

increased to 2 hours. In terms of the performance of the worst LSP, our approach is able to ensure that on average, the payoff of the worst performing LSP is still within about 21.1% from the LB solution and at least gain about 2.7% improvement over the uncoordinated solution. Meanwhile, even with doubling of the run-time, the centralized approach can only manage to ensure that the payoff of the worst performing LSP is within 32.0% from the LB solution while incurring a 13.1% additional cost as compared to an uncoordinated planning.

On average, across all LSPs, our approach return solutions that are well within 8.6% deviation from the LB solution and improve the payoff of the LSPs by an average of 10.4% from an uncoordinated planning approach. This is contrasted with the centralized approach which can only manage to return solutions that are within 14.9% of LB solution on average and an improvement of about 5.1% from the UB solution even when the run-time is doubled.

We observe that the worst performing LSP in centralized approach consistently returns g values that are positive (see Table 4.4) which indicates that the solution

Table 4.4: Our approach outperforms the centralized approach on every performance measures across 30 test instances.

Performance Measure		**Our Approach**	**Centralized (1hr)**	**Centralized (2hrs)**
Max payoff	Q1	17.1%	28.0%	23.9%
deviation from LB	Q2	21.1%	30.7%	28.8%
$f(s)$	Q3	24.2%	36.5%	33.5%
	Avg	**21.1**%	**35.0**%	**32.0**%
Max payoff	Q1	−3.1%	10.5%	6.8%
deviation from UB	Q2	−2.1%	13.7%	9.2%
$g(s)$	Q3	−1.1%	16.4%	15.4%
	Avg	**−2.7**%	**13.7**%	**13.1**%
Avg payoff	Q1	7.5%	15.5%	12.5%
deviation from LB	Q2	8.6%	17.2%	14.1%
$f'(s)$	Q3	9.4%	18.7%	17.1%
	Avg	**8.6**%	**17.4**%	**14.9**%
Avg payoff	Q1	−11.8%	−5.2%	−7.4%
deviation from UB	Q2	−9.9%	−2.4%	−4.6%
$g'(s)$	Q3	−8.2%	−0.6%	−1.6%
	Avg	**−10.4**%	**−3.0**%	**−5.1**%

for the worst performing LSP is even worse than that of an uncoordinated planning approach. This is because the centralized approach only concerns about the system optimality and not on the performance of each individual LSP. This reiterates our point that a centralized approach may result in some LSPs performing worse than if they are to plan independently.

Sensitivity Analysis. Our approach assumes that every LSP follow the generated coordinated schedules. However, in real-world settings, there are possibilities that some LSPs deviate from the generated plans. In this experiment, we investigate the impact of such plan deviations by any of the LSPs on the solution quality. To simplify the discussion, we assume scenarios where 10%, 30% and 50% of LSPs deviate from the generated schedules and follow their own planned schedules. We assume that LSPs are rational agents which mean that their own schedules are computed with the objective of minimizing their own total payoff/cost.

We generate and run another 30 test instances for this experiment. To evaluate the impact on the solution quality of the resulting schedules, we use the maximum and average payoff deviations from the generated plan as the performance measures. As shown in Table 4.5, a slight plan deviation caused by only 10% of the LSPs result in the worst performing LSP suffering a loss of almost 30%. Due to the stochastic

Table 4.5: The impact of plan deviations by 10%, 30% and 50% of the LSPs on the solution quality across 30 test instances.

Performance Measure		**10% of LSPs deviate**	**30% of LSPs deviate**	**50% of LSPs deviate**
Max payoff	Q1	12.5%	13.7%	16.3%
deviation from	Q2	18.7%	19.7%	21.4%
generated plan	Q3	27.7%	25.1%	47.2%
	Avg	**28.5**%	**35.7**%	**39.8**%
Avg payoff	Q1	4.3%	5.3%	6.1%
deviation from	Q2	5.4%	5.8%	8.0%
generated plan	Q3	6.3%	7.3%	10.3%
	Avg	**5.7**%	**7.2**%	**8.4**%

nature of our approach, the eventual worst performing LSP may not be the same every time the algorithm is run. This may create sufficient deterrence for LSPs from deviating from the generated schedules. On the other hand, even with half of the LSPs not following the generated plans, the average payoff deviation is kept within 10%. This shows that our proposed approach is able to produce solutions that are robust against plan deviations albeit with respect to the average performances of all the LSPs.

4.2.4.3 Experiment Discussion

The experiments show that our proposed decentralized approach outperforms a centralized approach given the available run-time limit of 1 hour in all 30 test instances and in all 4 performance measures. Furthermore, we also find that the centralized approach is computationally more expensive and therefore not as scalable as our decentralized approach as it needs longer run-time ($>$ 2 hours) to return solutions that are at least comparable to our approach.

To further verify the performance of the centralized approach and its lack of scalability, we run another set of experiments involving 5 LSPs with 100 pickup-delivery per LSP and each LSP having 10 vehicles. To simulate congestion at the delivery locations, we set the maximum capacity at 2 parking bays per location. Fig. 4.4 shows that both our proposed approach and the centralized approach produce comparable solutions in terms of total payoff across 30 test instances given the same time budget. To further substantiate this claim, we conduct the following paired

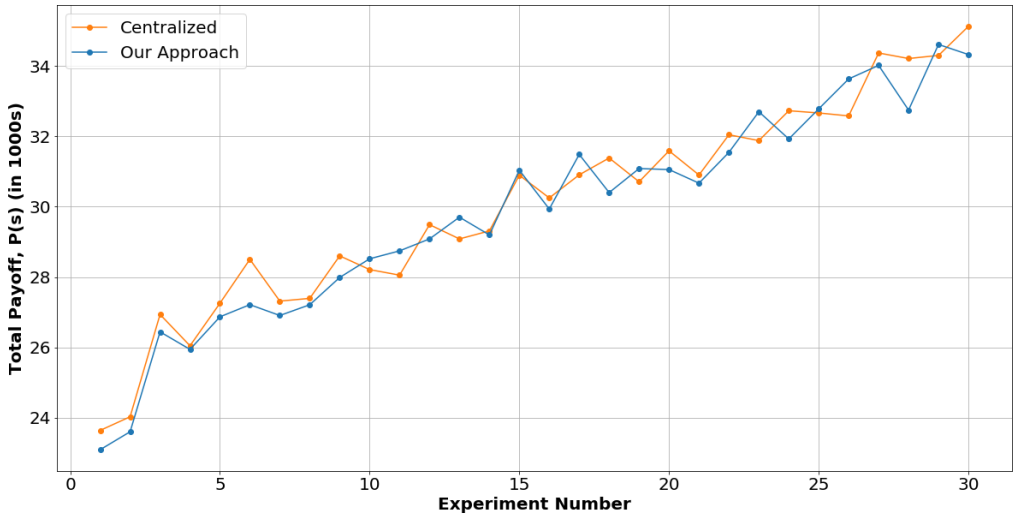


Figure 4.4: Our proposed approach and the centralized approach produce comparable solutions in terms of total payoff across 30 test instances (5 LSPs). Similar to Figure 4.3, we intentionally present the results as a line chart and sort the test instances based on increasing total payoff of the ideal solution for ease of visualization.

t-test:

$$H_0 : \mu_d = 0$$
$$H_1 : \mu_d \neq 0$$

where μ_d refers to the mean difference between the total payoffs of our proposed approach and the centralized approach. In this test, we assume 95% confidence level. The test returns a p-value of 0.79 which is greater than $\alpha = 0.05$. Thus, there is no significant evidence to reject the null hypothesis and as such we can conclude statistically that the total payoff of our proposed approach and centralized approach are comparable. We have shown earlier that the performance gap between the two approaches widens when the problem scale gets larger (see Section 4.2.4.2). Therefore, the centralized approach indeed performs well only with smaller scale problems.

Although further experimentation may be needed to evaluate the robustness of our approach against various problem scenarios such as varying the number of LSPs, vehicles, requests and pickup-delivery locations, we have deliberately chosen to conduct sufficiently varied types of experiments to show that even though there will be LSPs who gain more and others who will gain less, our approach is able to

ensure that there are enough incentives (and deterrence too) for LSPs to adopt and follow this coordinated planning as compared to them performing their own selfish, independent planning.

4.3 Conclusion

The key idea proposed in this chapter is a scalable, decentralized, coordinated planning approach that can be tailored to large-scale optimization problems involving multiple *loosely coupled* entities competing for shared resources. Our proposed iterative best response algorithm decomposes a multi-agent problem into multiple single-agent problems allowing existing single-agent planning algorithms to be applied to a smaller problem.

Even though we assume that the best response algorithms and the payoff functions of each LSP (or agent) are identical, our approach can be extended to problems where each LSP adopts different best response algorithm and payoff function. The best response computation algorithm is akin to a black-box which can be replaced with any solution algorithm to solve single-LSP VRPLC (or single-agent version of the problem). Moreover, even with non-identical payoff functions, the inequality condition in Equation 4.1 will still be valid and therefore our approach will still converge to an approximated equilibrium.

One key limitation of our approach is that we assume the environment is static which may not be the case in real-world setting. We assume that every LSP in the system is cooperative in the sense that it participates and adheres to the coordinated planning without any possibility of plan deviation such as dropping out of the system or making changes to their pickup-delivery requests. It is interesting to investigate and enhance our approach to take into consideration uncertainty in the environment and evaluate its robustness in a dynamic environment, as well as to extend it to domains beyond logistics.

Chapter 5

RL Approach to Coordinate Multi-Agent Dynamic Routing and Scheduling

In the previous two chapters, we propose new solution approaches to address each of the key features of real-world routing and scheduling problems (i.e. *dynamicity* and *multi-agent*). In this chapter we address and propose a solution approach to problems that incorporate both *dynamicity* and *multi-agent*. More precisely, we propose an MARL approach that combines Multi-Agent Value Function Approximation (MAVFA) with planning heuristic to solve multi-agent dynamic routing and scheduling problem directly without the need to decompose the action or the problem into multiple stages. In our approach, the learned value function is utilized by the heuristic to search for better decision. This is in fact a multi-agent extension to our proposed solution presented in Chapter 3. We focus our discussion on cooperative multi-agent setting where there is a need to coordinate decisions amongst agents for better global and individual returns. Therefore, to improve coordination and scalability, we incorporate iterative best response procedure that is proposed in Chapter 4 to act as a decentralized optimization heuristic and a coordination mechanism.

5.1 Motivation

As discussed in Section 2.6.3, current cooperative MARL approaches fall short in addressing multi-agent dynamic routing and scheduling problems directly. Current approaches either decompose the complex action into two stages or simplify the action into either discrete or continuous. In addition, current approaches assume

simultaneous actions by all agents and coordination amongst agents are done through communication. These assumptions are not ideal as there is no guarantee for proper event-handling process, for instance, two agents may respond to the same event or no agent is assigned to respond to the event. Thus, arising from the above-mentioned gaps in current works, we propose a cooperative MARL approach to address complex action directly with an explicit coordination mechanism in place of communication network (which is implicit in nature).

5.2 MAVFA with Planning Heuristic

In a multi-agent dynamic routing and scheduling problem, the objective at every decision epoch k is to select joint action x_k^* which maximizes the immediate reward and the expected future reward from yet-to-realized dynamic events which is represented by the approximated value function, $\hat{V}(S_k^x)$.

$$x_k^* = \underset{x_k \in X(S_k)}{\operatorname{argmax}} \{R(S_k, x) + \gamma \hat{V}(S_k^x)\} \tag{5.1}$$

To solve the optimization problem in Equation 5.1, we propose a solution approach that combines MAVFA based on value function factorization with planning heuristic where the former learns to approximate the joint value function and the latter is used to compute the argmax. In addition, we incorporate iterative best response procedure in our approach for a more scalable, coordinated decision-making.

