University of Nottingham
UK | CHINA | MALAYSIA

Deep Reinforcement Learning Hyper-heuristics for Online Marine Port Truck Dispatching

by

Yuchang Zhang, BSc, MSc

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

June 2022

Abstract

Real-world combinatorial optimisation problems (COPs) faced in ports are often subject to dynamics and uncertainties, i.e. the problem instance is not completely known in advance. Such problems are commonly called online combinatorial optimisation problems, in which uncertainties are often sequentially revealed over time during decision-making. In such situations, solutions generated as a priori often encounter various issues, consisting of inferior service quality, increased costs, and solution infeasibility, all of which would lead to substantial losses. This thesis is concerned with using Deep Reinforcement Learning (DRL)-based hyper-heuristics to solve online container truck dispatching problems in ports. The proposed method has comparatively better abilities to both exploit the special structures of the problem and discover the hidden patterns of the problem uncertainties from historical data. Specifically, two real-world online truck routing problems with progressively increasing difficulties drawn from a container truck terminal at an international port were addressed.

In this thesis, we first apply traditional DRL directly to solve online container truck dispatching problems. Then, we combine DRL with hyper-heuristics (HH) to create a new framework, namely a DRL-based hyper-heuristic (DRL-HH). Compared to traditional DRL, the proposed DRL-HH offers the advantage of reusing existing well-performing rules and experience in the form of low-level heuristics, while also enhancing the interpretability of solutions and exhibiting improved convergence properties.

The final research work of this thesis addressed a more challenging multi-scenario online container truck routing problem. In a multi-scenario

problem, the problem uncertainties (service time of trucks) switch between different distributions in different problem-solving stages. A new framework, called DRL-GA-HH, that combines DRL-HH with GA is then proposed. It takes advantage of DRL's ability to recognise specific environmental patterns so that priori plans obtained by GA in some more stable scenarios (i.e. semi-deterministic scenarios) can be selected to improve the performance of the algorithm. The new method performs better overall compared to the DRL-HH framework. Some effective tricks, such as GA with variable visions and GA with surrogate networks, are developed during the research. They can help GA perform better in an online environment and can also be useful for other meta-heuristics when interacting with DRL in online environments.

Acknowledgements

This long journey for pursuing a doctoral degree would not have been possible without the enormous support and guidance I have received from many generous people.

First, I would like to express my sincerest gratitude to my supervisor, Prof. Ruibin Bai, for his invaluable support, inspiration, and guidance during my research. He shows me the door to online combinatorial optimisation and algorithms research. His enthusiasm and creativity in applying research to solve real-life problems greatly stimulated my interest in research. He taught me how to find an interesting problem, how to define, model it, and how to break down a complex problem into components and then attack each step by step. I have been extremely lucky to have the chance to work under his supervision.

Many thanks go to Prof. Rong Qu and Prof. Uwe Aickelin as my co-supervisors. Both of them have been very generous with their time, feedback and advice. In particular, I thank Prof. Rong Qu for her help and guidance when I studied on the UK campus in 2020. She gave me several detailed guidance and useful advice for the third sub-study of this thesis. As it was during the COVID-19 pandemic lockdown, she was also constantly concerned about my mental state and often encouraged me.

Thanks to Dr. Jianjun Chen for his help during the research. My research is based on his research. The simulator he developed for the port environment saved me the trouble of redeveloping it and saved me a lot of time. I would also like to thank many Artificial Intelligence and Optimisation (AIOP) members, including Xia Li, Tianxiang Cui, Xiang Li, Nick Dong, Chaofan Tu, Menglin, Cui, Jiandong Liu, Xinan Chen and Jiahuan

Jin. They all helped me a lot during my research.

I sincerely thank the International Doctoral Innovation Centre (IDIC) for offering me a fully-funded PhD scholarship. I also thank China Scholarship Council (CSC) for financial support for my study abroad.

I would thank my close friends Chaofan Tu, Xiangzhi Zhang and Xin Wang for walking with me through the difficult and fulfilling years of my life.

I wish to thank my mother, Lijun Ding, my father, Shuancheng Zhang, for being extremely patient and supportive, and my brother, Yuxuan Zhang, for being so kind and energetic. Without their sincere support, I would not be able to make this journey.

List of Publications

1. **Y. Zhang**, R. Bai, and R. Qu, "A deep reinforcement learning based hyper-heuristic for container terminal truck routing problem under uncertain service times," in *The 9th Multi-disciplinary International Scheduling Conference: Theory and Applications (MISTA2019), 12-15 December 2019, Ningbo, China., 2019.* [Chapter 4]

2. **Y. Zhang**, R. Bai, R. Qu, C. Tu and J. Jin. "A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties." *European Journal of Operational Research 300.2 (2022)*: 418-427. [Chapter 5]

3. C. Tu, R. Bai, U. Aickelin, **Y. Zhang** and H. Du. "A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems." Expert Systems with Applications (2023): 120568.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Scopes

In recent years, the development of information technology, such as the Internet, the Internet of Things, big data and cloud computing, has greatly contributed to the development of artificial intelligence (AI). The information industry and Internet-related companies are rushing to enter the field of AI. Internet giants such as Apple, Google, Microsoft, Amazon and Facebook, as well as domestic Internet companies in China such as Baidu, Tencent, Alibaba, Huawei and iFLYTEK, have all developed AI-related plans and launched research in big data platforms, autonomous driving, smart chips, city brains, smart healthcare, smart logistics, smart security and other application ecologies. AI has become one of the three cutting-edge technologies of the 21st century, alongside genetic engineering and nanotechnology (Eden et al., 2012). Many developed countries regard AI as an important strategy to maintain national security and enhance competitiveness. They have all developed long-term strategies for the development of artificial intelligence, such as the 'National Artificial Intelligence (AI) Strategy' of the UK and the 'National Artificial Intelligence Research and Development Strategic Plan' of the USA. In 2017, China also incorporated

the development of AI into its national strategic plan and issued the 'Development Plan for a New Generation of Artificial Intelligence', which sets out the overall deployment and key tasks for the development of AI. AI has already become an important tool to promote the development of a country's economy, military, people's livelihood, science and technology in the coming decades.

Meanwhile, with the further development of maritime trade, the number of container terminals is growing and the competition between them is becoming increasingly fierce. In order to gain greater viability, terminals need to utilise the latest information technologies, such as AI technologies, and intelligent optimisation methods to manage their operations effectively and efficiently. Due to this practical need, research in container terminal operations has greatly attracted the attention of academia and industry.

As demonstrated in Figure 1.1, a marine container terminal can be categorised into three major components: the berth area, yard area, and entry–exit zone (in this particular instance, it embodies five berths). Quay cranes (QCs) with a fixed quantity are stationed along the berth area to load and unload containers on and off the vessels. A solitary rail is erected parallel to the berth line for QCs to shift between diverse berths; however, they must not cross each other's path. The yard area serves as a provisional area for the containers and is often separated into yard blocks of the same or similar dimensions. Each yard block has an exclusive identifier (for instance, 55, 5H situated at the center of the yard area) and entails one or two yard cranes (YCs) to carry out loading and unloading tasks. Since QCs and YCs are only capable of moving unit-sized containers at one time, queues may crop up at any operating point. Typically, berths are located in deep seawater areas, linking to yards with a small thoroughfare network comprising bridges over shallow water regions, road sections, and intersections.

To transport containers between QCs and YCs through this road network, trucks (specifically yard trucks or internal trucks) are utilized. Finally, the entry-exit area encompasses gates regulating the arrival of external trucks.

Figure 1.2 is a photo of a real-world marine container terminal. Combining this figure with Figure 1.1, one can have a more intuitive understanding of the environment of the port and the scheduling optimisation problems involved in ports.

Basically, in container terminals, quayside operations generally include berth allocation problems, and QC scheduling problems. Berths and QCs are the main scarce resources of container terminals, and their scheduling is a critical aspect affecting the whole terminal operation. The berth scheduling problem refers to the formulation of berth plans for ships arriving in port during the planning cycle, including the berthing location, berthing time and departure time of the ships. The QC scheduling problems refer to the allocation of a QC to each vessel to complete the loading and unloading tasks.

While the yard-side operations include YC scheduling and yard truck (YT) dispatching problems. The YC scheduling deals with the efficient allocation of YCs to handle the loading and unloading of containers in the yard area. The primary objective is to improve the efficiency of container storage and pick-up, which in turn reduces the overall operational costs and improves the terminal's productivity.

The YT dispatching refers to the efficient allocation and scheduling of YTs to transport containers between the QCs and YCs. The primary goal is to minimise the total travel time, waiting time, or operational costs while ensuring timely and efficient container movements. Unlike the optimisation efforts for QCs and YCs, the enhancement of truck dispatching is essential not only for augmenting equipment efficiency within the berth or yard zones

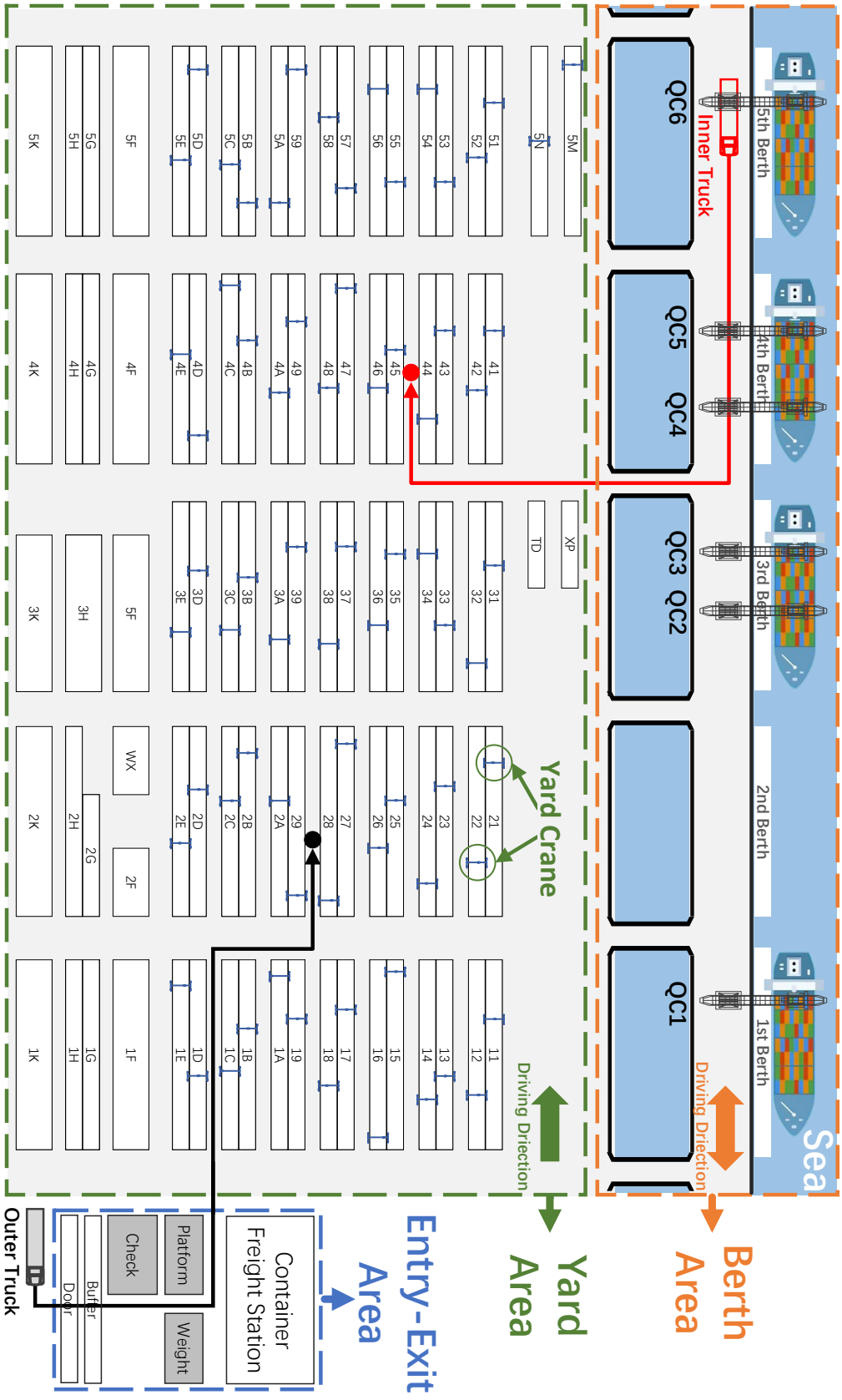

Figure 1.1: *A bird-view of a marine container terminal (Ningbo Meishan port)*

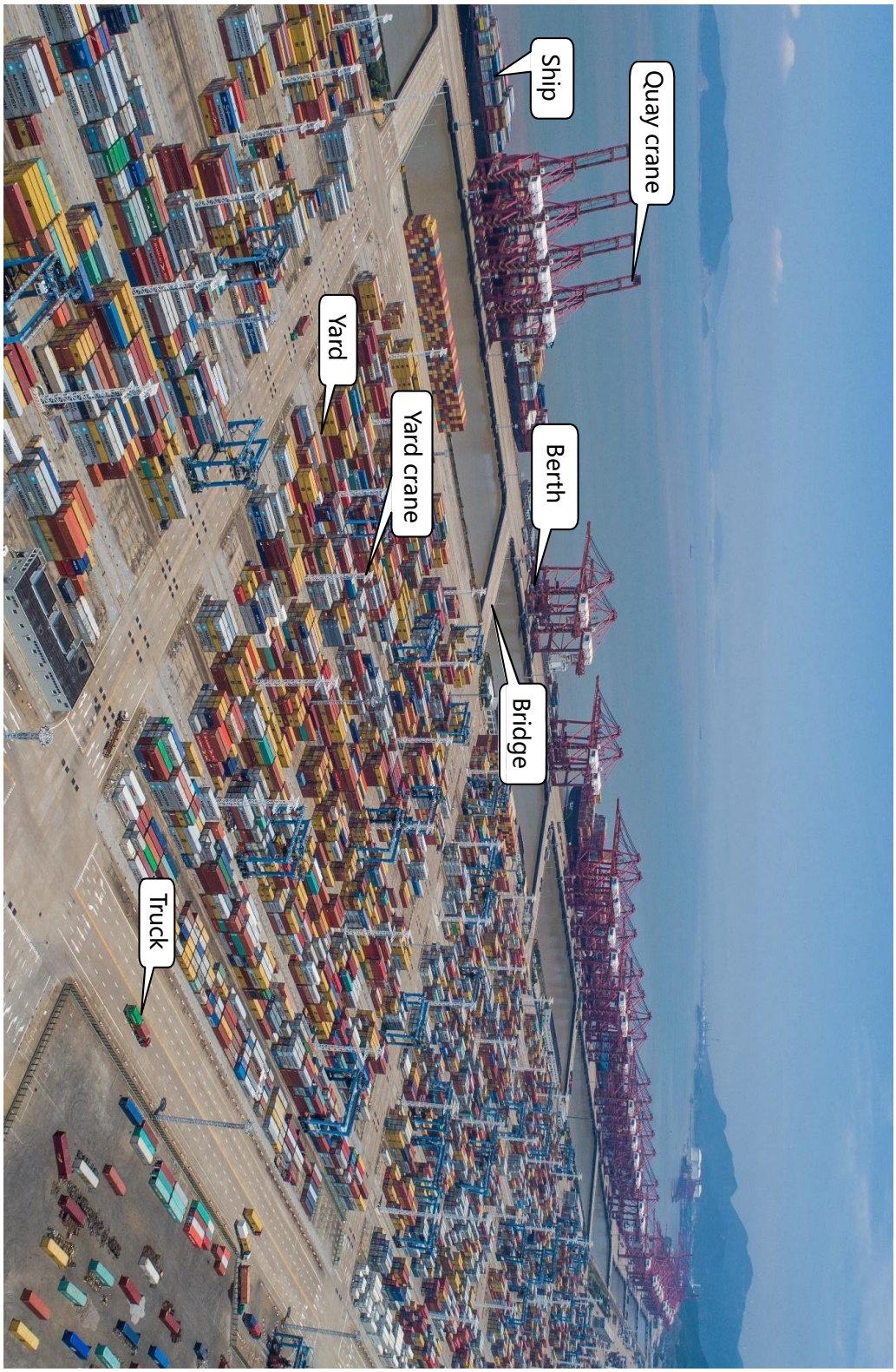

Figure 1.2: *A photo of a real-world marine container terminal*

but also for elevating the overall port performance. This arises from the fact that nearly all other equipment within the container terminal must engage with YTs. However, because of the interrelations between sea-side and yard-side operations, all the schedulable equipment including QCs, YCs and YTs need to be considered in YT dispatching. This renders the problem highly challenging, primarily due to three underlying reasons: Firstly, the scale of the problem is often extensive, given the vast number of containers to be handled, as well as the significant involvement of QCs, YCs and YTs. Secondly, the problem is highly non-linear, as all equipment (QCs, YCs, and YTs) can only handle unit-sized containers each time, leading to queues and inevitable waiting times. Lastly, uncertainties at various stages of operations, such as processing times by QCs, YCs, and trucks, are common, and predefined plans based on deterministic models are likely to fail.

In general, due to its direct relevance to enhancing the overall efficiency of port terminals and its highly challenging nature, the YT dispatching problem holds significant academic and practical value, thus constituting the rationale for choosing it as the research subject of this thesis.

Deep Reinforcement Learning The idea of reinforcement learning (RL) has its roots in the study of behavioural psychology. In 1927, Thorndike introduced the Law of Effect (Thorndike, 1927): Given a situation, if an animal behaves in a way that makes it feel comfortable, this behaviour will be associated with the situation more strongly. When the situation is repeated, the animal is also more likely to adopt the behaviour again. On the contrary, if the animal takes a behaviour that is uncomfortable, it is less likely to take the behaviour again if the situation is repeated. In other words, the mapping between a behaviour and a situation depends on its utility in that situation. RL is precisely this kind of learning that mimics

the evolutionary process of organisms to adapt to their environment. In general, RL is characterised by two features: firstly, it is not based on static labeled data, but rather on active action to test the environment. Secondly, the feedback obtained from the action is not a clear-cut right or wrong, but rather a vague evaluation. In essence, RL learns how to map from the environment to the optimal action. RL acquires knowledge from interactions in the environment and the corresponding evaluations, and improves action strategies to adapt to the environment in order to achieve the desired goal (Thrun & Littman, 2000).

Developments in reinforcement learning include trial and error learning (Bellman, 1966), dynamic programming and optimal control (Bertsekas, 2007), and temporal difference learning (Seymour et al., 2004). In the 1980s, after their respective developments, they converged to form the theoretical framework of RL. After combining OR and control theory, RL made theoretical and algorithmic breakthroughs and was applied to sequential decision problems such as automated control and optimal decision making (Azhikodan et al., 2019). RL usually uses trial-and-error learning mechanisms. Since rewards are often delayed, classical RL can only improve its own action strategies based on the sparse payoffs in learning, and the exploration mechanism is mainly based on random trial-and-error, lacking an effective strategy selection method. Therefore, in classical RL, "infinite" and "sufficient" access to the state-action trajectory is required to traverse all the states to converge. In more complex problems, the number of state-action pairs is so large that a dimensional catastrophe can occur. Currently, the mainstream approach to solving the Curse of Dimensionality is to abstract and generalize the state and action spaces.

Recently, RL has been further developed by combining it with deep learning (DL). Deep learning has strong perceptual capabilities but lacks

decision planning capabilities, while RL has decision-making capabilities but lacks direct perception of information in complex environments. In 2015, Mnih et al. (2015) of Deep Mind combined convolutional neural networks (CNN) with Q-learning, successfully pioneering the combination of DL and RL, which is DRL.

Researchers in the machine learning community have recently attempted to tackle combinatorial optimisation problems using only machine learning methods (i.e., without explicitly exploiting the structures of the mathematical models). Although these methods have made some progress, they suffer from some problems, such as the lack of generalisation across different scenarios, inefficiency in data use, and the inability to discover insights and interpret solution structures. There seems to be very little interaction between the machine learning and OR communities. Combining rapidly evolving machine learning algorithms with the long-established and effective algorithms from the OR community would leverage their respective strengths to achieve better algorithmic performance. Due to its adeptness in solving sequential decision problems, DRL was noticed by researchers in the OR community.