5.2.1 MAVFA based on Value Function Factorization

We assume that the joint value function approximate, $\hat{V}(S_k^x)$ can be factorized into the value function approximates of each agent as shown in Equation 5.2. Similar to other value function factorization approaches like VDN and QMIX, we assume an Individual Global Max (IGM) principle. This assumption is reasonable because most cases of multi-agent dynamic routing and scheduling problems are not a zero-sum game. For instance, in same-day delivery problem, an agent which is assigned to a new job may gain profit or loss but this does not directly result in other agents suffering loss or gaining profit (here, we do not consider loss of opportunity).

$$\begin{aligned}\hat{V}(S_k^x) &= F\Big(\hat{V}(S_{k,1}^{x_{k,1}}), \hat{V}(S_{k,2}^{x_{k,2}}), ..., \hat{V}(S_{k,|I|}^{x_{k,|1|}})\Big) \\ &= F\Big((\hat{V}(S_{k,i}^{x_{k,i}}))_{i\in I}\Big)\end{aligned} \tag{5.2}$$

Combining Equation 5.1 and Equation 5.2, the objective function can be rewritten as follows.

$$x_k^* = \underset{x_k \in X(S_k)}{\operatorname{argmax}}\Big\{R(S_k, x_k) + \gamma F\Big((\hat{V}(S_{k,i}^{x_{k,i}}))_{i\in I}\Big)\Big\} \tag{5.3}$$

This objective function can be further rewritten as follows:

$$\begin{aligned}x_k^* = \underset{x_k \in X(S_k)}{\operatorname{argmax}} \underset{i\in I}{\operatorname{argmax}}\Big\{R\Big((S_{k-1,i}^{x_{k-1},i}, \omega_k, x_{k,i}), \\ (S_{k,-i}, x_{k,-i})\Big) \\ + \gamma F\Big((\hat{V}(S_{k,i}^{x_{k,i}}))_{i\in I})\Big)\Big\}\end{aligned} \tag{5.4}$$

This is because each pre-decision joint state can be represented by $|I|$ different realizations corresponding to the number of agents that can be assigned to the handle the new dynamic event (see Equation 5.5).

$$\begin{aligned}S_k &= \Big((S_{k-1,i}^{x_{k-1},i}, \omega_k), (S_{k-1,-i}^{x_{k-1},-i}, \varnothing)\Big) \\ &= \Big((S_{k-1,i}^{x_{k-1},i}, \omega_k), (S_{k,-i})\Big)\end{aligned} \tag{5.5}$$

We propose to represent $\hat{V}(S_k^x)$ as neural networks with parameters θ and its architecture can be found in Figure 5.1. Our approach learns the policy to execute complex action directly because the learned value function represents the value of a post-decision state, a state after executing both *event-handling* and *re-planning* actions.

5.2.1.1 Local Value Network

The architecture of the local value network, $\hat{V}(S_{k,i}^{x_{k,i}}, \theta_i)$ can be found in Figure 5.2. For scalability purpose, agent shares the same local network parameter θ_i. Here, we assume agents are homogeneous. We propose an encoder network in the form of multilayer perceptrons to extract the key features of the plan in its raw form and encoding it into a lower-dimensional vector representation. Depending on the

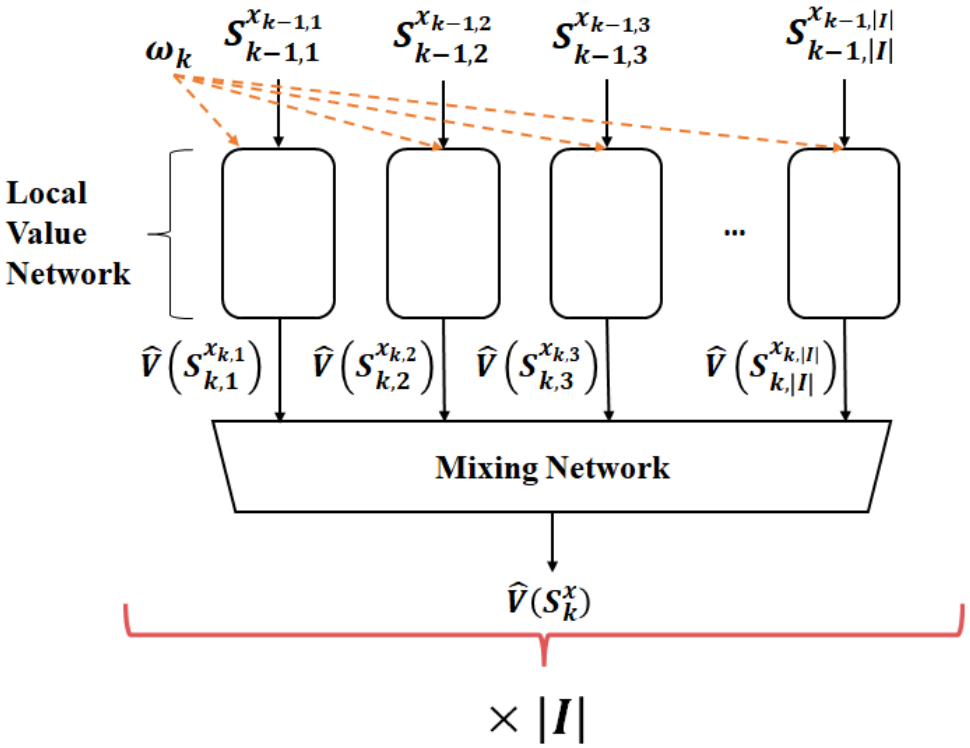


Figure 5.1: Given $|I|$ realizations of pre-decision state S_k, there are correspondingly $|I|$ possible variations of $\hat{V}(S^x_k)$.

problem, handcrafted features can be concatenated to enhance learning process. Many real-world multi-agent dynamic routing and scheduling problems often include complex constraints that have been thoroughly studied and thus expert knowledge should not be totally discarded.

The exact architecture of this network may differ depending on the problem and the nature of the state information. Recurrent Neural Network (RNN) or Convolutional Neural Network (CNN) can be used if the state information are sequential in nature or has spatial relationship respectively.

5.2.1.2 Mixing Network

We represent F in Equation 5.2 as neural networks and we loosely refer this network as a "Mixing Network". We use the term "mixing" in its general definition and it does not refer to the specific mixing network structure as proposed in QMIX. Similar to the local value network, the architecture of this network is also context-specific. The key feature of this network is that extra information that is common to all

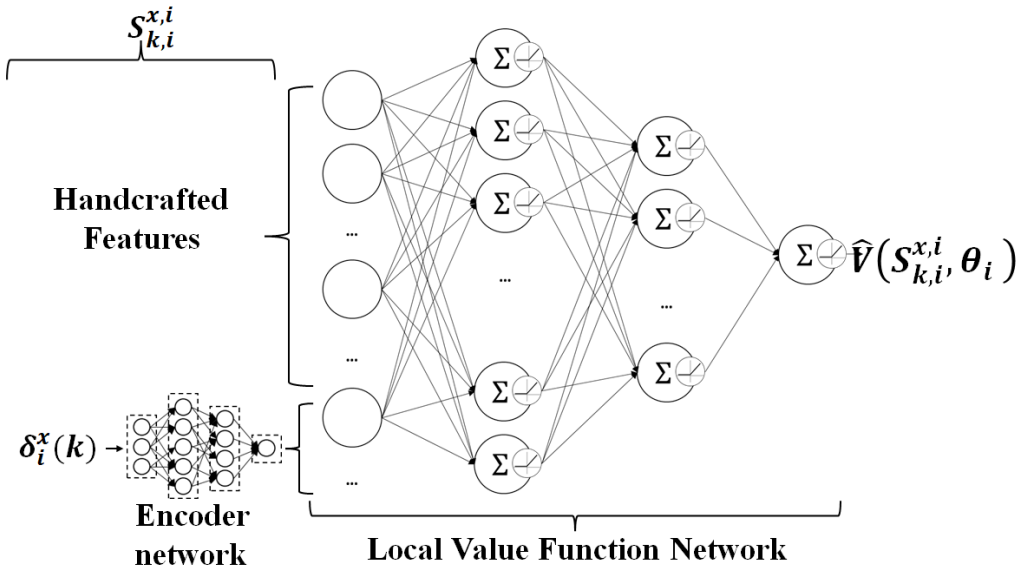


Figure 5.2: Local Value Network

the agents can be passed into the network to enhance coordination. Rashid *et al.* [100] in their ablation study have shown that providing extra state information does improve performance.

5.2.1.3 Learning Algorithm

To learn the parameter θ, we propose to use an on-policy TD learning with experience replay (see Algorithm 6). This learning algorithm is similar for Algorithm 1 with the main differences lie in how decision is being computed in line 7 and how the value function is being represented.

5.2.1.4 VFA-Guided Heuristic

To compute $\text{argmax}_{x_k \in X(S_k)}$ in Equation 5.1, we propose to use heuristic. The learnt value function $\hat{V}(S_k^x)$ is used to guide the heuristic to search for better decision that is both anticipatory and coordinated. Heuristic is chosen because of the need to make quick decision. Computing a decision for even a single-agent instance of dynamic routing and scheduling problem is equivalent to solving a mid-sized NP-hard problem (such as VRP or job scheduling problem) with complex constraints which cannot be solved to optimality given the short decision time.

Algorithm 6: MAVFA via TD Learning with Experience Replay

Input : No. Simulation Runs N, Replay Memory D, Value Function $\hat{V}(.,\theta)$, Target Value Function $\hat{V}_T(.,\theta^-)$

Output : θ

$i = 1$
while $i \leq N$ **do**
  Initialise S_0 with the initial plan
  $k = 1$
  while $S_k \neq S_K$ **do**
    $S_k \leftarrow (S^x_{k-1}, \omega_k)$
    With probability ε select a random decision otherwise compute x^*_k based on Equation 5.4
    $S^x_k \leftarrow (S_k, x^*_k)$
    Store transition $(S_k, S^x_k, R(S_k, x_k))$ in D
    if *time to learn* **then**
      Sample random minibatch of transitions $(S_m, S^x_m, R(S_m, x_m))$ from D
      if $S^x_m \neq S_K$ **then**
        $y_m \leftarrow R(S_m, x) + \gamma \hat{V}_T(S^x_m, \theta^-)$
      else
        $y_m \leftarrow R(S_m, x_m)$
      end
      Perform a batch gradient descent on $(y_m - \hat{V}(S^{x_{m-1}}_{m-1}, \theta))^2$ with respect to parameter θ
      Reset $\hat{V}_T = \hat{V}$ for every C steps
    end
    $k \leftarrow k + 1$
    $\varepsilon \leftarrow$ decay ε
  end
  $i \leftarrow i + 1$
end
return θ

5.2.2 Iterative Best Response Procedure

We propose an iterative best response procedure to compute $\text{argmax}_{x_k \in X(S_k)}$ in a decentralized manner. The purpose of incorporating this procedure is twofold. Firstly, it acts as a scalable, decentralized optimization heuristic. To compute $\text{argmax}_{x_k \in X(S_k)}$ directly is akin to solve the multi-agent optimization problem centrally (as a single-agent) which will be computationally expensive. Secondly, this procedure provides an explicit coordination mechanism amongst agents via asynchronous actions.

5.2.2.1 Optimization Heuristic

In Sections 2.7.3.2 and 4.2, we have discussed in details how optimization problem can be formulated as a n-player game where every pure-strategy equilibrium of a game is a local optimum since no player can change its strategy to improve the objective function [67]. The premise of this approach is that the problem must meet the criteria of being a potential game. Potential game possesses a pure-strategy equilibrium and has the Finite Improvement Property (FIP) [86]. Having the FIP means that every path generated by a best response procedure will converge to an equilibrium.