1.2 Motivations and Research Questions

For real-world combinatorial optimisation problems, such as the container truck dispatching problem, it is not always possible to acquire accurate information of all the problem characteristics at the beginning. Instead, the real values of problem parameters are often sequentially revealed over time during decision making. This leads to an important branch of combinatorial optimisation problems, online combinatorial optimisation problems (with strong real-time constraints) or combinatorial optimisation problems with uncertainties (with relatively weak real-time constraints). Both suffer

some uncertainties and are subject to varying degrees of real-time requirements.

In this type of problems, solutions generated as a priori often encounter various issues such as inferior service quality, increased costs, and infeasible solutions, all of which would lead to substantial losses. This can be illustrated by a small test instance of the online container truck routing problem defined in Section 2.2. In this small test instance, 80 truckload transportation tasks were given between 2 yard cranes and 2 quay cranes, with up to 20 trucks available to complete these tasks. In reality, the crane service times cannot be accurately predicted in advance due to practical factors (see Section 2.2 for details). In this test example, as depicted in Table 1.1, the crane service times were drawn from a uniform distribution of [80, 120] seconds. That is, the service times had a maximum error of 20% from their mean values of 100 seconds per task. If the mean value of 100 seconds is used as inputs (Since the true values are not known in advance, their mean is the best estimate of their true values.), the best objective value (makespan) that could be obtained using several metaheuristic methods was 5588 seconds. However, when the obtained best routing plan was executed and the crane service times revealed their true values in an online fashion, the actual objective value was 11085 seconds, representing a 98.4% performance drop.

Table 1.1: An example of the performance deterioration of a deterministic solution for problems with uncertainties. The results are based on a small container truck routing test instance with 4 cranes, 20 trucks and 80 tasks. Crane service times were uniformly drawn from [80,100].

	determ. method (expected obj.)	**determ. method** (actual obj)
obj.	5588	11085
performance loss	n.a.	98.4%

Also, in the Uncertain Capacitated Arc Routing Problem studied in Mei et al. (2010), offline methods have shown to generate infeasible solutions when the actual demand of a task exceeds the remaining capacity of the vehicle. As a result, the vehicle cannot fully serve the task as expected by the offline methods.

Therefore, developing other approaches that can accommodate uncertainties and make decisions that can adequately balance the conflicts between solution optimality and its resistance to unpredictable (and sometimes disruptive) changes is critical. DRL has achieved great success in sequential decision-making problems. Solving combinatorial optimisation problems with uncertainties using DRL has several advantages. Firstly, although the training process is done offline, the trained model can be applied online, thereby better meeting real-time requirements. Secondly, it is able to adapt the selected actions to the changes in the environment so that it can accommodate the uncertainties of the parameters in the problems. Thirdly, compared with general heuristics that adopt some rules to construct solutions efficiently but without a global optimisation perspective, DRL's learning mechanism allows it to learn to sacrifice short-term returns for greater long-term returns. Therefore, DRL is an excellent approach to solving combinatorial optimisation problems with uncertainties. This thesis is concerned with the design of DRL-based hyper-heuristics that can solve real-world combinatorial optimisation problems with uncertainties in ports.

In real production environments, decisions made by DRL based on deep neural networks are often difficult to understand by humans. This is a significant barrier to the application of DRL-based algorithms in industry, especially in industries with high security requirements. To make such algorithms more acceptable to industry, we consider designing DRL as a hyper-heuristic with a set of human-understandable rules serving as the

low-level heuristics (LLHs). This framework can increase the interpretability of the algorithm and facilitate its diffusion in industry. Additionally, the use of the LLHs reduces the number of actions and hence the size of the search space. As a result, the training can converge quicker than traditional DRL methods.

Furthermore, the uncertainties of parameters in real-world combinatorial optimisation problems can be very high. Sometimes, these important parameters change drastically, while other times the changes are small. This makes it hard to solve them with a single trained neural network. In this situation, we can classify the variation of these parameters and define each category as a scenario. If there are appropriate methods that can intelligently recognise different scenarios and make proper decisions accordingly, the overall quality of a solution will improve. Some meta-heuristics based on deterministic models perform well when the parameters are relatively stable, while DRL performs better when the parameters change drastically. Therefore, it is necessary to design a unified framework that can choose between DRL and meta-heuristics based on the magnitude of the environment changes, so that the value of the objective function is optimised.

1.3 Contributions

The contributions of this thesis are summarised as follows.

- Although other researchers have studied applying DRL to solve combinatorial optimisation problems, most of the problems that they solved have no uncertainties. This thesis attempted to apply DRL to challenging online container truck dispatching problems (See Chapter 3 for more details).

- In order to enhance the interpretability of black-box-style DRL meth-

ods and help industry practitioners better understand the high-level behavior of DRL agents, this thesis introduced DRL into a constructive hyper-heuristics framework. The proposed framework not only enhances the interpretability of traditional DRL methods but also increases the speed of convergence of the training of DRL agents (See Chapter 4).

- Some real-world combinatorial optimisation problems exhibit a greater degree of uncertainty. Changes in some important parameters of the problem are sometimes dramatic, but in other cases, they are very minor. As a rule of thumb, meta-heuristic algorithms, such as GA, can produce solutions of higher quality when the parameters are relatively stable, while DRL has good adaptability when the parameters vary drastically. We combine the advantages of both methods and design a DRL and GA-based hyper-heuristic (DRL-GA-HH). This framework is capable of intelligently recognising different scenarios and making proper decisions in those scenarios (See Chapter 5).

- To solve the issue of long training time by the DRL-GA-HH framework, two mechanisms were introduced in this thesis. The first is a variable length vision for GA, and the second is the introduction of a surrogate network (SN) for the fitness assessment of individuals in GA. The new framework, denoted as DRL-GASN-HH, significantly reduces the training time (See Chapter 5).

1.4 Thesis Outline

This thesis is structured as follows:

- In Chapter 2, preliminary is introduced, including the introduction to the solved problems and the related important algorithms or methods

that were developed to solve them. Literature is also reviewed in this chapter.

- Chapters 3, 4, and 5 are the contribution chapters which have been introduced in Section 1.3.

- Chapter 6 draws conclusions from the research findings in this thesis, as well as directions for future studies.

Chapter 2

Preliminary and Literature Review

This chapter provides preliminaries for the research described in this thesis. Firstly, the combinatorial optimisation problems (COPs) and the related important concepts are introduced. Secondly, the online container truck dispatching problem solved in this thesis is introduced. Thirdly, literature about methods to solve COPs in ports is reviewed. After that, some important algorithms (techniques) involved in this thesis are introduced and relevant literature is reviewed, too.

2.1 Combinatorial Optimisation Problems

Combinatorial optimisation problems are a class of problems that exist everywhere in production and daily life, for example, the vehicle routing problem (VRP) which aims to minimise the transport costs and/or time in freight logistics, the timetabling problem which organises courses or examinations in schools so that all students and teachers can work and study with their most preferred patterns, the job scheduling problem widely faced in manufacturing, and so on.

Combinatorial optimisation also has applications in information technology, industrial engineering, communication networks and economic management, which have a significant impact on practical production and life. Some of the more widely studied COPs include graph coloring problems (GCP) (Hao & Liu, 2017), project scheduling problems (PSP) (Herroelen et al., 1998), educational timetabling problems such as exam timetabling problems (ETTP) and high school timetabling problems (HSTP), knapsack problems, packing problems, traveling salesman problems (TSP) (Papadimitriou & Steiglitz, 1998), set covering, facility location problems, etc.

The objective of a COP is to find the optimal element of some set of discrete items. The mathematical model of the COP for minimisation is as follows.

$$\begin{aligned} & \min f(x) \\ s.t. \quad & g(x) \geq 0 \\ & x \in \Omega \end{aligned} \tag{2.1}$$

where x is the vector of the discrete decision variables, Ω denotes the domain of the decision variables of the problem, and $f(x)$ and $g(x)$ are the objective and constraint functions, respectively. Thus, a COP can usually be represented by a tuple (Ω, F, f), where $F = \{x \mid x \in \Omega, g(x) \geq 0\}$ denotes the feasible domain and any $x \in F$ is defined as a feasible solution to the problem. The objective of solving a COP is to find $x^* \in F$ satisfying $\forall x \in F, f(x^*) \leq f(x)$. Such an x^* is called the optimal solution and its corresponding objective function value $f(x^*)$ is called the optimal objective value.

2.1.1 VRP and Some Basic Concepts

Since VRP is a classic COP and has some similarities to the yard truck (YT) dispatching problem addressed in this thesis, this section first introduces some of the basic concepts involved in the COP based on VRP. Finally, by clarifying these concepts, the category to which the problem addressed in this thesis belongs is identified.

The Vehicle Routing Problem (VRP) was first proposed in 1959. It refers to dispatching a fleet of vehicles that originally stayed at a depot to serve a set of customers with different freight demands. It is generally defined on a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$, where $\mathcal{V} = \{v_0, ..., v_n\}$ is the set of vertices; $\mathcal{E} = \{(v_i, v_j) | (v_i, v_j) \in \mathcal{V}^2, i \neq j\}$ the arc set; and $\mathcal{C} = (c_{ij})_{(v_i,v_j)\in\mathcal{E}}$ a cost matrix defined over $\mathcal{E}$, representing distances, travel times, or travel costs. Vertex v_0 is usually called the depot, while the remaining vertices in $\mathcal{V}$ represent customers (or requests) that need to be serviced. The objective is usually to minimise the total travel distance, cost or time on the condition that all customer's needs are met under certain constraints (Dantzig & Ramser, 1959).

2.1.1.1 Dynamic VS. Static and Deterministic VS. Stochastic

According to Psaraftis (1980), VRPs have two dimensions: *evolution* and *quality* of information. The evolution of information refers to changes in the available information during route execution (dynamic or static). Quality of information refers to the possibility of having *uncertainty* in the available information (deterministic or stochastic). For instance, when the demand of a customer is only known as a range estimate of its real demand. Based on these two dimensions, four categories of routing problems are identified: static and deterministic, static and stochastic, dynamic and deterministic, dynamic and stochastic problems.

- In *static and deterministic* problems, all information on the problem components (input) is known in advance and the routes do not change during route execution.

- In *static and stochastic* problems, the problem input is partially known, and it is represented with random parameters, where the real values are known during the execution of the route. Additionally, the routes are designed before their execution, and then only minor changes are accepted.

- In *dynamic and deterministic* problems, partial or all information on the problem input is unknown, and it is only disclosed during the execution or design of the routes. No component of the problem is represented with random variables. For these problems, vehicle routes are redefined in an ongoing fashion with the real-time help of a central decision-maker. This type of problem is also referred to as *online* or *real time* by some researchers (Jaillet & Wagner, 2008a).

- In *dynamic and stochastic* problems, partial or all information on the problem input is unknown and only disclosed during the execution or design of the routes. Meanwhile, at least one input is represented with a random variable. Unlike the previous category, exploitable stochastic knowledge is available on the dynamically revealed information. *Dynamic and stochastic* problems can be seen as an extension of *dynamic and deterministic* problems, where additional stochastic knowledge is dynamically available. This allows for anticipating future events through probability distributions of random variables or historic information (Rios et al., 2021). As before, routes are redefined in an *online* fashion.

2.1.1.2 Online VS. Offline

An *online optimisation problem* is one in which decisions must be made in real-time, with incomplete information about future events. In other words, the decision-maker must make choices based on what they know at the time, without knowing what will happen in the future (Jaillet & Wagner, 2008b). An example of an online optimisation problem is real-time traffic routing, where a navigation app must determine the best route for a driver based on current traffic conditions. Another application is some real-time services, such as same-day pickup and delivery offered by FedEx and other courier companies. A third example pertains to a relatively nascent industry, namely Jet Taxis. A Jet Taxi is a diminutive aircraft that caters to regional airports, whereby customers can solicit real-time transportation services from one airport to another.

On the other hand, an *offline optimisation problem* is one in which all information is known in advance and decisions can be made ahead of time. This allows for more comprehensive analysis and optimisation of solutions. An example of an offline optimisation problem is designing a delivery route for a fleet of trucks that will operate over a period of several days, under fixed demand and no uncertainty.

2.1.2 Occurrence of COPs with Uncertainties

In general, different research strands can be observed in the scientific literature when considering COP with uncertainties. In the operations research (OR) community, COP with uncertainties is considered for different problem formulations, e.g. the vehicle routing problem (VRP) (Ritzinger et al., 2016), the container relocation problem (Zehendner et al., 2017) or the scheduling problem (Ouelhadj & Petrovic, 2009). Concerning scheduling problems, another strand, appearing in the chemical industry optimisation

area can be found, where a different online scheduling procedure is proposed (Gupta & Maravelias, 2016).

In the evolutionary computation (EC) community, COP with uncertainties has also been considered extensively. In recent years, one of the popularly studied methods in this community is genetic programming hyper-heuristic (GPHH). It has achieved successes in solving several uncertain problems, such as project scheduling (Park et al., 2018; Zhu et al., 2021), combinatorial bi-level optimisation (Kieffer et al., 2019; Song & Lin, 2021), and resource allocation (Tan et al., 2018, 2020).

2.2 Online Container Truck Dispatching

The first real-world problem considered in this thesis is a yard container truck dispatching problem faced by one of the largest international ports. At the same time, it is also a type of problem faced by many maritime ports, airports and logistics centres. The related works are reviewed in Section 2.3.2. The problem is concerned with the optimal truck assignments for a list of predefined container transportation between the vessels (seaside) and the container yard in a container terminal (see the typical layout in Figure 1.1). On each day, the terminal is visited by several vessels with a list of predefined containers to be loaded and/or unloaded. Cranes are required to handle the operations at both the seaside (QCs) and the yard area (YCs). The yard area consists of a number of yard blocks, each with a unique yard block ID (e.g., 51-5K in Figure 1.1), and is equipped with a single yard crane. A fleet of homogeneous trucks, based at the depot initially, transports containers between QCs and yard blocks. In this problem, the depot, each QC and each yard block is represented as a node.

As the most valuable resources in a container terminal, QCs are often considered as the primary focuses in operations optimisation. Therefore,

in this study, the objective is set to be the minimisation of the total QC waiting time between two consecutive operations, which is mostly caused by late truck arrivals.

Different from other container truck dispatching problem studies, the problem considered in this study is modeled as an online problem to realistically formulate the uncertainties of the crane operation times (i.e. loading/unloading time) caused by the complexities in container stacking requirements, operator proficiency, weather conditions and differences among cranes. Meanwhile, since each crane can only handle one operation at any time, it is extremely challenging, if ever possible, to deal with the truck queues at both QCs and YCs with deterministic problem formulation.

Each time a vessel arrives, high-level decisions are made in terms of the assignments of the berth, the QCs and the yard blocks for this vessel. For practicality, these decisions are made separately from the truck dispatching problems concerned in this thesis.

Containers are either in small size (20-inch) or in large size (40-inch). We use *task* to define a standard operation unit consisting of either two small containers of the same Source-Destination pairs or one large container. Each task is then defined by a source node (SN), a destination node (DN) and the details of the corresponding container(s). Both SN and DN can be a QC deployed at the seaside or a YC which is close to a small yard area in the yard. Picking up containers is associated with loading time, namely source node service time (SST); While unloading the container leads to an unloading time, namely destination node service time (DST). The transportation time is the travel time from the source node to the destination node. Table 2.1 shows an example of tasks from real world. A node starting with 'CR' indicates that the node is a QC, and other nodes, such as '$5M$', '46' or '53' indicate a yard block (or a YC). It should be

noted that SST and DST cannot be acquired in advance. They can only be obtained after the loading or unloading operation at the working node is completed. Since our method is to train a model at first, then use the trained model to make decisions. So SST and DST are known in the training process, but they are unknown in the test sets. A task is serviced by one truck exactly.

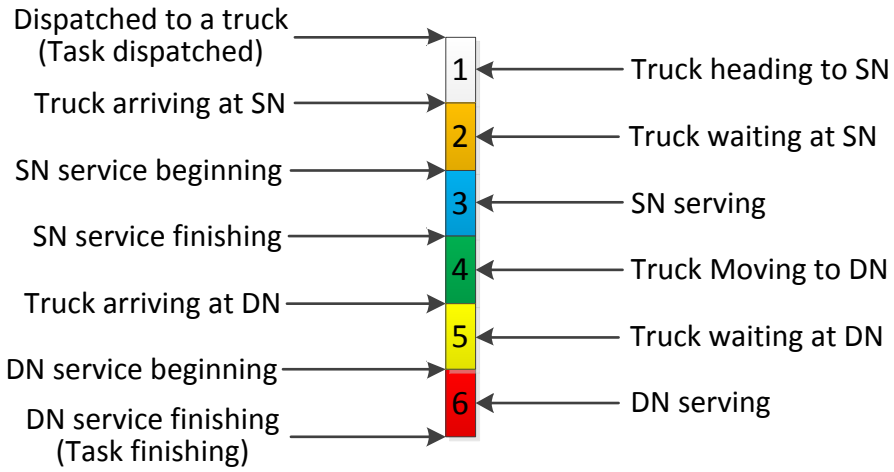

Figure 2.1: *The flow of handling a task.*

The detailed events of handling a task are shown in Figure 2.1. The timings of these events for all tasks define a full truck dispatching solution.

In this problem, the SN and DN of the tasks to be dispatched within a task set are known in advance. Additionally, for each assigned QC, a load balance planner is used to generate a *work queue* that specifies the operation sequence of the containers to be loaded and unloaded from this vessel. Since every task involves a QC, for example, the one for the first task in Table 2.1 is $CR13$. So, for a given task set, at first, tasks are grouped by the QCs to which they belong, with each group being called the *work queue* mentioned above. As shown in Figure 2.2, the task dataset shown in Table 2.1 can be divided into four *work queues*, where the number after 'T' indicates the sequence number of the task. For example, 'T6' means task No. 6. Again, practical rules require that each QC is responsible for one type of operation

only (i.e. either loading or unloading but not both). For example, the QC $CR13$ in Table 2.1, only performs the unloading tasks. The QCs strictly follow the sequences defined by the work queues while operations at each yard block (YC) adopt the First-Come-First-Serve (FCFS) policy.

Table 2.1: An example of tasks

NO.	SOURCE NODE	DESTINATION NODE	SOURCE NODE SERVICE TIME (s)	DESTINATION NODE SERVICE TIME (s)
0	CR13	52	141	118
1	CR07	45	124	111
2	CR07	46	131	127
3	CR07	47	146	114
4	CR13	53	142	103
5	CR07	48	108	103
6	CR13	54	131	128
7	CR13	43	118	118
8	CR07	47	118	118
9	CR13	44	124	132
10	CR13	53	121	106
11	CR13	54	137	111
12	2B	CR08	129	115
13	2C	CR08	121	121
14	29	CR08	125	119
15	1A	CR09	124	116
16	1B	CR09	137	119
17	1C	CR09	117	118
18	1F	CR09	123	111
19	2A	CR08	109	110
20	1G	CR09	127	123
21	2E	CR08	130	109
22	2E	CR08	111	106
23	1H	CR09	147	116

2.2.1 Online Dispatching Patterns

There are two widely used online dispatching patterns in port truck dispatching systems. One is *task-oriented dispatching*, and the other is *truck-oriented dispatching*. Task-oriented dispatching means that a new

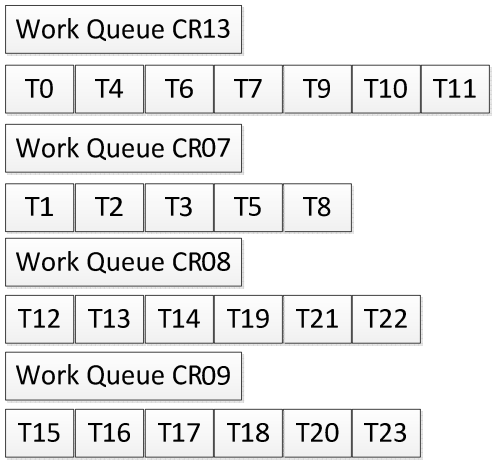

Figure 2.2: *Grouped work queues*

dispatching request should be invoked every time a new task (a movement of an unassigned container) emerges. The disadvantage of this approach is that it reduces the utilisation of the container trucks, and it can cause traffic congestion when the container trucks wait for tasks in the yard. Truck-oriented dispatching means that a new dispatching request should be invoked each time a truck is idle (e.g. a new truck has just joined the system or a truck has just completed its current task). In this modeling, *truck-oriented dispatching* is applied.