In the same manner, we formulate the optimization problem found in Equation 5.4 as an I-player game Γ with agents represented as players having a finite set of strategies Δ_i and sharing the same payoff function. We define the payoff function, $u^i(\delta)$ as the utility of agent i when all agents follow a joint plan δ. We deliberately remove the index k to simplify the notation. Here, we make the assumption that the payoff of each agent is not dependent on other agents' payoff such that we can define a function, $P(\delta) = \sum_{i \in I} u^i(\delta)$. This assumption is consistent with the earlier IGM assumption.

$P(\delta)$ is an *ordinal potential function* for Γ since for every $i \in I$ and for every $\delta_{-i} \in \Delta_{-i}$

$$u^i(\delta_i, \delta_{-i}) - u^i(\delta'_i, \delta_{-i}) > 0 \text{ iff}$$
$$P(\delta_i, \delta_{-i}) - P(\delta'_i, \delta_{-i}) > 0 \text{ for every } \delta_i, \delta'_i \in \Delta_i. \tag{5.6}$$

The proof can be found in Equation 4.2. Meanwhile, an equilibrium of Γ is a local optimum since no player can improve its payoff by changing its individual plan. Conversely, every optimal solution, δ^* of Γ is an equilibrium since $u^i(\delta^*) \geq u^i(\delta_i, \delta^*_{-i})$ for all $i \in I$ where $\delta_i \in B_i(\delta^*_{-i})$. Intuitively, computing optimal action while assuming the states and actions of other agents do not change will be sub-optimal. This is because other agents may take further actions in response to another agent's action and return a solution with better overall immediate and future payoffs.

To search for local optimal solution of the optimization problem in Equation 5.4, we propose the same iterative best response algorithm found in Algorithm 4. To reiterate, the key idea of this algorithm is to improve a chosen joint plan iteratively by computing the best responses of each player assuming the rest of the

players adopt the chosen joint plan until no improvement can be obtained or until terminating conditions are met. In short, this algorithm explores multiple local optimal solutions and returns the best one found. The detailed description of this algorithm can be found in Section 4.2.3.1

5.2.2.2 Explicit Coordination

Unlike communication network, the proposed iterative best response procedure induces a more explicit form of coordination as agents take turn to respond to the other agents' actions. In simple terms, knowing what other agents are doing (communication) do not necessarily mean that agents are proactively coordinating.

Although, combining the use of communication network and iterative best response seem intuitive conceptually, it is not correct methodologically. The presence of communication network means that an agent's state and action are dependent of other agents' states and actions. In other words, best response procedure will not converge because an agent can never assume other agents' plans and payoffs to be static because the moment an agent performs a best response action, the payoffs of other agents' change correspondingly.

5.2.3 Implementation Consideration

The main challenge of learning policy to make complex decision directly is that it is very computationally expensive to run numerous training episodes. To illustrate, assuming each decision takes a realistically reasonable time of 10s and there are 30 dynamic incidents per day, one simulation episode takes 10 mins. To train the model in magnitude of a thousand will take at least 10,000 mins or at least a week if training episodes are done in sequence.

To address this scalability issue, we parallelize the generation of experiences [52]. As each training episode is independent of each other, multiple episodes can be run in parallel resulting in faster learning process as more experiences are being generated at any one time. This distributed RL mechanism relies one a single learner but multiple workers to generate experiences.

5.3 Application: Multi-Agent Dynamic Police Patrol Dispatching and Rescheduling Problem

We apply this approach to solve a Multi-Agent Dynamic Police Patrol Dispatching and Rescheduling Problem (MADPRP), a real-world problem which motivates this research work. This problem is particularly challenging as the complex action includes both rerouting the sequence of locations to patrol (spatial) and rescheduling the time spent at each location (temporal).

5.3.1 Problem Description and Model

MADPRP is essentially a multi-agent version of the DPRP. As MADPRP and DPRP share many common problem descriptions and modelling features, this section will highlight additional details that are specific to MADPRP. We refer readers to Section 3.4 for detailed problem description and model formulation for the single-agent DPRP.

5.3.1.1 Problem Description

In MADPRP, there are $|I|$ police sectors in charge of patrolling $|J|$ patrol areas. We define each patrol sector as an agent; a higher-order decision-making entity which are capable of executing complex decision. Each police sector i consists of $|I_i|$ patrol teams that patrol $|J_i|$ patrol areas within its sector and each patrol shift has a duration of $|T|$ time periods. At the start of the shift, each agent is assigned to an initial patrol schedule. Throughout the shift, incidents occur dynamically and a patrol team from a certain sector is dispatched to respond to the incident which results in the need to reschedule its own and/or even the schedules of all other agents. Coordination amongst the agents is crucial as patrol teams can cross over to other sectors to respond to an incident and perform routine patrol so as to ensure that incidents are responded within target time and all patrol areas across all sectors are sufficiently patrolled.

Incident. A dynamic incident, ω_k occurs at decision epoch k and is described as the following tuple: $\langle \omega_k^i, \omega_k^j, \omega_k^t, \omega_k^s \rangle$ where ω_k^i refers to the sector in which the incident takes place, $\omega_k^j \in J_{\omega_k^i}$ refers to the location of the incident, $\omega_k^t \in T$ refers

to the time period when the incident occurs and ω_k^s refers to the number of time periods needed to resolve the incident. We assume deterministic resolution time.

Patrol Presence. On top of responding to an incident within a target time, police patrol also aims to project presence. We define patrol presence as a function of the number of effective time periods each patrol area is being patrolled in a shift. Each patrol area j needs to be patrolled for at least Q_j time periods in a given shift. We propose a presence utility function of a patrol area j, $U_p(j)$ where the utility factor of any additional patrol time periods beyond the minimum patrol time requirement decreases exponentially (see Equation 5.7)). This is to simulate that any additional patrol time periods beyond the minimum requirement are less effective in projecting presence. σ_j refers to the total patrol time periods in patrol area j by all teams across the patrol sectors in a given joint patrol schedule, $\delta(k)$.

$$U_p(j) = \min(\sigma_j, Q_j) + 1_A \times \sum_{i=1}^{\sigma_j - Q_j} i \times e^{-\beta_p i} \tag{5.7}$$

$$\text{where } 1_A = \begin{cases} 0, \sigma_j - Q_j \leq 0 \\ 1, \sigma_j - Q_j > 0 \end{cases}$$

We define a fitness function, $f_p(\delta(k))$ to quantify the goodness of a given schedule $\delta(k)$ in terms of its ability to project presence. We represent $f_p(\delta(k))$ as a ratio of total effective patrol time of to the total time in a shift across all agents and their patrol teams (see Equation 5.8). Thus, a schedule is deemed to have good patrol presence if the patrol teams spend most of the time patrolling rather than travelling between patrol areas and each patrol area is being patrolled sufficiently.

$$f_p(\delta) = \frac{\sum_{j \in J} U_p(j)}{|T| \times |I|} \tag{5.8}$$

Response Time. The response time to an incident at decision epoch k, τ_k is computed as the time taken by the assigned patrol team, x_k^m to act upon the dispatch call from the point where incident occurs $(x_k^t - \omega_k^t)$ plus the travel time from its current location to the incident location. A successful incident response happens

when $\tau_k \leq \tau_{target}$. We assume that any dispatch call must be acted upon within τ_{max}.

$$\tau_k = (x_k^t - \omega_k^t) + d(j_{t'}^{x_k^m}, \omega_k^j) \tag{5.9}$$
$$\text{where } x_k^t - \omega_k^t \leq \tau_{max}, t' = x_k^t$$

Problem Objective. The objective of the problem is to make dispatching and rescheduling decisions at every epoch that maximize the number of successful incident responses while minimizing the reduction in patrol presence within and across all sectors.

5.3.1.2 Model Formulation

Similar to DPRP, we model MADPRP as route-based MDP. As mentioned earlier, this section will only highlight additional key modelling features that are specific to MADPRP.

State. Each state can be further categorized as local and global states. Local state refers to information unique to an agent while global state refers to shared information across the agents which may be useful to induce coordination. Each local post-decision state, $S_{k,i}$ is represented as the following tuple:

$$\langle \delta_i(k), \frac{\sigma_i(k)}{Q_j}, f_{util}(\delta_i(k)) \rangle \tag{5.10}$$

where $\frac{\sigma_i(k)}{Q_j}$ is the ratio of total patrol time of each patrol area covered in $\delta_i(k)$ over its minimum patrol requirement and $f_{util}(\delta_i(k))$ is a function that compute the ratio of the total patrol time over the total shift time of all patrol teams. The shared global state includes the current time, t_k and $(\frac{\sigma_j(k)}{Q_j})_{j \in J}$ which represents the ratios of total patrol time of each patrol area in all sectors over its minimum patrol requirement when all agents follow a joint schedule, $\delta(k)$. Meanwhile, the post-decision state S_k^x captures the changes to the state upon executing a decision.

Action/Decision. x_k is the action of assigning a patrol sector/agent to dispatch one of their patrol teams to an incident and updating the joint schedule of all agents at decision epoch k. x_k is represented as the following tuple: $\langle x_k^i, x_k^m, x_k^t, \delta^x(k) \rangle$

where $x_k^i \in I$ is the sector/agent assigned to respond to the incident, $x_k^m \in I_{x_k^i}$ is the dispatched patrol team belonging to the assigned agent, $x_k^t \in T$ the time period which the assigned patrol team starts to act upon the dispatch call and $\delta^x(k)$ the resulting joint schedule after executing action x_k.

In a real-world operational setting, the disruption to the initially-planned schedule must be minimized. Similar to DPRP, we propose the use of Hamming distance to quantify the extent of disruption to the original joint schedule and this distance must be within a given threshold, P_{max}.

$$D_h(\delta^x(k), \delta(0)) < P_{max} \tag{5.11}$$

Reward Function. The reward function, $R(S_k, x_k)$ is designed in such a way that high reward is given to a successful incident response while minimizing the reduction in patrol presence at the same time. We introduce $f_r(x_k)$ to quantify the response utility after executing x_k. We propose the use of exponentially decreasing function to represent the response utility similar to the ones proposed by [2] and [89]. In other words, the later the incident is being responded, the more severe the impact of the incident and the less effective a response would be in resolving the incident. Thus, patrol teams have more incentives to respond to the incident as early as possible.

$$R(S_k, x_k) = f_r(x_k) \times f_p(\delta^x(k)) - f_p(\delta(k)) \tag{5.12}$$

$$f_r(x_k) = e^{-\beta_r \times \max(0, \tau_k - \tau_{target})} \tag{5.13}$$

5.3.2 Solution Approach

We apply our proposed cooperative MARL approach to learn the value function of joint schedules of all patrol sectors after a patrol team from a particular sector has been dispatched to attend to a dynamic incident and rescheduling actions have been performed across all sectors. In other words, the learned value function will guide the rescheduling heuristic to find dispatch and rescheduling decisions that are anticipatory and coordinated; meaning that the decisions result in a joint schedule that takes into account future occurrences of incidents both within and beyond

own sectors and where agents' schedules are coordinated and take into account both individual (sector level) and global (system level) interests. In this problem context, each agent is interested in fulfilling both its *proactive* and *reactive* patrol requirements. We propose to use the rescheduling heuristic based on ejection chains that is introduced in Section 3.4.2.2 as the planning heuristic in this problem.

5.3.2.1 MAVFA based on Value Function Factorization

We assume that the value function approximation of the joint post-decision state can be factorized into the value function approximates of each patrol sectors as stated in Equation 5.2. This IGM assumption is valid in this problem because the total number of incidents and the total routine patrol time across all police sectors are the summation of the incidents responded within every sector and the total routine patrol time of each patrol sector respectively. In other words, a system optimal decision i.e. the decision that returns the joint schedule with the best value function is achieved when each agent chooses a decision that returns agent-level schedule that has the best value function locally.

5.3.2.2 Iterative Best Response as Scalable Optimization Heuristic and Coordination Mechanism

A straightforward approach to search for the joint schedule with the best value function would be to compute $\text{argmax}_{x_k \in X(S_k)} \text{argmax}_{i \in I}$ naively by assigning an incident to each of the $|I|$ agents and followed by rescheduling the joint schedule across all agents centrally. This approach is both computationally expensive and practically not realistic. Thus, our proposed approach computes $\text{argmax}_{x_k \in X(S_k)}$ in a decentralized manner via iterative best response. The following paragraphs illustrate how the proposed iterative best response acts as a scalable, decentralized optimization heuristic to search for rescheduling actions and at the same time induces communication amongst the agents.

At a given decision epoch, an incident can be assigned to $|I|$ agents. For each agent, a single-agent DPRP is solved to determine which patrol team within a sector is assigned to an incident and the corresponding amended schedules of all patrol teams within a sector. At this stage, the schedules of the patrol teams in the other sectors remain. This current solution is sub-optimal as patrol teams from other

sectors may perform rescheduling actions and return a joint schedule with better value function. Therefore, the proposed iterative best response algorithm aims to improve this sub-optimal current solution by computing the best responses of each agent assuming the rest of the agents' schedules remain the same.

At each best response iteration, assuming other agents' schedules remain, a chosen agent will try to take best response action i.e. rescheduling action to obtain a better overall value function. This rescheduling action includes repairing defects caused by other agents i.e. patrolling other agents' patrol areas that are not patrolled sufficiently. This step is akin to agents communicating with each other and coordinate their actions to arrive at a joint schedule that has the best value function (the term best here refers to local optima and an approximate equilibrium solution (see Section 4.2.3.1 for detailed explanation)).

5.3.3 Experiments

We evaluate our approach on a problem scenario involving 3 police sectors in the North-Central region of a city-state that we reside in (see Figure 5.3). There are a total of $|J| = 62$ patrol areas where each area is represented by a 2km x 2km hexagonal grid. Each sector represents different problem complexities in terms of the ratio of patrol team per patrol area, the diversity of the patrol areas and the spatial distribution of dynamic incidents (see Table 5.1). The 3 chosen police sectors refer to Sectors A, C and D found in Section 3.4.3. Due to the classified nature of the data, synthetically-generated data based on publicly-available data sources are used in this experiment.

5.3.3.1 Experimental Setup and Design

We divide our experiment into two phases to evaluate the impact of each of the components of our proposed approach on the solution quality and computational time through a series of ablation studies. For a fairer comparison, we use the same reward function and rescheduling heuristic in all of the models.

Phase 1. We evaluate the performance of our joint learning mechanism against another approach that decomposes the problem into two stages where the first stage involves learning a dispatch policy and second stage involves executing rescheduling

Figure 5.3: Hexagonal grids drawn over 3 police sectors. Image is intentionally blurred for anonymity purposes.

Table 5.1: Different patrol sectors representing different problem structures and complexities.

Sector	**Parameter**	**Description**
1	$\|I_1\| = 4$, $\|J_1\| = 14$	High patrol team-to-area ratio, relatively homogeneous patrol densities (medium)
2	$\|I_2\| = 4$, $\|J_2\| = 23$	Low patrol team-to-area ratio, relatively homogeneous patrol densities (low)
3	$\|I_3\| = 4$, $\|J_3\| = 25$	Low patrol team-to-area ratio, more diverse patrol densities (low to high)

action. We use another popular value-based RL algorithm, DQN to learn the dispatch policy in the first stage. In addition, we also evaluate the effect of iterative best response as a coordination mechanism by comparing against another approach with a communication network. We run a total of 10,000 training episodes where each episode represents a given initial joint schedule and a set of dynamic events occurring throughout the planning horizon. We evaluate the solution quality based on the resulting cumulative rewards in terms of the average % improvement of incident success rate over myopic approach. Here are the models being run in this phase:

- **MAVFA-BR-H.** This is our proposed approach where BR refers to iterative best response procedure and H refers to heuristic
- **MAVFA-C-H.** This is a model that incorporates communication network as an implicit coordination mechanism instead of iterative best response procedure. We adapt a Attention-Based Convolutional Communication Network proposed by Jiang *et al.* [53].
- **MAVFA-H.** This is an MAVFA model without any communication mechanism amongst agents.
- **MADQN-BR-H.** This is the two-stage approach similar to Chen *et al.* [25]. In order to evaluate solely on the joint vs. two-stage learning mechanisms, we retain the same components, BR and H.

Phase 2. We evaluate the impact of our proposed MAVFA algorithm and iterative best response procedure in making anticipatory and coordinated decision-making during execution. We run 30 experiments to simulate one month's worth of daily operations and for each experiment, we run 20 different set of realizations of dynamic incidents to simulate different possible daily scenarios. There are 600 data points representing a sufficiently substantial sample size for statistical evaluation. In this phase, we evaluate the impact of our approach against the absence of coordination, collaboration and anticipation (myopic) by running the following baseline models for comparison on top of MAVFA-BR-H and MAVFA-H:

- **VFA-H.** This model assumes each sector runs its own independent single-agent VFA without any form of communication and collaboration amongst agents.
- **BR-H.** This is a version of our approach without VFA i.e. myopic approach.

We evaluate our approach in terms of its ability to make anticipatory and coordinated decision-making based on overall success rate (% of incidents responded within a target time) across all sectors and the success rate of the worst performing agent. Better coordination should result in more incidents being responded on time resulting in better overall and individual agents' success rates. For computational time, we evaluate based on the time taken per decision (dispatch and reschedule).

We include results in terms of mean and 95% Confidence Interval (CI) of the mean rather than just using a single point estimate since the problem and the heuristic used are stochastic in nature.

5.3.3.2 Size of State Space

Based on the state representation in Section 5.3.1.2, the size of the state space based on the experiment settings can be estimated as follows:

- *Time.* Based on our definition of T being in multiple of 10-minute time periods, the value of time ranges from 0 to 71.
- *Patrol Utilization.* As shown in Equation 5.10, for a given agent, the patrol utilization values ranges from $0/|T|$ to $72/|T|$ where 0 represents a scenario whereby an agent spends all the time travelling rather than patrolling while 72 represents another extreme scenario whereby an agent remains in the same patrol area throughout the whole duration of the shift.
- *Patrol Presence.* For each patrol area j, the patrol time of an agent i ranges from 0 to a maximum of $72 \times |I_i|$. 0 represents a scenario whereby agent i does not patrol area j while the upper bound value represents a scenario whereby the agent spends all its time in patrol area j.
- *Encoded Schedule.* In every sector, each patrol team's schedule is encoded into a vector with a dimension of 5 where each element can take a value between 0 to 1.

Thus, the size of state space in our experiment can be estimated to be $72^{1+|I|} \times (72 \times |I_i|)^{|I|\times|J|} \times 10^{5\times\sum_{i\in I}|I_i|}$ assuming each element of the encoded schedule is rounded off to one decimal place. The first number refers to the time and patrol utilization of each agent, the second number refers to the patrol presence of each patrol area by each team in each sector and the last number refers to the vector representation of the schedules of each patrol team in each sector.

5.3.3.3 Model Parameters

Here are the details of the implemented networks for our proposed approach:

```
embedding.weight          torch.Size([1, 64])
encoder.fc1.weight        torch.Size([128, 73])
encoder.fc1.bias          torch.Size([128])
encoder.fc2.weight        torch.Size([64, 128])
encoder.fc2.bias          torch.Size([64])
encoder.fc3.weight        torch.Size([5, 64])
encoder.fc3.bias          torch.Size([5])
local_vnetwork.fc1.weight         torch.Size([64, 86])
local_vnetwork.fc1.bias           torch.Size([64])
local_vnetwork.fc2.weight         torch.Size([32, 64])
local_vnetwork.fc2.bias           torch.Size([32])
local_vnetwork.fc3.weight         torch.Size([1, 32])
local_vnetwork.fc3.bias           torch.Size([1])
mix_network.fc1.weight    torch.Size([64, 66])
mix_network.fc1.bias      torch.Size([64])
mix_network.fc2.weight    torch.Size([32, 64])
mix_network.fc2.bias      torch.Size([32])
mix_network.fc3.weight    torch.Size([1, 32])
mix_network.fc3.bias      torch.Size([1])
```

Figure 5.4: The detailed breakdown of the sizes of the trainable weights at each layer. *fc* denotes a fully-connected layer. The sum of all the weights is 32199.

- **Encoder.** We represent the encoder network as a fully-connected neural network with 2 hidden layers with 128 and 64 nodes respectively with encoding dimension of 5.
- **Local Value Network.** We represent the encoder network as a fully-connected neural network with 2 hidden layers with 64 and 32 nodes respectively.
- **Mixing Network.** We represent the encoder network as a fully-connected neural network with 2 hidden layers with 64 and 32 nodes respectively.

We use ReLU as activation function and Adam optimizer for all the networks. There are a total of 32199 trainable parameters (see Figure 5.4 for the detailed breakdown). Meanwhile, Table 5.2 summarizes the values of the hyperparameters used.

Implementation. The implementation codes are written in Python while PyTorch is used to build and train the neural networks. To generate the initial schedules (see Section A.3), Python API of any CPLEX version that is compatible to the corresponding Python version is required. The experiments are run on a server with the following configurations: Rocky Linux 8.6, 64-Core processor and 384GB RAM. The detailed descriptions of the simulator used to run the model can be found in Appendix A.5.

Table 5.2: List of hyperparameters used in the implementation of MAVFA.

Hyperparameter	**Value**
γ	0.99
buffer size	1e5
batch size	64
τ (for soft update)	1e3
learning rate	5e-4
learn frequency	every 10 steps
update frequency	every 20 steps
ε-decay rate	25000
ε-decay start	0.9
ε-decay end	0.05

5.3.3.4 Experimental Results and Discussion

Phase 1. We observe that the cumulative rewards (represented as % improvement over myopic) stabilize after around 6000 training episodes (see Figure 5.5). We assume stability where the standard deviation of the cumulative rewards are kept within 20% of the sample mean for at least 600 consecutive episodes. Figure 5.5 shows that our explicit coordination mechanism is able to learn joint value function that results in a better overall success rate as compared to the model with an implicit communication mechanism and one without any form of communication. In terms of training time, our approach takes on average below 20 minutes/episode with 30 parallel processes (running 30 episodes concurrently) which translates to about over 100 hours for 10,000 episodes.

On the other hand, we observe that our proposed joint learning approach seems to only outperform the two-stage approach by a slight margin. However, we include the symbol * beside MADQN-BR-H to indicate that this model has been pre-trained and went through one round of training prior this experiment. This is because we observe that 10,000 episodes were not sufficient for this model to learn effectively as it is only able to achieve an average of -2% improvement over myopic. Thus, given the same number of training episodes, our proposed approach will outperform the two-stage approach by a bigger margin.

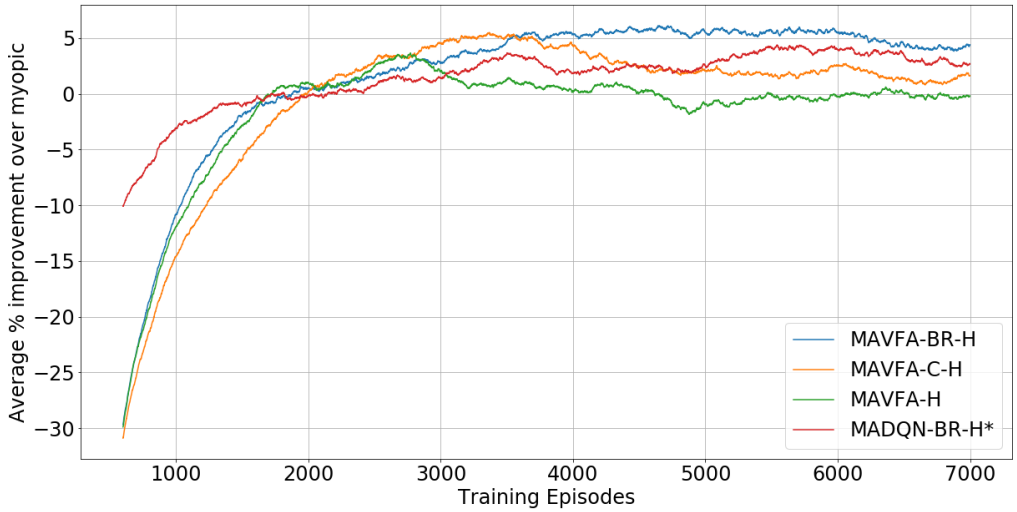


Figure 5.5: Average cumulative rewards over the last 600 training episodes.