2.2.2 Scheduling Mode

The scheduling mode in the container terminal refers to the general arrangement of the port resources (including QCs, YCs and container trucks), mainly including 'line mode' (LM) and 'plane mode' (PM).

The LM, also known as static mode, pairs each YT with a particular QC. It allows a particular YT to serve only a particular QC during transport until all tasks assigned to that YT by that QC have been finished. The diagram of LM is shown in Figure 2.3. The YT is driven empty to the paired QC during an unloading task and waits under the QC. When the QC loads a container to the YT, it is driven to the import area in the yard

with a heavy load. After unloading the container, the truck continues to return empty to the QC to get another container and then moves it to the yard, too. Similarly, loading a container is the reverse of unloading one. In most non-automated real ports, this approach is still the dominant one (Bose et al., 2000). However, in LM, the YTs drive heavy only half of the routes, which will reduce the utilisation of the YTs and increase the operating costs of the container port. Besides, LM also causes long waiting times due to the improper allocation of YTs by the operator. For example, when the operator allocates a small number of YTs, QCs and YCs will have to wait for some time after YTs have been loaded and unloaded, whereas ideally, the loading and unloading operation of all QCs and YCs should be uninterrupted. Conversely, when too many YTs are allocated, it will cause YTs to queue under QCs and YCs waiting to be loaded and unloaded, which is a waste of port resources and will lead to traffic congestion, significantly reducing the overall efficiency of the ports.

The difference between LM and PM is that the YTs no longer serve a specific QC, but instead, all QCs share all YTs according to the actual situation on site. In PM, when a dispatched task is finished, a YT does not need to drive back empty to get the next task. Instead, the central dispatcher will dynamically dispatch the YT according to an optimisation algorithm or some rules (for example, shortest travel distance) based on its current position and some other features, as shown in Figure 2.4. This mode reduces the empty travel distance of the YTs and makes full use of the loading and unloading capacity of the QCs and YCs. Besides, it reduces traffic congestion caused by queuing of YTs during the waiting process. Therefore, PM is essential for improving the overall competitiveness of a port (Liang et al., 2014) and it is applied in the studies in this thesis.

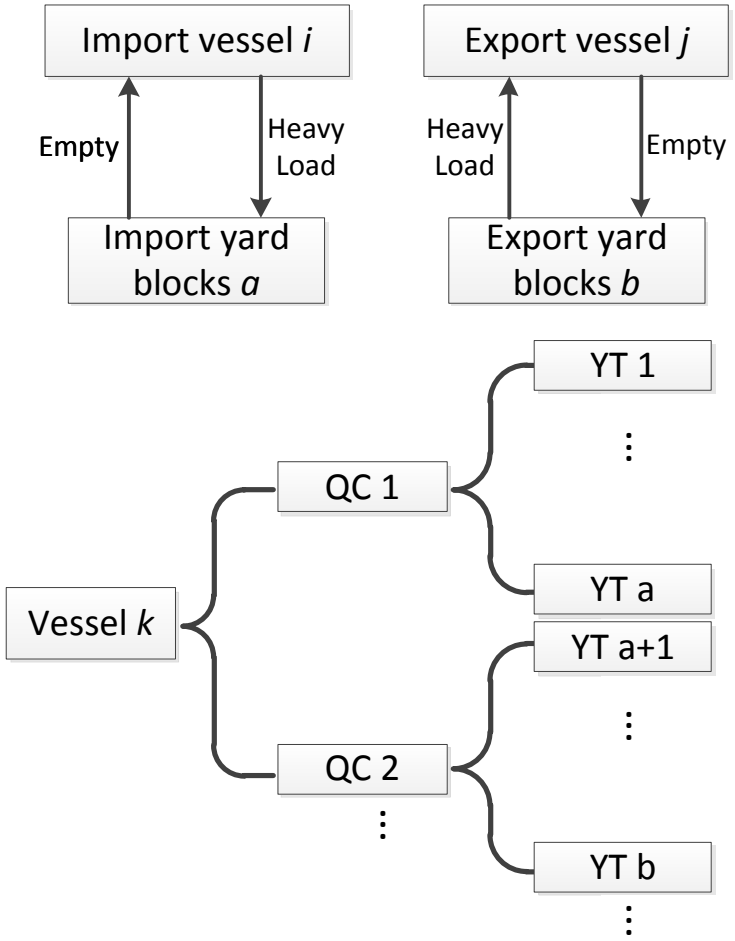

Figure 2.3: *A schematic diagram of the line mode*

2.2.3 Problem Formulation

This problem is initially formulated in Chen et al. (2016). Here, we provide a slightly different formulation. The problem can be defined with a directed graph $G = (N, A)$, where vertices $N = \{0\} \bigcup N^{ship} \bigcup N^{yard}$ are the union of the depot (node 0), QCs N^{ship}, and YCs N^{yard}. Set A denotes the arcs between the nodes. Let WQ_l denote the work queue list associated with QC $l \in N^{ship}$ and n_l be the size of WQ_l. Denote q_l^h be the h-th task in work queue WQ_l. For each task i from some work queue, denote $t(i)^{ship}$ (respectively $tt(i)$, $t(i)^{yard}$) be its operation time at the QC (respectively its transportation time, and operation time at the yard crane). Denote $td(i, j)$ be the deadheading time from the destination of task i to source of task j

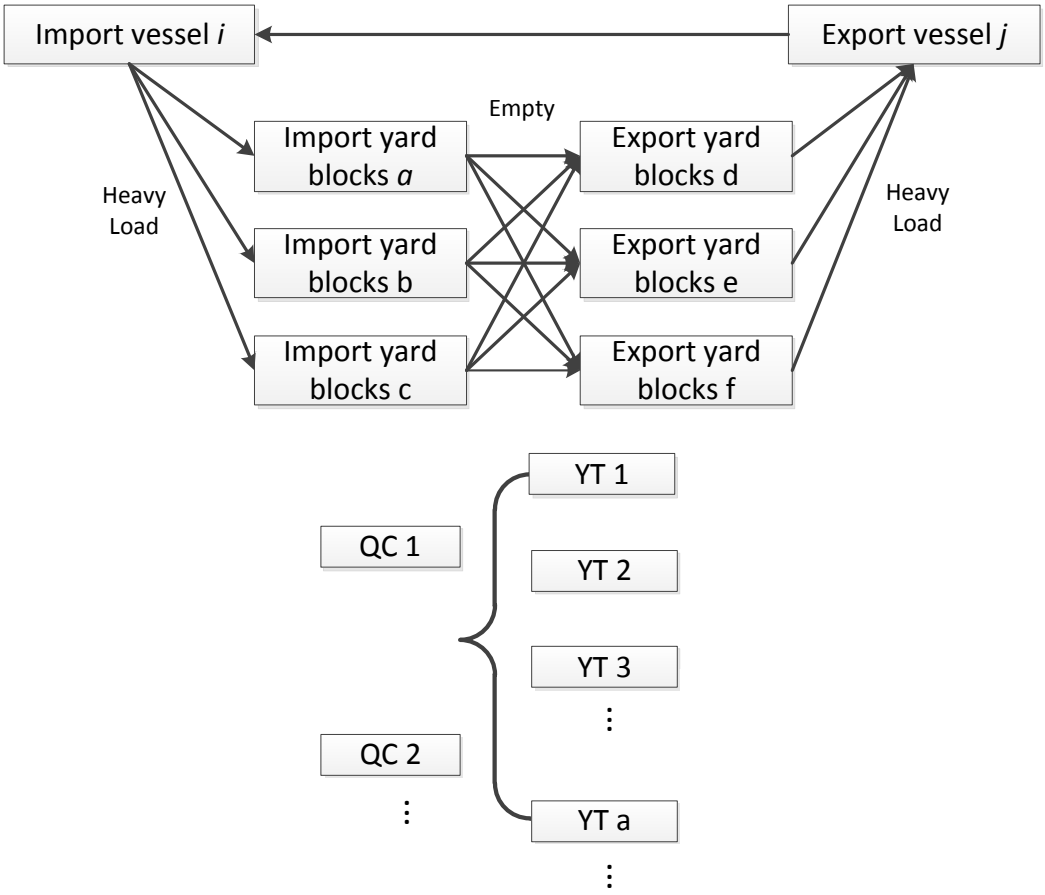

Figure 2.4: *A schematic diagram of the plane mode*

when task j is serviced immediately after task i by the same truck. Denote K be the set of homogeneous trucks to be dispatched. Let $Q = \bigcup\{WQ_l\}$ be the set of all tasks in all work queues.

2.2.3.1 Decision Variables

There are two sets of decision variables. The first set is the truck assignment decisions $x(i,j)^k$, $\forall i,j \in Q, k \in K$, which takes value 1 if task j is immediately serviced after task i by truck k, and 0 otherwise. The second variable set determines the operation start times at the QCs and YCs (respectively denoted as $T(i)^{ship}$, $T(i)^{yard}$) for each task $i \in Q$. For the ease of mathematical formulation, we also use $T(i)^{SN}$ and $T(i)^{DN}$ to denote the operation start time of task i at the source and destination

nodes, respectively. Similarly, we use $t(i)^{SN}$ and $t(i)^{DN}$ to stand for the service times for task i at the source and destination nodes, respectively.