Phase 2. Our proposed approach is statistically able to produce decisions that result in higher overall success rate (O) and success rate of the worst-performing agent (W) as compared to the other baselines except for VFA-H (see Table 5.3). Our proposed coordination mechanism account to about 13% increase in overall success rate (MAVFA-BR-H vs. MAVFA-H) while collaboration amongst agent significantly increases the success rate of the worst performing agent by more than 18% (MAVFA-BR-H vs. VFA-H).

VFA-H represents a solution approach where each agent manages its own patrol operation. The overall success rate is higher simply because Sector 1 has a higher patrol team-to-area ratio which increases its ability to respond to local incident quickly, skewing the average result. Cooperation in light of limited resources inevitably means that some sort of compromise is needed which in this, our approach is able to ensure that every sector's success rate is at least of a certain reasonable threshold ($> 50\%$).

Although our proposed approach is slower than the other models, it is still able to compute the decision within an operationally realistic time of less than $30s$ on average (see Table 5.3). In fact, our value function approximation steps and iterative best response procedure account for less than $1s$ and less than $15s$ of additional computation time per decision respectively. The computational time can be further improved via engineering means such as parallelization of the heuristic or code optimization.

Table 5.3: Our approach statistically outperforms the other models in terms of overall success rate (O) and the success rate of the worst-performing agent (W).

Model	**Success Rate**	**Time Per Decision(s)**
MAVFA-BR-H	(O) $65.0 \pm 0.7\%$	24.1 ± 2.1
	(W) $\mathbf{52.4 \pm 0.9}\%$	
MAVFA-H	(O) $57.3 \pm 0.6\%$	9.3 ± 1.0
	(W) $44.0 \pm 0.8\%$	
VFA-H	(O) $\mathbf{67.6 \pm 0.6}\%$	1.4 ± 0.2
	(W) $44.2 \pm 1.0\%$	
BR-H	(O) $62.2 \pm 0.8\%$	23.7 ± 2.1
	(W) $47.9 \pm 1.0\%$	

Discussion. The magnitude of improvements resulted from our proposed approach may not seem substantial (a 5% improvement over myopic approach). In reality however, a 5% improvement translates into 3 more incidents responded within target time, which is quite significant in the law enforcement context. Each late response to an urgent incident like traffic accident, violent crimes and suspected terrorist attack can have major adverse impact to public safety and security.

Ritzinger *et al.* [102] summarize the performances of various offline methods in the literature that solve DVRP with stochastic customers. Most approaches (mainly on single-agent problems) manage to achieve improvements in the region of $5\% - 10\%$ over myopic. Given the additional complexity of our problem (routing and scheduling) and multi-agent setting, an improvement of 5% is comparable with those cited in that study.

We note that comparison against existing cooperative MARL approaches such as VDN, QMIX and COMA would strengthen our evaluation attempt. However, such comparisons are not so straightforward. Given that these approaches require the problem or action to be decomposed into two stages, the rescheduling heuristic chosen would have to be different. In addition, these approaches assume simultaneous actions by all agents which mean that additional step is needed to ensure proper handling of dynamic event to prevent incident being ignored or more than one agent responding to one incident. Thus, modifications to these approaches such as action-masking are needed and these may result in deviations from these approaches' original design. The eventual solution quality of these modified approaches need

to be assessed more carefully as any improvement may come from the rescheduling heuristic used.

5.4 Conclusion

We presented a pioneering effort on a cooperative MARL approach to solve multi-agent dynamic routing and scheduling problem directly. In actual fact, the proposed approach can be applied to any generic multi-agent sequential decision problem with complex action. Moving ahead, there are many opportunities to further evaluate and build upon the ideas proposed. For example, our proposed approach can be evaluated on other multi-agent sequential decision problem settings. This will require exploration of more context-specific network architecture. It would also be interesting to compare our approach with existing cooperative MARL approaches. However, as mentioned earlier, additional care is required in designing the experiment to ensure fair comparison, which we hope to address in the future.

Chapter 6

Conclusion and Future Works

In this thesis, we present solution approaches that synergize techniques from AI and OR communities to address the *dynamic* and/or *multi-agent* aspects of real-world routing and scheduling problems. We focus our discussions based on the two real-world motivating problem domains namely urban logistics and law enforcement. However, we acknowledge that real-world routing and scheduling problems are wide-ranging in terms of scales, varieties and complexities and even new variants of these problems continue to surface with the ever-changing technological and industry landscapes such as the emergence of electric vehicle routing problem [5, 76] and truck-and-drone coordinated delivery [135]. These present ample opportunities for further research in applying RL to solve routing and scheduling problems. In the following sections, we highlight two major directions amongst many others for further works that can arise from the contributions of this thesis. The two directions are chosen deliberately to represent aspects of research that are peculiar to real-world problem settings.

6.1 Addressing Learning Aspect of RL-Based Approaches in Real-World Problems

As mentioned in Section 2.6.1, there have been numerous works that propose RL-based approaches to solve routing and scheduling problems (or COPs in general) in recent years. In particular, recent works are focusing more on solving real-world variants of those problems such as order batching problem [6], train time-tabling problem [72] and large-scale routing problem in logistics system [140]. However, most of the contributions of these works, including ours, are mainly revolving around methodologies to compute solution either by learning an optimal policy directly or

learning to approximate value function and combining it with a heuristic. In these works, assumptions are usually made with regard to these two key components of an RL-based approach: the reward function and the quality of training data. RL agents learn mainly from reward signal and the learnt policy is optimized based on the training data. Unlike in typical games or toy problems, real-world problems do not have clearly-defined reward function and the input training data cannot be assumed to follow a certain known static distribution. For example, in same-day delivery problem, a good solution or a reward may not necessarily be dependent only on total travel time but may include other complex criteria like how similar vehicles' routes are with each other and even intangible measure like drivers' subjective route preference. In addition, a learned policy may work generally for most instances but not on specific occasions such as special holiday sales or during the pandemic. This is because patterns of occurrences of dynamic order requests differ from occasions to occasions.

Learning reward function based on observed behaviours. One of the main challenges of application of RL to solve real-world problems will be how to define a good reward function. Reward function can be subjective, for example system-generated recommended optimal routes may not be compatible with drivers' preferences [77]. Thus, it would be interesting and practically more applicable to learn the reward function based on real, observed behaviour of the agents and this is in fact an inverse reinforcement learning problem [90].

Adaptable learning mechanism in view of dynamic environment. In our works, we assume learning a generic policy that is applicable for all occasions. This assumption is not true in reality. For instance, similar to same-day delivery context, in law enforcement context, incident patterns differ from day to day and thus, distinct policies may need to be learnt from different set of data and applied to different problem scenarios. Haliem *et al.* [49] proposes an adaptive deep RL approach that can identify and adapt to changes in the environment in ride-sharing context. The authors propose an anomaly detection-like mechanism to detect changes to the dataset to trigger relearning of policies. Such adaptive learning mechanism is very much relevant when learning to solve real-world routing and scheduling problems.

6.2 Addressing Additional Complexities of Real-World Problems

Although we try to model and solve real-world problems that closely mimic those in real-life, we make several simplification assumptions to reduce the complexities of the problem. For instance, in MADPRP, we include operational constraints such as dual patrol objectives and minimal disruption to existing schedule but there are several problem constraints or requirements that could have been taken into account such as those discussed in the next two paragraphs.

Collaboration amongst heterogeneous agents. In Chapter 5, we assume homogeneous agents and dynamic event that only requires the attention of a single agent. However, in reality, some dynamic events may require attentions from multiple agents with differing capabilities, for example in the context of disaster response [22] and even in law enforcement context, some incidents may require attention from emergency medical services or fire department. This added complexity will pose a new research challenge of learning to coordinate multiple heterogeneous agents in the presence of occurrences of a more complex dynamic event.

Learning fairness in multi-agent setting. Fairness in terms of balanced workload is important consideration in real-world multi-agent systems. For example, in patrol scheduling problem, an imbalanced workload may result in some patrol agents patrolling more locations or responding to more incidents than the others. This may in turn cause fatigue and reduce the effectiveness of those patrolling agents. Similarly, in urban logistics context, some drivers may be required to serve more orders than the others which in turn result in fatigue and unfair work distribution. Thus, taking into account fairness in solving real-world multi-agent routing and scheduling problems would result in a more realistic and effective solution. However, learning to optimize both system performance (joint reward) and fairness at the same time is a complex optimization problem [55] and would require a novel research contribution.

Bibliography

[1] L. Agussurja, S.-F. Cheng, and H. C. Lau, "A state aggregation approach for stochastic multiperiod last-mile ride-sharing problems", *Transportation Science*, vol. 53, no. 1, pp. 148–166, 2019.

[2] S. Amador, S. Okamoto, and R. Zivan, "Dynamic multi-agent task allocation with spatial and temporal constraints", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, 2014.

[3] N. Azi, M. Gendreau, and J.-Y. Potvin, "A dynamic vehicle routing problem with multiple delivery routes", *Annals of Operations Research*, vol. 199, no. 1, pp. 103–112, 2012.

[4] D. Barbucha and P. Jedrzejowicz, "Agent-based approach to the dynamic vehicle routing problem", in *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, Springer, 2009, pp. 169–178.

[5] R. Basso, B. Kulcsár, I. Sanchez-Diaz, and X. Qu, "Dynamic stochastic electric vehicle routing with safe reinforcement learning", *Transportation research part E: logistics and transportation review*, vol. 157, p. 102 496, 2022.

[6] M. Beeks, R. R. Afshar, Y. Zhang, R. Dijkman, C. van Dorst, and S. de Looijer, "Deep reinforcement learning for a multi-objective online order batching problem", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 435–443.

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

[8] R. Bent and P. Van Hentenryck, "The value of consensus in online stochastic scheduling." In *ICAPS*, vol. 4, 2004, pp. 219–226.

[9] R. W. Bent and P. Van Hentenryck, "Scenario-based planning for partially dynamic vehicle routing with stochastic customers", *Operations Research*, vol. 52, no. 6, pp. 977–987, 2004.

[10] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu, "Rollout algorithms for combinatorial optimization", *Journal of Heuristics*, vol. 3, no. 3, pp. 245–262, 1997.

[11] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs", in *International Conference on Machine Learning*, PMLR, 2020, pp. 980–991.

[12] C. Boutilier, "Sequential optimality and coordination in multiagent systems", in *Proceedings of the 16th international joint conference on Artifical intelligence-Volume 1*, 1999, pp. 478–485.

[13] C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations", *Artificial intelligence*, vol. 121, no. 1-2, pp. 49–107, 2000.

[14] A. Bracher, N. Frohner, and G. R. Raidl, "Learning surrogate functions for the short-horizon planning in same-day delivery problems", in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2021, pp. 283–298.

[15] R. I. Brafman and C. Domshlak, "From one to many: Planning for loosely coupled multi-agent systems", in *Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling*, 2008, pp. 28–35.

[16] R. I. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz, "Planning games." In *IJCAI*, Citeseer, 2009, pp. 73–78.

[17] G. W. Brown, "Iterative solution of games by fictitious play", *Activity analysis of production and allocation*, vol. 13, no. 1, pp. 374–376, 1951.

[18] M. Brown, S. Saisubramanian, P. Varakantham, and M. Tambe, "Streets: Game-theoretic traffic patrolling with exploration and exploitation", in *Twenty-Sixth IAAI Conference*, 2014.

[19] E. K. Burke and T. Curtois, "An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010", in *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling PATAT*, Citeseer, vol. 10, 2010, p. 13.

[20] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan, "Rich vehicle routing problem: Survey", *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–28, 2014.

[21] E. Campos-Nañez, A. Garcia, and C. Li, "A game-theoretic approach to efficient power management in sensor networks", *Operations Research*, vol. 56, no. 3, pp. 552–561, 2008.

[22] L. Capezzuto, D. Tarapore, and S. D. Ramchurn, "Anytime and efficient multi-agent coordination for disaster response", *SN Computer Science*, vol. 2, no. 3, pp. 1–15, 2021.

[23] J. Chase, T. Phong, K. Long, T. Le, and H. C. Lau, "Grand-vision: An intelligent system for optimized deployment scheduling of law enforcement agents", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 459–467.

[24] C. Chen, S.-F. Cheng, and H. C. Lau, "Multi-agent orienteering problem with time-dependent capacity constraints", *Web Intelligence and Agent Systems: An International Journal*, vol. 12, no. 4, pp. 347–358, 2014.

[25] J. Chen, A. K. Umrawal, T. Lan, and V. Aggarwal, "Deepfreight: A model-free deep-reinforcement-learning-based algorithm for multi-transfer freight delivery", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 510–518.

[26] X. Chen, M. W. Ulmer, and B. W. Thomas, "Deep q-learning for same-day delivery with vehicles and drones", *European Journal of Operational Research*, vol. 298, no. 3, pp. 939–952, 2022.

[27] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization", *Advances in Neural Information Processing Systems*, vol. 32, pp. 6281–6292, 2019.

[28] Y. Chen, Y. Qian, Y. Yao, Z. Wu, R. Li, Y. Zhou, H. Hu, and Y. Xu, "Can sophisticated dispatching strategy acquired by reinforcement learning?" In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 1395–1403.

[29] W.-C. Chiang and R. A. Russell, "Simulated annealing metaheuristics for the vehicle routing problem with time windows", *Annals of Operations Research*, vol. 63, no. 1, pp. 3–27, 1996.

[30] D. P. Cuervo, C. Vanovermeire, and K. Sörensen, "Determining collaborative profits in coalitions formed by two partners with varying characteristics", *Transportation Research Part C: Emerging Technologies*, vol. 70, pp. 171–184, 2016.

[31] B. Dai and H. Chen, "A multi-agent and auction-based framework and approach for carrier collaboration", *Logistics Research*, vol. 3, no. 2-3, pp. 101–120, 2011.

[32] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem", *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[33] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "Tarmac: Targeted multi-agent communication", in *International Conference on Machine Learning*, PMLR, 2019, pp. 1538–1546.

[34] A. De Filippo, M. Lombardi, and M. Milano, "Methods for off-line/on-line optimization under uncertainty." In *IJCAI*, 2018, pp. 1270–1276.

[35] F. De Nijs, M. T. Spaan, and M. M. de Weerdt, "Best-response planning of thermostatically controlled loads under power constraints", in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[36] M. Dewinter, C. Vandeviver, T. Vander Beken, and F. Witlox, "Analysing the police patrol routing problem: A review", *ISPRS International Journal of Geo-Information*, vol. 9, no. 3, p. 157, 2020.

[37] M. Drexl, "Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints", *Transportation Science*, vol. 46, no. 3, pp. 297–316, 2012.

[38] J. K. Falkner and L. Schmidt-Thieme, "Learning to solve vehicle routing problems with time windows through joint attention", *arXiv preprint arXiv:2006.09100*, 2020.

[39] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: A survey", *Journal of Artificial Intelligence Research*, vol. 61, pp. 623–698, 2018.

[40] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[41] M. Gansterer and R. F. Hartl, "Collaborative vehicle routing: A survey", *European Journal of Operational Research*, vol. 268, no. 1, pp. 1–12, 2018.

[42] L. Gao, M. Chen, Q. Chen, G. Luo, N. Zhu, and Z. Liu, "Learn to design the heuristics for vehicle routing problem", *arXiv preprint arXiv:2002.08539*, 2020.

[43] A. Garcia, D. Reaume, and R. L. Smith, "Fictitious play for finding system optimal routings in dynamic traffic networks", *Transportation Research Part B: Methodological*, vol. 34, no. 2, pp. 147–156, 2000.

[44] F. Glover, "Ejection chains, reference structures and alternating path methods for traveling salesman problems", *Discrete Applied Mathematics*, vol. 65, no. 1-3, pp. 223–253, 1996.

[45] T. F. Gonzalez, "Handbook of approximation algorithms and metaheuristics", Chapman and Hall/CRC, 2007.

[46] J. C. Goodson, B. W. Thomas, and J. W. Ohlmann, "A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs", *European Journal of Operational Research*, vol. 258, no. 1, pp. 216–229, 2017.

[47] M. Guajardo, M. Rönnqvist, P. Flisberg, and M. Frisk, "Collaborative transportation with overlapping coalitions", *European Journal of Operational Research*, vol. 271, no. 1, pp. 238–249, 2018.

[48] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored mdps", *Journal of Artificial Intelligence Research*, vol. 19, pp. 399–468, 2003.

[49] M. Haliem, V. Aggarwal, and B. Bhargava, "Adapool: A diurnal-adaptive fleet management framework using model-free deep reinforcement learning and change point detection", *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2471–2481, 2021.

[50] F. D. Hildebrandt, B. Thomas, and M. W. Ulmer, "Where the action is: Let's make reinforcement learning for stochastic dynamic vehicle routing problems work!" *arXiv preprint arXiv:2103.00507*, 2021.

[51] K. L. Hoffman and T. K. Ralphs, "Integer and combinatorial optimization", *Encyclopedia of Operations Research and Management Science*, pp. 771–783, 2013.

[52] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay", in *International Conference on Learning Representations*, 2018.

[53] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning", in *International Conference on Learning Representations*, 2020. [Online]. Available: `https://openreview.net/forum?id=HkxdQkSYDB`.

[54] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation", *Advances in neural information processing systems*, vol. 31, 2018.

[55] J. Jiang and Z. Lu, "Learning fairness in multi-agent systems", in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 13 854–13 865.

[56] W. Joe and H. C. Lau, "Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 394–402.

[57] W. Joe and H. C. Lau, "Coordinating multi-party vehicle routing with location congestion via iterative best response", in *Multi-Agent Systems: 18th European Conference, EUMAS 2021, Virtual Event, June 28–29, 2021, Revised Selected Papers*, Springer Nature, 2021, p. 72.

[58] W. Joe, H. C. Lau, and J. Pan, "Reinforcement learning approach to solve dynamic bi-objective police patrol dispatching and rescheduling problem", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 453–461.

[59] A. Jonsson and M. Rovatsos, "Scaling up multiagent planning: A best-response approach", in *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[60] B. B. Keskin, S. R. Li, D. Steil, and S. Spiller, "Analysis of an integrated maximum covering and patrol routing problem", *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 215–232, 2012.

[61] J. H. Kingston, "Repairing high school timetables with polymorphic ejection chains", *Annals of Operations Research*, vol. 239, no. 1, pp. 119–134, 2016.

[62] M. A. Klapp, A. L. Erera, and A. Toriello, "The one-dimensional dynamic dispatch waves problem", *Transportation Science*, vol. 52, no. 2, pp. 402–415, 2018.

[63] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations*, 2018.

[64] E. Lam and P. Van Hentenryck, "A branch-and-price-and-check model for the vehicle routing problem with location congestion", *Constraints*, vol. 21, no. 3, pp. 394–412, 2016.

[65] E. Lam, P. Van Hentenryck, and P. Kilby, "Joint vehicle and crew routing and scheduling", *Transportation Science*, vol. 54, no. 2, pp. 488–511, 2020.

[66] T. J. Lambert and H. Wang, "Fictitious play approach to a mobile unit situation awareness problem", *Univ. Michigan, Tech. Rep*, 2003.

[67] T. J. Lambert Iii, M. A. Epelman, and R. L. Smith, "A fictitious play approach to large-scale optimization", *Operations Research*, vol. 53, no. 3, pp. 477–489, 2005.

[68] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, p. 436, 2015.

[69] J. Leigh, S. Dunnett, and L. Jackson, "Predictive police patrolling to target hotspots and cover response demand", *Annals of Operations Research*, vol. 283, no. 1, pp. 395–410, 2019.

[70] B. Li, G. Wu, Y. He, M. Fan, and W. Pedrycz, "An overview and experimental study of learning-based optimization algorithms for vehicle routing problem", *arXiv preprint arXiv:2107.07076*, 2021.

[71] J. Li, Y. Ma, R. Gao, Z. Cao, A. Lim, W. Song, and J. Zhang, "Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem", *IEEE Transactions on Cybernetics*, 2021.

[72] W. Li and S. Ni, "Train timetabling with the general learning environment and multi-agent deep reinforcement learning", *Transportation Research Part B: Methodological*, vol. 157, pp. 230–251, 2022.

[73] X. Li, W. Luo, M. Yuan, J. Wang, J. Lu, J. Wang, J. Lü, and J. Zeng, "Learning to optimize industry-scale dynamic pickup and delivery problems", in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, IEEE, 2021, pp. 2511–2522.

[74] Y. Li, H. Chen, and C. Prins, "Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests", *European Journal of Operational Research*, vol. 252, no. 1, pp. 27–38, 2016.

[75] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search", in *Advances in Neural Information Processing Systems*, 2018, pp. 539–548.

[76] B. Lin, B. Ghaddar, and J. Nathwani, "Deep reinforcement learning for the electric vehicle routing problem with time windows", *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[77] S. Liu, H. Jiang, S. Chen, J. Ye, R. He, and Z. Sun, "Integrating dijkstra's algorithm into deep inverse reinforcement learning for food delivery route planning", *Transportation Research Part E: Logistics and Transportation Review*, vol. 142, p. 102 070, 2020.

[78] J. Los, F. Schulte, M. Gansterer, R. F. Hartl, M. T. Spaan, and R. R. Negenborn, "Decentralized combinatorial auctions for dynamic and large-scale collaborative vehicle routing", in *International Conference on Computational Logistics*, Springer, 2020, pp. 215–230.

[79] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments", *Advances in neural information processing systems*, vol. 30, 2017.

[80] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems", in *International Conference on Learning Representations*, 2019.

[81] Y. Ma, X. Hao, J. Hao, J. Lu, X. Liu, T. Xialiang, M. Yuan, Z. Li, J. Tang, and Z. Meng, "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems", *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 609–23 620, 2021.

[82] S. Martin, D. Ouelhadj, P. Beullens, E. Ozcan, A. A. Juan, and E. K. Burke, "A multi-agent based cooperative approach to scheduling and routing", *European Journal of Operational Research*, vol. 254, no. 1, pp. 169–178, 2016.

[83] S. Meisel, "Anticipatory optimization for dynamic decision making", Springer Science & Business Media, 2011, vol. 51.

[84] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[85] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, p. 529, 2015.

[86] D. Monderer and L. S. Shapley, "Potential games", *Games and economic behavior*, vol. 14, no. 1, pp. 124–143, 1996.

[87] A. Mukhopadhyay, C. Zhang, Y. Vorobeychik, M. Tambe, K. Pence, and P. Speer, "Optimal allocation of police patrol resources using a continuous-time crime model", in *International Conference on Decision and Game Theory for Security*, Springer, 2016, pp. 139–158.

[88] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem", in *Advances in Neural Information Processing Systems*, 2018, pp. 9839–9849.

[89] S. A. Nelke, S. Okamoto, and R. Zivan, "Market clearing–based dynamic multi-agent task allocation", *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 1, pp. 1–25, 2020.

[90] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning", in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 663–670.