2.2.3.2 Objective Function

The objective of the problem is to minimise the aggregated QC waiting times between two consecutive tasks, which equals to the time difference between the operation start time of the current task and the completion time of the previous task, expressed as shown in (2.2).

$$\min \sum_{l \in N^{ship}} \sum_{h=1}^{n_l - 1} [T(q_l^{h+1})^{ship} - T(q_l^h)^{ship} - t(q_l^h)^{ship}] \tag{2.2}$$

2.2.3.3 Constraints

The following constraints need to be satisfied to ensure the feasibility of the solution. Constraint (2.3) ensure that all tasks are serviced exactly in the order specified by the work queues. Constraint (2.4) ensures the feasible operation start times for any two consecutive tasks assigned to the same truck. Constraints (2.5)-(2.6) make sure that each task is serviced by exactly one truck. Another important constraint is the FCFS policy at each yard crane which has an effect on a truck's waiting time before the assigned tasks can be started.

$$T(q_l^{h+1})^{ship} \geq T(q_l^h)^{ship} + t(q_l^h)^{ship}, \quad \forall q_l^h \in WQ_l, l \in N^{ship} \tag{2.3}$$

$$T(j)^{SN} \geq (T(i)^{DN} + t(i)^{DN} + td(i,j))x_(i,j)^k \quad \forall i, j \in Q, k \in K \tag{2.4}$$

$$\sum_{j \in Q} \sum_{k \in K} x(i,j)^k = 1 \quad \forall i \in Q \tag{2.5}$$

$$\sum_{i \in Q} \sum_{k \in K} x(i,j)^k = 1 \quad \forall j \in Q \tag{2.6}$$

2.2.3.4 Problem Characterisation

In the offline version of this problem, although the crane operation times ($t(i)^{ship}$, $t(i)^{yard}$) are random variables that obey a certain distribution, their true values are known before problem solving.

However, in the online problems solved in this thesis, the crane operation times ($t(i)^{ship}$, $t(i)^{yard}$) are subject to uncertainties and are assumed to be revealed in an *online* fashion. We model this online COP as a sequential decision problem (i.e., multi-stage). Once a decision is made, no change can be made later on to reverse the decision.

This problem falls under the category of a *dynamic* problem, primarily due to the fact that while the source nodes and destination nodes of all tasks are known in advance, the actual dispatching of a specific task occurs as a dynamic event. This is contingent upon the availability of idle vehicles, as a task can only be dispatched when such a vehicle becomes available. The appearance of an idle vehicle is intricately linked to the sequence of previously dispatched tasks and their respective service times, making it a dynamic occurrence.

Moreover, this problem can also be classified as a *stochastic* problem. This classification stems from the fact that even after the dispatcher selects a particular task, its precise service times at both the source node and destination node remain unknown. Instead, it follows a random value that adheres to a certain distribution. Only upon the task's final completion does its service time become evident.

In general, this problem can be characterised as an *online*, *dynamic*, and *stochastic* problem.

2.2.4 VRP VS. YT dispatching problem

The Vehicle Routing Problem (VRP) was first proposed in 1959. It refers to dispatching a fleet of vehicles that originally stay at a depot to serve a set of customers with different freight demands. The objective is usually to minimise the total travel distance, cost or time on the condition that all customers' needs are met under certain constraints (Dantzig & Ramser, 1959). The YT dispatching problem is similar to the classical VRP problem. However, there is a difference between them, as shown below:

- The objective of VRP is generally to minimise the total travel distance by the vehicles. While the objective of the YT dispatching problem is not directly measured by the total travel distance of all trucks because the time required to complete all tasks depends on the container truck consuming the longest time. That is the time when the last YT completes the last task. Therefore, the YT dispatching problem usually minimises the time required to complete all tasks.

- The VRP tries to find the sequence of tasks to be finished by a vehicle or a fleet of vehicles, while for the YT dispatching problem, the solution is the set of routes formed by multiple trucks travelling between the QCs and YCs during the operations.

- In VRP, a vehicle starts at one node. When completing all tasks allocated to it, it needs to return to its original point. However, in the YT dispatching problem, the truck starts from one node (a QC or YC) and does not have to return to the original point. It can park at any available depot, usually near the end of the last task performed.

- The VRP generally limits the maximum distance a vehicle can travel. The capacity of different vehicles varies, and the amount of goods

required by each customer is usually different. However, there is no limit to the maximum travel distance in the YT dispatching problem. The transport capacity of YTs is the same, and the goods transported are standard containers.

In summary, the YT dispatching problem is not the same as a traditional VRP problem. However, the solution methods to these two problems are somewhat similar due to some common characteristics. Methods to solve the COPs in ports are reviewed in the next section.

2.3 Methods for Combinatorial Optimisation Problems in Ports

There is a vast amount of research on port optimisation problems involving all stages of port operations. However, since our focus in this thesis is the YT dispatching and scheduling problems, the literature related to quayside operations is not reviewed in this section. We briefly introduce literature related to other yard side operations, then conduct a comprehensive and in-depth review of the solution methods to the YT dispatching problem. Several surveys on whole port operations can be found in the literature. Bierwirth & Meisel (2015); Carlo et al. (2015) focused on quayside operations, while Covic (2018) probed the yard operations, mainly including yard vehicle scheduling and YC scheduling.

2.3.1 Methods for Yard Crane Scheduling

In yard side operations, YC scheduling is a topic that has been mainly studied. In terms of the number of YCs involved, YC scheduling can be divided into the scheduling of a single YC and multiple YCs. Chung et al. (1988) were the first to investigate the YC scheduling problem, developing

a graphical simulation-based incentive model and comparing it with the Port of Portland to demonstrate the method's effectiveness. Narasimhan & Palekar (2002) explored the single YC scheduling problem and proved the NP-difficult feature of this problem. Then it was solved using the branch-and-bound (B&B) algorithm with the objective of minimising the total operation time of the YCs. Kim & Kim (2003) were the first to study the YC scheduling problem from the perspective of routing optimisation and developed a mixed-integer programming model with the objective of minimising the total operation time of the YCs, too. They designed a heuristic algorithm to compute the proper sequence of container tasks. Based on this study, Gharehgozli et al. (2014) improved the model and solved them using dynamic programming and a two-stage algorithm, respectively. However, these studies did not consider the YTs in the scheduling of YCs. Therefore, Ng & Mak (2005) considered the impact of the arrival of the YTs and assumed that the number of tasks and the predicted completion time are known. This type of study considers the tasks as several sub-tasks but does not classify the sub-tasks. In order to further optimise the scheduling, CHEN et al. (2016) divided the sub-tasks in the YC scheduling into different phases and developed a path optimisation model to minimise the distance travelled between the different phases. However, the model only considers the distance between phases, ignoring the distance travelled by the YCs within a phase, making it difficult to ensure a global optimal route for YCs.

Other studies have taken into account the actual operation of container terminals and explored the scheduling of multi-YCs for both crossable and non-crossable operations. Although multi-YC scheduling only adds the constraint of inter-YC conflicts compared to single-YC scheduling, the complexity of the problem increases significantly. Vis & Carlo (2010) studied

the scheduling of two crossable YCs within a yard block, while Bian et al. (2014) studied a problem in which two rail gantry cranes cannot cross each other. Lim et al. (2007) explored the parallel scheduling problem for multiple YCs and constructed a backtracking search framework, in which a simple approximation heuristic was developed to solve the model. Meanwhile, a simulated annealing heuristic with random neighbourhoods was designed for the large-scale problem.

Based on the predicted workload of each yard block at each time slot, Zhang et al. (2007) developed an integer programming dynamic scheduling model for YCs with the objective of minimising the remaining workload.

2.3.2 Methods for Yard Truck Dispatching

Exact Methods Since the yard truck dispatching problem is an NP-hard problem, only small-scale problems of this type can be solved by exact methods. Bae & Kim (2000) formulated MIP models and solved them with LINDO software. Lee et al. (2009) also applied an MIP to solve an integrated problem of yard assignments and YT dispatching, and the objective of the problem is to minimise the YTs' total congestion and waiting time. Luo et al. (2016) presented an MIP to solve a YT dispatching problem, in which storage constraints of containers were not considered. Usually, such exact methods are able to obtain an optimal solution to the problem by solving the corresponding integer or mixed-integer programming (MIP) models. However, due to their time-consuming nature, it is impractical to use such methods in real ports. The results calculated by such methods are often used as a baseline to measure the performance of other advanced methods.

Rules and simple heuristics The second type of solution methods is based on some fixed rules (heuristics) related to time, distance or process-

ing capacity of YCs, etc., such as the closest vehicle, first-come-first-served (FCFS) or shortest waiting time. Cheng et al. (2005) used a greedy strategy to minimise the total waiting time of YTs at the YCs, and then they compared it with a new network flow model, which increases the throughput of the port. Grunow et al. (2007) compared the FCFS, and the shortest-travel-time (STT) rule, which tends to dispatch a YT to pick up a new container from the nearest location. In Homayouni & Tang (2013), they dispatch YTs to the tasks by the "nearest available vehicle" heuristic rule. The objective of the problem is to minimise the total waiting time for both YTs and YCs, and according to simulation results, the utilisation of the YTs and YCs is improved. Based on a self-developed simulation model, Taner et al. (2014) compared the impact of different dispatching rules on the throughput of a container terminal. The results show a significant difference in the impact of different dispatching rules on that. To handle uncertainties in real-world YT dispatching problems in a very flexible way, Chen et al. (2016) established a dynamic heuristic that is manually crafted based on human experience. The algorithm obtained solutions that are superior to those used in practice.

These rule-based approaches are intuitive, straightforward, and easy to understand, but it is often difficult to achieve high-quality solutions since they are usually very simple. Another drawback of this approach is that developing rules for addressing scheduling optimisation problems in ports necessitates significant expertise, and they can only accommodate a subset of problem scenarios rather than all possible situations. When a container terminal undergoes changes in its equipment, route scheduling, or task scheduling strategy, experts must fine-tune these rules, incurring substantial additional time and expenses. Moreover, human cognitive constraints limit the extent to which experts can investigate these rules, allowing them to explore only a minimal portion of the comprehensive search

space. However, these methods have the advantage that they can be applied to online problems, as some rules require only local information (the real-time values of some uncertain variables and other key features of the port) to make a decision.

Optimisation Techniques The third category of solution methods is optimisation techniques such as metaheuristics, which account for approximately 75% of studies in the literature (Schwientek et al., 2017). Hartmann (2004) formulated a general model for a YT dispatching problem in container terminals. In the model, the assignments of tasks are subject to some precedence constraints, and the objective is to minimise the weighted sum of average delay and average setup time per task. The problem is solved by combining a GA with some dispatching heuristics. Nguyen & Kim (2009) developed an MIP model for an automatic lifting vehicle (ALV) dispatching problem, in which the time window constraints were considered. They solved the problem using an evolutionary algorithm based on a genetic algorithm (GA). Kaveshgar & Huynh (2015) developed a MIP model for a YT dispatching problem, in which some real-world constraints such as container priority were considered. A combination of a GA and a greedy search algorithm is then designed to solve the problem. Niu et al. (2016) has designed two swarm intelligence techniques: particle swarm optimisation (PSO) and bacterial colony optimisation (BCO), to solve a YT dispatching problem. The high performance of their methods is demonstrated by comparison with a GA. Hu et al. (2019) investigated the container-cluster operation and optimal scheduling strategy of the YTs. To address the problem, they developed a multi-objective mathematical programming model, which was subsequently solved using an adaptive GA.

These studies which used optimisation techniques such as metaheuristics, do have demonstrated favorable outcomes, as evidenced by decreased

ship docking durations, reduced empty-truck travel distances, or diminished overall truck travel distances. However, there are also some shortcomings in these methods. Firstly, metaheuristics may take a considerable amount of time to converge to an optimal or near-optimal solution, especially for large-scale and complex yard truck dispatching problems. This drawback is particularly fatal in online problems. Secondly, usually, these methods require careful tuning of parameters, such as population size, mutation rate, and crossover rate in GA, to achieve good performance. Selecting the appropriate parameters can be challenging and time-consuming. Thirdly, metaheuristics, such as GA, may require problem-specific knowledge to design effective crossover and mutation operators. This may limit their applicability to a wide range of yard truck dispatching problems.

2.3.2.1 Machine Learning Based Methods

The last category, which has developed rapidly in recent years, is machine learning (ML)-based methods. Although ML is not the exact algorithm, it has great advantages in assisting in enhancing the performance of traditional optimisation algorithms. Firstly, it replaces heavy computation with an approximation. Secondly, the use of ML reduces the difficulty of algorithm design and increases the generality of the algorithm. For example, Khalil et al. (2016) proposed an ML method to surrogate a strong branching strategy in an MIP model. Their method can generate smaller search trees compared with a state-of-the-art commercial solver. Although the problem they solved was not a YT dispatching problem, as long as we formulate a problem as a MIP, it can be solved with this method. Also, Lodi & Zarpellon (2017) provided a survey of applying ML to branch and bound.

There is less research on ML solving the YT dispatching problem, but

studies about applying ML to VRP are rich. Based on the similarities between the YT dispatching problem and VRP mentioned earlier, a collection of studies on VRP, accompanied by a limited assortment of research on the YT dispatching problems, is reviewed here.

ML-Guided Perturbative VRP Algorithms ML is widely used in helping select *perturbative* heuristics in metaheuristics or hyper-heuristics (Bengio et al., 2021). For the offline optimisation problems, where problem-related parameters are predetermined and available before solving, perturbation-based metaheuristics stand out as highly favored solution methods (Bai et al., 2023). The perturbation-based metaheuristics create new solutions by perturbing existing solutions in accordance with specific transition rules. The initial use of ML to guide the solution perturbation was perturbative hyper-heuristics, which employed various adaptive mechanisms based on the principles of traditional reinforcement learning (Burke et al., 2019). They can appropriately select a set of perturbative low-level heuristics at different problem-solving stages.

Bai et al. (2007) presented a hyper-heuristic framework for a VRP with time windows problem, in which they use reinforcement learning to assist the selection of low-level heuristics. They also explored the relationship between the memory size of the learning mechanism and the method's performance. Sabar et al. (2014b) also used reinforcement learning in a hyper-heuristic. They designed a reward in reinforcement learning based on the dynamic multi-armed bandit-extreme value, and competitive results were obtained. Asta & Özcan (2014) introduced an apprenticeship learning to hyper-heuristics. The method takes another state-of-the-art hyper-heuristic as an expert and trains a new hyper-heuristic by mimicking the behaviour of that expert. Experimental results show that the method is able to perform well on unseen problem instances. Compared with the hyper-heuristic

served as the expert, it shows better performance. Chen & Tian (2019) present a generic deep reinforcement learning-based hyper-heuristic framework, which comprises two essential components, namely a region picker and a rule picker. During the training process, the actions of DRL are to select appropriate rewrite regions and rewrite rules to modify the existing solutions, which in turn generates new solutions. The framework was tested on three classical COPs, showing better performance than Google OR-tools in VRP.

ML for Constructive VRP Solutions Some other studies explored introducing ML to the *constructive* methods that incrementally build a complete solution from scratch. This method uses ML to learn directly a constructive solver, which can be seen as a sequential decision-maker and be trained via a *reinforcement learning (RL)* or an *evolutionary learning* process. Therefore, when constructive methods are applied, the solved VRP is usually modelled as a sequential decision-making process. The benefit of such modeling is that it can better adapt to the dynamic or online nature of the problems in the real world, which require quick responses at each decision point. Compared with the research on static VRP, there are relatively fewer studies on dynamic stochastic and online VRP.

Nazari et al. (2018) was probably the first to apply attention-based models for VRP. At each decision point, both the static elements and the dynamic elements (e.g. customer demand changes after being visited) of the problem are embedded before being inputted into the attention mechanism. In Balaji et al. (2019), the stochastic and dynamic capacitated VRP problems with several constraints are studied. They formulated a Markov Decision Process for the problem and solved it with the vanilla RL technique. They also stressed the importance of developing effective simulators. Peng et al. (2019) proposes a more general model that can handle input

variations: the dynamic attention model (AM-D), which can explore and exploit hidden structural information dynamically and effectively in the construction process since the embedding of each node is updated in real-time after each truck returns to the depot.

There are few studies that directly use RL as a constructive approach to the YT dispatching problem. Zeng et al. (2011) proposed an RL-based method to construct a complete solution step by step to solve a YT dispatching problem, in which Q-learning is used to learn the dispatching rules for YTs. The experimental results show that this algorithm can efficiently solve the YT dispatching problem and shows better adaptability.

In the other branch, the traditional evolutionary algorithm and its fusion with heuristic-based methods continue to advance. Jacobsen-Grocott et al. (2017) applied a genetic programming (GP) hyper-heuristic (GPHH) method to solve a dynamic VRP with time windows (DVRPTW). They use GP to evolve low-level heuristics to accept or reject new requests. The experimental results show that the higher the degree of dynamics in the problem, the more pronounced the advantage of this method compared with the handcrafted heuristics. Conflicts exist between performance and interpretability for evolved policies. Large complex policies are usually good in performance but poor in interpretability, and conversely, simple policies are good in interpretability but poor in performance. To solve this problem, Wang et al. (2019) used three ensemble GP methods, namely, Bagging GP, Boosting GP, and Cooperative Co-evolution GP, to solve the uncertain capacitated arc routing problem. Compared with the previous GPHH, their method evolved smaller trees with better performance.

Chen et al. (2020) proposed a data-driven GP heuristic that evolved different decision-making rules (heuristics) in solving real-world YT dispatching problems in a container port. Superior results were reported in

comparison with those from Chen et al. (2016).

2.4 Metaheuristics

In the last two decades, the field of metaheuristics has grown tremendously as a solution to real-world optimisation problems. They can perform well in situations where exact optimisation techniques cannot provide satisfactory results. For complicated optimisation problems (non-deterministic polynomial time hard problems), metaheuristic techniques can produce high-quality solutions in less time than traditional optimisation techniques. They are widely used in finance, planning, scheduling and engineering design.

Most real-world optimisation problems have a high degree of complexity, non-linear constraints, interdependencies between variables and a large solution space. Some of these problems also require short response times. This calls for a technique that can solve complex optimisation problems with a fast response time. A metaheuristic algorithm is an optimisation method that can provide a relatively good solution in a reasonable time. As one type of approximation method, metaheuristics are similar to guidance mechanisms of the underlying heuristics. They are not problem or domain-specific and can be applied to any optimisation problem. The term 'metaheuristic' was introduced by Glover (1986).

The metaheuristic is formally defined as an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. At each iteration, it manipulates either a complete (or partial) single solution or a collection of such solutions. The subordinate heuristics may be high or low-level procedures, or a simple local search, or just a construction method. (Swan et al., 2022)

Metaheuristic algorithms create a balance between the *exploration* and

exploitation of the search space. Basically, *exploration* refers to the process of searching the solution space for new, potentially better solutions. It involves diversifying the search by investigating different regions of the solution space, which helps avoid getting trapped in local optima. It is crucial for discovering promising areas of the solution space that may contain high-quality solutions. *Exploitation*, on the other hand, focuses on refining and improving the existing solutions within a promising region of the solution space. It involves intensifying the search around the current best solutions to find better ones in their vicinity. And it is essential for converging towards an optimal or near-optimal solution once a promising area has been identified. Too much exploration may lead to excessive time spent searching without converging, while too much exploitation may result in premature convergence to sub-optimal solutions. (Korošec & Eftimov, 2020)

Over the past decades, many metaheuristics have been proposed, such as genetic algorithms, memetic algorithms, artificial immune systems, simulated annealing, tabu search, ant colony optimisation, particle swarm optimisation and differential evolution. A key advantage of metaheuristics lies in their nature as abstract search methods (Senington & Duke, 2013). The core search logic can be employed to address any problem that can be broken down into several fundamental components, specifically solution representation, evaluation of solution quality, and a concept of locality. The latter signifies the capacity to produce neighboring solutions through a heuristically-informed function based on one or more existing solutions.

2.4.1 Classification of Metaheuristics

Metaheuristics have a number of attributes that can be used to help classify them (Talbi, 2009), such as whether it is nature-inspired or whether it is a population-based or single-point search. The detailed classification is

as follows:

Depending on whether it is nature-inspired or not, metaheuristics can be classified as nature-inspired (e.g. evolution (GA) or ant colony optimisation (ACO)) and non-nature-inspired (e.g. Tabu search and iterated local search (ILS)).

Based on whether it is an individual or population-based search method, it includes population-based and single-point search metaheuristics. The former (e.g. GA and PSO) iterate and manipulate a group of solutions, more focused on the exploration of search space. However, the latter (e.g. TS and SA) operate on a single solution. We use this classification to introduce the metaheuristics in the following sections.