[91] R. Nissim, R. I. Brafman, and C. Domshlak, "A general, fully distributed multi-agent planning algorithm", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, 2010, pp. 1323–1330.

[92] J. Oren, C. Ross, M. Lefarov, F. Richter, A. Taitler, Z. Feldman, D. Di Castro, and C. Daniel, "Solo: Search online, learn offline for combinatorial optimization problems", in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, 2021, pp. 97–105.

[93] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of Operations Research*, vol. 41, no. 4, pp. 421–451, 1993.

[94] B. Peng, M. Liu, Z. Lü, G. Kochengber, and H. Wang, "An ejection chain approach for the quadratic multiple knapsack problem", *European Journal of Operational Research*, vol. 253, no. 2, pp. 328–336, 2016.

[95] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems", *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[96] W. B. Powell, "Approximate Dynamic Programming: Solving the Curses of Dimensionality", John Wiley & Sons, 2011, vol. 842.

[97] W. B. Powell, "A unified framework for stochastic optimization", *European Journal of Operational Research*, vol. 275, no. 3, pp. 795–821, 2019.

[98] H. N. Psaraftis, M. Wen, and C. A. Kontovas, "Dynamic vehicle routing problems: Three decades and counting", *Networks*, vol. 67, no. 1, pp. 3–31, 2016.

[99] M. Qi, W.-H. Lin, N. Li, and L. Miao, "A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows", *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 248–257, 2012.

[100] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning", in *International Conference on Machine Learning*, PMLR, 2018, pp. 4295–4304.

[101] C. Rego, "A subpath ejection method for the vehicle routing problem", *Management science*, vol. 44, no. 10, pp. 1447–1459, 1998.

[102] U. Ritzinger, J. Puchinger, and R. F. Hartl, "A survey on dynamic and stochastic vehicle routing problems", *International Journal of Production Research*, vol. 54, no. 1, pp. 215–231, 2016.

[103] S. Ropke and J.-F. Cordeau, "Branch and cut and price for the pickup and delivery problem with time windows", *Transportation Science*, vol. 43, no. 3, pp. 267–286, 2009.

[104] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows", *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.

[105] A. Rosenfeld and S. Kraus, "When security games hit traffic: Optimal traffic enforcement under one sided uncertainty", in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 3814–3822.

[106] J. Ruan, L. Meng, X. Xiong, D. Xing, and B. Xu, "Learning multi-agent action coordination via electing first-move agent", in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 624–628.

[107] S. Samanta, G. Sen, and S. K. Ghosh, "A literature review on police patrolling problems", *Annals of Operations Research*, pp. 1–44, 2021.

[108] N. Secomandi, "A rollout policy for the vehicle routing problem with stochastic demands", *Operations Research*, vol. 49, no. 5, pp. 796–802, 2001.

[109] P. Shaw, "A new local search algorithm providing high quality solutions to vehicle routing problems", *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[110] M. A. L. Silva, S. R. de Souza, M. J. F. Souza, and A. L. C. Bazzan, "A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems", *Expert Systems with Applications*, vol. 131, pp. 148–171, 2019.

[111] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge", *Nature*, vol. 550, no. 7676, p. 354, 2017.

[112] N. Soeffker, M. W. Ulmer, and D. C. Mattfeld, "Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review", *European Journal of Operational Research*, 2021.

[113] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning", in *International Conference on Machine Learning*, PMLR, 2019, pp. 5887–5896.

[114] N. N. Sultana, V. Baniwal, A. Basumatary, P. Mittal, S. Ghosh, and H. Khadilkar, "Fast approximate solutions using reinforcement learning for dynamic capacitated vehicle routing with time windows", *arXiv preprint arXiv:2102.12088*, 2021.

[115] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward", in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 2085–2087.

[116] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy", *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–32, 2021.

[117] A. Torreño, E. Onaindia, A. Komenda, and M. Štolba, "Cooperative multi-agent planning: A survey", *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[118] P. Toth and D. Vigo, "The vehicle routing problem", SIAM, 2002.

[119] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and M. Hennig, "Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests", *Transportation Science*, vol. 53, no. 1, pp. 185–202, 2018.

[120] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and B. W. Thomas, "Route-based markov decision processes for dynamic vehicle routing problems", Technical report, Braunschweig, Tech. Rep., 2017.

[121] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and B. W. Thomas, "On modeling stochastic dynamic vehicle routing problems", *EURO Journal on Transportation and Logistics*, vol. 9, no. 2, p. 100 008, 2020.

[122] M. W. Ulmer, D. C. Mattfeld, and F. Köster, "Budgeting time for dynamic vehicle routing with stochastic customer requests", *Transportation Science*, vol. 52, no. 1, pp. 20–37, 2017.

[123] M. W. Ulmer, B. W. Thomas, A. M. Campbell, and N. Woyak, "The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times", *Transportation Science*, vol. 55, no. 1, pp. 75–100, 2021.

[124] M. W. Ulmer, B. W. Thomas, and D. C. Mattfeld, "Preemptive depot returns for dynamic same-day delivery", *EURO Journal on Transportation and Logistics*, pp. 1–35, 2018.

[125] P. Varakantham, H. C. Lau, and Z. Yuan, "Scalable randomized patrolling for securing rapid transit networks", in *Twenty-Fifth IAAI Conference*, 2013.

[126] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks", in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[127] S. A. Voccia, A. M. Campbell, and B. W. Thomas, "The same-day delivery problem for online purchases", *Transportation Science*, vol. 53, no. 1, pp. 167–184, 2017.

[128] J. Vokrínek, A. Komenda, and M. Pěchoucek, "Agents towards vehicle routing problems." In *AAMAS*, 2010, pp. 773–780.

[129] W. Wang, Z. Dong, B. An, and Y. Jiang, "Toward efficient city-scale patrol planning using decomposition and grafting", *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[130] X. Wang and H. Kopfer, "Dynamic collaborative transportation planning: A rolling horizon planning approach", in *International Conference on Computational Logistics*, Springer, 2013, pp. 128–142.

[131] X. Wang and H. Kopfer, "Collaborative transportation planning of less-than-truckload freight", *OR spectrum*, vol. 36, no. 2, pp. 357–380, 2014.

[132] Y. Wang, L. Lei, D. Zhang, and L. H. Lee, "Towards delivery-as-a-service: Effective neighborhood search strategies for integrated delivery optimization of e-commerce and static o2o parcels", *Transportation Research Part B: Methodological*, vol. 139, pp. 38–63, 2020.

[133] G. Weiss, "Multiagent systems", The MIT Press Cambridge, 2013.

[134] M. Wooldridge, "An introduction to multiagent systems", John wiley & sons, 2009.

[135] G. Wu, M. Fan, J. Shi, and Y. Feng, "Reinforcement learning based truck-and-drone coordinated delivery", *IEEE Transactions on Artificial Intelligence*, 2021.

[136] L. Xin, W. Song, Z. Cao, and J. Zhang, "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems", in *Proceedings of 35th AAAI Conference on Artificial Intelligence*, 2021.

[137] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning", *Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.

[138] K. Zhang, F. He, Z. Zhang, X. Lin, and M. Li, "Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach", *Transportation Research Part C: Emerging Technologies*, vol. 121, p. 102 861, 2020.

[139] W. Zhang and T. G. Dietterich, "Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling", *Journal of Artificial Intelligence Reseach*, vol. 1, pp. 1–38, 2000.

[140] Z. Zong, H. Wang, J. Wang, M. Zheng, and Y. Li, "Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning", in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4648–4658.

Appendix A

Supplementary Materials

A.1 Approximate Value Iteration

Algorithm 7: Approximate Value Iteration

Input : Initial Values $\hat{V}_0$, No. Simulation Runs N, Step Size α
Output : Values $\hat{V}_N$

$i = 1$
while $i \leq N$ **do**
  $k = 1$
  while $S_k \neq S_K$ **do**
    if *new order = True* **then**
      $S_k \leftarrow (S^x_{k-1}, \omega)$
      // use SA to find the optimal feasible decision x_k
      $x_k \leftarrow \arg\min_{x \in X(S_k)} \{R(S_k, x) + \gamma \hat{V}_{i-1}(S^x_k)\}$
      // proceed with the updated route
    else
      // proceed with the existing route
      $S^x_k \leftarrow S_k$
    end
    $S^x \leftarrow S^x \cup \{S^x_k\}$
    $R_k \leftarrow R_{k-1} + R(S_k, x)$
    $k \leftarrow k + 1$
  end
  // Update the Value Lookup Table
  forall $S^x_k \in S^*$ **do**
    $\hat{V}_i(S^x_k) \leftarrow (1 - \alpha)\hat{V}_{i-1}(S^x_k) + \alpha(R_K - R_k)$
  end
  $i \leftarrow i + 1$
end
return $\hat{V}_N$

The above algorithm is adapted for the problem discussed in Section 3.3. The value function approximate is only updated at the end of every episode (line 16). This corresponds to a Monte Carlo method of learning value function. α refers to learning step size and in-depth discussion on tuning this parameter can be found in Chapter 11 of [96].

A.2 Multiple Scenario Approach

Algorithm 8: Multiple Scenario Approach with Consensus Algorithm

Input : Initial State S_0, Horizon H, Sample Size J
Output : Total Cost $Cost(S_K)$, Final State S_K

1: $k = 1$;
2: **while** $S_k \neq S_K$ **do**
3: | **if** *new order = True* **then**
4: | | $S_k \leftarrow (S^x_{k-1}, \omega)$
5: | | $j = 1$
6: | | $X^* = [\ ]$
7: | | **while** $j \leq J$ **do**
8: | | | $\Omega_j \leftarrow SAMPLE(S_k, H)$
9: | | | $x^*_j \leftarrow OPTIMIZE(S_k, \Omega_j)$
10: | | | $x^*_j \leftarrow x^*_j \setminus \Omega_j$
11: | | | $X^* \cup x^*_j$
12: | | | $j = j + 1$
13: | | **end**
14: | | $x^* \leftarrow \arg\min_{x^*_j \in X^*} Cost(S_k, x^*_j)$
| | // proceed with the updated route
15: | | $S^x_k \leftarrow (S_k, x^*)$
16: | **else**
| | // proceed with the existing route
17: | | $S^x_k \leftarrow S_k$
18: | **end**
19: | $k \leftarrow k + 1$
20: **end**
21: **return** $Cost(S_K), S_K$

Whenever a dynamic event occurs (a new order appears), an empty list to store samples of optimal routes is created (line 6). For every sample, $SAMPLE$ function will produce a set of future new orders until a given time $k+H$ based on current state, S_k (line 8). $OPTIMIZE$ function will then optimize (using Simulated Annealing)

the route assuming all sampled new orders are known beforehand (line 9). A resulting optimal route without the sampled new orders is added to the list X^* and the cost of each optimal route is being tracked (lines 10 and 11). The optimal route which returns the lowest average total cost is be then chosen to update the current route (lines 14 and 15). Similar to Algorithm 7, this algorithm is adapted for the problem found in Section 3.3.

A.3 Integer Programming Model for Static Police Patrol Scheduling Problem

To derive the initial patrol schedules for the experiment in Section 3.4, we formulate and solve the following integer programming model that represents a set cover model of a police patrol scheduling problem.

Parameters

I = a set of agents, $I \in \{1, 2, ..., |I|\}$

J = a set of patrol areas, $J \in \{1, 2, ..., |J|\}$

T = a set of time periods in a shift, $T \in \{1, 2, ..., |T|\}$

Q_j = minimum patrol time (in terms of time period) for patrol area j

τ_{target} = a response time target

$d(j, j')$ = travel time from patrol area j to another patrol area j'

N_j = a set of neighbouring patrol areas within τ_{target} from patrol area j

S = a set of scenario

Ω_s = a set of incidents in scenario s

ω_s = incident in scenario s, $\omega_s \in \Omega_s$, $\omega_s = \langle \omega_s^j, \omega_s^t, \omega_s^g \rangle$ where ω_s^j = location (patrol area) of the incident, ω_s^t = start time of the incident and ω_s^g = resolution time of the incident

Decision Variables

$z_{s,\omega}$ = set to 1 if incident ω in scenario s is covered or 0 if otherwise

$y_{i,j,t}$ = set to 1 if agent i is patrolling at patrol area j at time period t

Objective Function

$$maximize \frac{\sum_{s \in S} \sum_{\omega \in \Omega_s} z_{s,\omega}}{\sum_{s \in S} |\Omega_s|} + \frac{\sum_{i \in I} \sum_{j \in J} \sum_{t \in T} y_{i,j,t}}{|T| \times |I|} \tag{A.1}$$

subjected to:

$$\sum_{j \in J} y_{i,j,t} \leq 1, i \in I, t \in T \tag{A.2}$$

$$\sum_{t \in T} \sum_{i \in I} y_{i,j,t} \geq Q_j, j \in J \tag{A.3}$$

$$t + d(j, j') - t' \leq M(2 - y_{i,j,t} - y_{i,j',t'}), i \in I, j, j' \in J, t < t' \in T \tag{A.4}$$

$$\sum_{i \in I} \sum_{j \in N_{\omega_s^j}} y_{i,j,\omega_s^t} \geq z_{s,r}, s \in S, \omega_s \in \Omega_s \tag{A.5}$$

The objective of the problem is to maximize the proportion of incidents successfully responded within the response time target plus the proportion of time spent by each agent patrolling on a given shift period. Constraint A.2 ensures that every agent can only be patrolling at one patrol area at any one time. Constraint A.3 ensures that each patrol area is being patrolled for at least a required number of time periods. Constraint A.4 ensures that there is enough time periods in between two consecutive patrol areas to cater for travel time. Lastly, Constraint A.5 ensures that an incident is covered if there is at least one agent patrolling in a patrol area within the response time target from the incident location.

A.4 Repair Operations in Rescheduling Heuristic based on Ejection Chain

In the proposed rescheduling heuristic based on ejection chain found in Section 3.4, we introduce two repair operators namely *Insert* and *Replace* to repair Type 1 defect. In the *Insufficient* case (see Figure A.1), the *Insert* operator will insert the necessary travel time period(s) (in this example, additional 1 time period is required) into the schedule by pushing the destination patrol area to a later time period as illustrated in neighbouring schedule n_1. Meanwhile, *Replace* operator replaces the infeasible destination patrol area to feasible ones. In Figure A.1, patrol area 5 can be replaced by patrol areas 2 or 4 because these patrol areas are 1 time period away from the origin patrol area i.e. patrol area 1 (see n_2 and n_3).

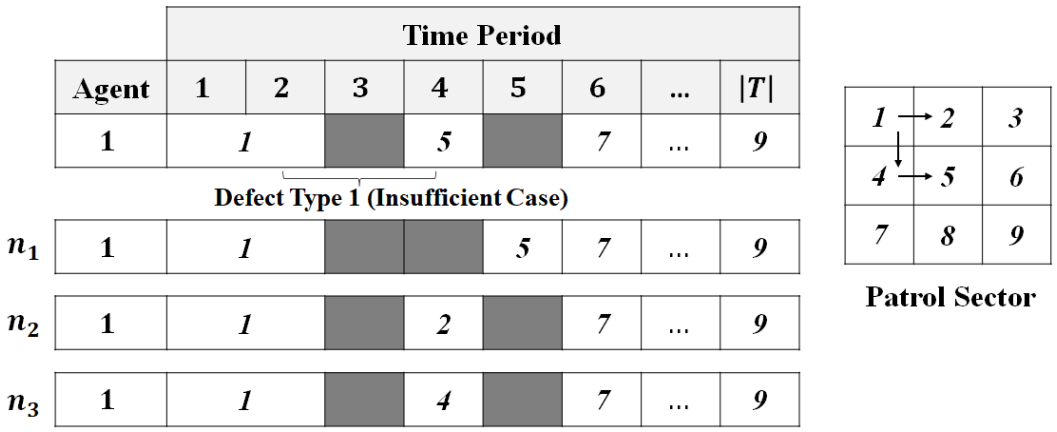


Figure A.1: Sample schedule with Type 1 Defect (*Insufficient*) case and corresponding repair operations.

In the *Excess* case, the *Insert* operator will insert either the origin or destination patrol area to the excess time period. In Figure A.2, only 1 time period needed to travel between patrol area 1 to patrol area 4. Thus, 1 excess time period can be replaced by either the origin (see n_1) or destination patrol area (see n_2). Meanwhile, the *Replace* operator will replace the destination patrol area with feasible ones. For example, patrol area 4 can be replaced by patrol area 3 or 7 since these patrol areas are 2 time periods away from patrol area 1 (see n_3 and n_4).

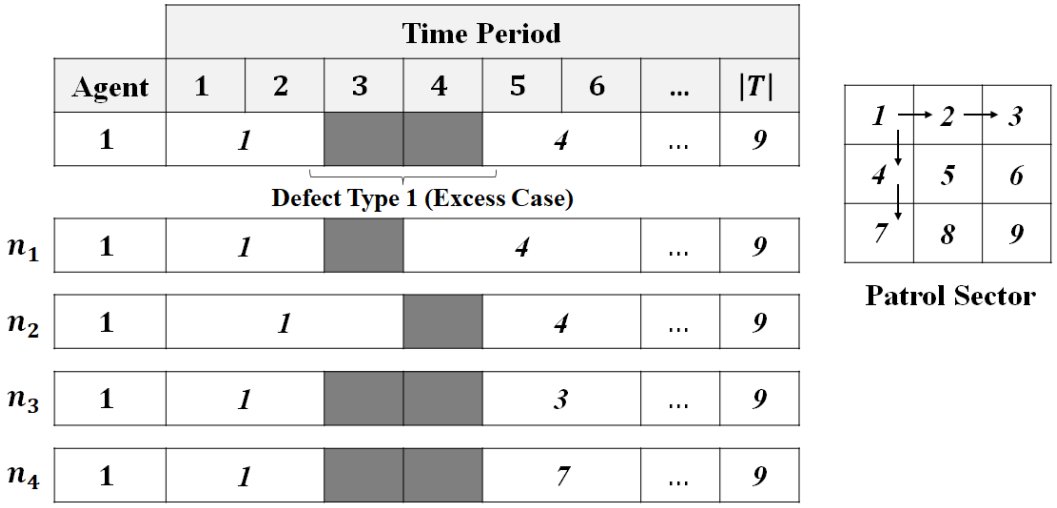


Figure A.2: Sample schedule with Type 1 Defect (*Excess*) case and corresponding repair operations.

A.5 Simulator for Dynamic Routing and Scheduling Problem

The simulator for dynamic routing and scheduling problem used in the experiments in this thesis is implemented in Python. In this section, we describe the general framework of the simulator setup and how it works. The detailed implementations (such as the environment setup, planning heuristic used and neural network architecture) and the parameters are problem-specific and those information can be found in the respective experiment sections of the problems being run in this thesis.

A.5.1 Input Data

The simulator takes in 3 key input components mainly the number of episodes to run, the initial routes and schedules of all agent and the scenarios to run for every episode. In other words, for each training or testing episode, the input to the simulator must include the initial plans of every agent and the dynamic event scenarios which include the arrival times of every dynamic event and the events themselves (implemented as an *order* object or *incident* object).

Data Source. The initial plans and the scenarios can be obtained from historical data or can be generated synthetically. For the SDDP problem in Section 3.3, the initial routes of every vehicle is obtained from a local LSP that we work with. However, since the available data does not include the detailed information of the dynamic orders, we synthetically extract some orders to be dynamically inserted during the planning horizon with a pre-determined rate of arrival of dynamic orders (see Figure A.3). Meanwhile for both DPRP and MADPRP, due to the sensitive nature of the data, we synthetically-generate these input data based on publicly-available data sources. The arrival of dynamic event is modelled as Poisson Process with a pre-determined λ as rate of occurrence of the event.

A.5.2 Simulation Run

For each training or testing episode, the simulator will initialize the routes or schedules of every agent with the chosen instance data consisting of initial plans and the dynamic event scenarios. The simulator will then iterate every pre-determined

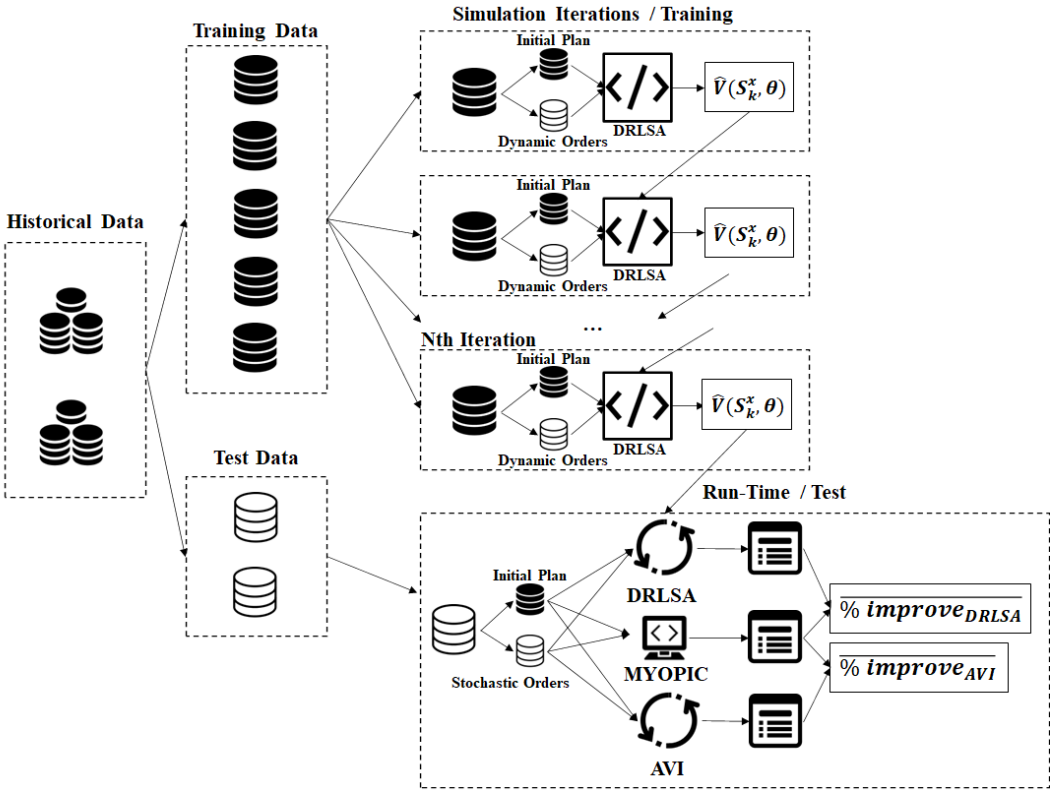


Figure A.3: Dynamic orders are generated by extracting a subset of orders and synthetically insert them during the planning horizon to simulate arrival of new dynamic orders.

time step/interval (as defined in the environment setup) until the end of planning horizon. At every time step, if there is no dynamic event occurring, the simulator will then proceed to the next time step, otherwise, it will trigger the event-handling and re-planning step. This is done until the end of planning horizon or until terminal state is reached. At the end of the run, important summary statistics (for e.g. computational time, total number of orders fulfilled, etc) are saved for evaluation.

Event-Handling and Replanning Step. Our proposed RL approach is implemented and executed during this step. The codes are implemented in a modular way such that we can use any models, both our proposed approach or other baseline models, to compute the event-handling and re-planning decisions. The output of this step is an updated route or schedule of every agent.

State Representation. A function is created to transform or aggregate the raw forms of the initial plans and dynamic order into the respective state representation format (for e.g. see Equation 3.3.2.3).