Another way to classify metaheuristics is according to whether it is *iterative* or *greedy*. Iterative and greedy approaches are two types of search strategies commonly used in metaheuristics. *Iterative metaheuristics*, such as PSO and SA, repeatedly refine and improve a solution or a set of solutions through a series of iterations. These algorithms typically involve a balance between exploration and exploitation. They often employ a combination of deterministic and stochastic search techniques to avoid getting trapped in local optima and to converge towards a global optimum or near-optimal solution. *Greedy metaheuristics*, on the other hand, make locally optimal choices at each step. These algorithms typically prioritise exploitation over exploration, which can lead to faster convergence but may also result in getting trapped in local optima. Greedy metaheuristics are often simpler and more computationally efficient than iterative metaheuristics but may not be as effective in finding high-quality solutions for complex problems.

Metaheuristics can also be classified according to whether the objective function is static or dynamic. Algorithms with a static objective function, such as PSO, always follow the objective function as given in the problem

representation, while other algorithms (e.g. guided local search) modify it during the search.

They can also be classified according to whether the algorithm only uses information from the current state. Some algorithms, such as SA, use information only about the current state of the search, while others, such as TS, utilise some information collected during the iterative process.

According to neighbourhood structures, it contains both one and multiple neighbourhoods. Most metaheuristics operate on a single neighbourhood structure (e.g. ILS). Only a few of them use multiple neighbourhood structures, for example, variable neighbourhood search (VNS).

2.4.2 Single-solution-based Metaheuristics (S-metaheuristics)

S-metaheuristics are a class of optimisation algorithms that employ a single-point search strategy. These algorithms start from an initial solution and search the solution space by locally modifying the current solution in each iteration. Although they may not explore the solution space as extensively as population-based metaheuristics, they typically have advantages in computational efficiency. There are two important actions in S-metaheuristics: generating the initial solution and selecting a neighbourhood.

- Generating initial solution

  The initial solution can be generated either randomly or by using a greedy heuristic. The use of the latter technique is more complex but results in faster convergence.

- Definition of neighbourhood

  A neighbourhood function N is a mapping, $N : S \rightarrow 2^S$, that assigns to each solution s of S (search space) a set of solutions, $N(s) \subset S$. The

neighbourhood $N(s)$ of a solution s in a continuous space is the ball with centre s and radius equal to r, with $r > 0$. For a discrete optimisation problem, the neighbourhood $N(s)$ of a solution s is represented by the set $\{s'|d\ (s', s) \leq r\}$, where d represents a given distance that is related to the move operator. A solution s' in the neighbourhood of s ($s' \in N(S)$) is called a neighbour of s. A neighbourhood with strong locality leads to better performance. Choosing a rich neighbourhood improves the chances of finding a good solution and also reduces the computation time. Some of the S-metaheuristic algorithms have been described in this section.

2.4.2.1 Simulated Annealing

The Simulated Annealing (SA) algorithm was presented and introduced to the field of optimisation by Metropolis et al. (1953) and (Kirkpatrick et al., 1983), respectively. Inspired by the annealing process in statistical mechanics, SA aims to minimize entropy production and reach the lowest energy state. It is a memoryless neighborhood search algorithm that escapes local optima to avoid premature convergence.

The SA algorithm starts with a single initial solution, and at each iteration, a new solution is generated using the random neighbourhood strategy. For a minimising problem, if the new solution's objective function value is less than the current solution, it is accepted, and the latter is replaced. On the contrary, if the value of the new solution is higher than that of the current solution, SA adopts a probability-based measure to decide whether to replace the current solution. The probability-based measure is defined as follows:

$$P = exp(\frac{\Delta E}{KT}) \tag{2.7}$$

where ΔE is the change in energy (cost function) between old and candidate

solutions, K is the Boltzmann constant and T is the control parameter value at a given iteration.

The probability measure allows the selection of non-improving solutions, thereby helping the algorithm to get out of local optima. The algorithm starts with a high initial temperature to ensure that a large number of solutions can be accepted in the beginning. As the iterations proceed, the temperature (T) reduces, and the algorithm begins to converge towards an optimum solution.

The pseudocode of SA is shown in Algorithm 1.

Algorithm 1 Pseudocode of SA

Input:
    Bounds of solution space, h_b, l_b
    Initial temperature $T_0 \geq 0$
    Cooling schedule $CT(t)$
    Maximum iteration at fixed temperature, N
    Objective function $f(*)$
Generate initial solution $s \in S$, $s \leftarrow rand \times (h_b - l_b) + l_b$, where $rand$ is a random number between $[0, 1]$.
Start iteration at initial temperature $T \leftarrow T_0$
Calculate fitness value for initial solution $val \leftarrow f(s)$
Initialize repetition counter $n \leftarrow 0$
repeat
    while $n \leq N$ **do**
        Generate candidate solution in neighbourhood of s, $s' \in S$
        Compute fitness value for candidate $val\_new \leftarrow f(s')$
        if $val > new\_val$ **then**
            $s \leftarrow s'$
        else if $P(in\ Equation\ 2.7) < rand$ **then**
            $s \leftarrow s'$
        end if
        $n \leftarrow n + 1$
    end while
    Reduce temperature as per cooling schedule, $T \leftarrow CT\ (T)$
until stopping criterion is met

The stopping criteria could be the minimum temperature reached or no change in fitness over a given number of iterations, etc.

SA can be easily implemented to give good solutions if certain precau-

tions are taken into account. First, the initial temperature should be set adequately high. Too low a temperature can cause the system to get stuck in local optima, and a very high value of T can cause difficulty in reaching the optimum solution. Also, there should be a gradual reduction in the control parameter T. Iterations at each temperature should be enough to stabilise the system. The employed cooling schedule greatly influences the quality of the solution. SA is well suited to problems with a rough landscape (large number of local optima), but if the optimisation problem has a smooth landscape with limited local optima, SA is not of much use and unnecessarily delays convergence.

2.4.2.2 Tabu Search

The TS algorithm is a local search method proposed by Glover & Laguna (1998). It employs a Tabu list to avoid revisiting previously explored solutions and accelerate the search for optimal solutions. TS, like SA, accepts non-improving solutions to escape local optima but uses a deterministic strategy to explore the entire neighborhood.

TS starts with a random initial solution and maintains a Tabu list to prevent revisiting stored solutions. At each iteration, new candidate solutions are generated. If a candidate solution is better and not in the Tabu list, it is accepted. Otherwise, the best neighbor replaces the existing solution. Talbi (2009) applied medium-term and long-term memory to reinforce and diversify TS algorithms as well. Aspiration criteria allow moves yielding better solutions to override the Tabu list.

The stopping criteria could be the minimum value of fitness function, the maximum number of iterations or the maximum iterations without improvement, etc.

Advancements in TS involve storing move or solution attributes instead

of solutions in the Tabu list, reducing the time and memory requirements of the algorithm. The size of the Tabu list can be static, dynamic, or adaptive. Improvements focus on exploiting gathered information, efficient neighborhood operators, better initial solutions, and parallel search strategies. An efficient way of utilising the information gathered is to use the elite (best) solution found so far to generate new solutions to speed up convergence. Other methods, such as reactive TS, aim to move away from previously visited local optima.

The effectiveness of the TS algorithm depends on the appropriate selection of the neighbourhood operator and search space. In addition, diversification should be very carefully handled to avoid getting stuck at local minima. Also, the tabu list should be carefully formulated to make the search effective while minimising computation time and memory requirements. TS excels in avoiding cycles but is less efficient than SA in escaping local minima.

2.4.3 Population-based Metaheuristics (P-metaheuristics)

P-metaheuristics start with a series of initial solutions distributed over the search space. These solutions are iterated and manipulated to generate a new set. Then the replacement procedure is used to select a new solution set from both the existing and newly generated solutions. This process proceeds until the stopping criteria are satisfied. P-metaheuristics vary in the way they perform generation and selection and in the way they organise search memories. These algorithms typically enable a broad search of the solution space, which helps in finding global optima or near-optimal solutions.

There are some common techniques for *population initialisation*, such as random generation, sequential diversification and parallel diversification. And the *stopping criteria* of P-metaheuristics can be a fixed number of

iterations, minimum value of the objective function or maximum iterations without improvement.

2.4.3.1 Genetic Algorithm

GA, which is inspired by Darwin's theory of evolution, was presented by Holland (1975). It is a global search heuristic, holding inherent parallelism. The individuals can be flexibly expressed, such as binary strings, tree structure, vector of real numbers, etc., based on the characteristics of a specific problem.

Each encoded solution is called a chromosome, with each decision variable in a chromosome being called a gene. A set of individuals is selected based on a selection policy to be parents. The selection policy is related to the fitness function (objective function) value corresponding to each individual. These parents, through crossover and mutation, generate offspring to be a part of a new generation. The selection policy used can be:

- Roulette wheel selection

  In this, the selection probability of an individual is directly proportional to its fitness.

- Rank-based fitness assignment

  In this method, relative fitness is associated with individuals. It prevents the selection process from being dominated by highly fit individuals, thus preserving diversity in the population.

- Tournament selection

  In this, a set of individuals is randomly chosen and the best amongst them is selected for mating.

- Elitism

In this scheme, a fixed number of the best chromosomes are kept and the remaining population is generated using any of the selection procedures discussed above. This helps in preserving the best solution in the population.

Selected individuals are used to produce new solutions (offspring) using crossover and mutation operators.

- Crossover operators

  The operation of crossover is used to produce offspring that inherit characteristics from both parents. Crossover has a high probability ranging from $0.6 - 1.0$. Some of the crossover techniques are single-point crossover, two-point crossover or uniform crossover. For real-coded GAs (RCGAs), we can use techniques such as arithmetic crossover, heuristic crossover and geometrical crossover.

- Mutation operator

  Mutation is a rare phenomenon with a low probability of around $0.001 - 0.1$. It is used to change some information contained in randomly selected chromosomes. It aids in diversification of the population and the larger exploration of the search space.

The next task is to produce a new generation. For this, the old generation needs to be replaced by the new one. The extreme replacement strategies are (Glover & Laguna, 1998):

- Generational replacement

  The replacement will concern the whole population of size μ. The offspring population will systematically replace the parent population.

- Steady-state replacement

At each generation, only one offspring is generated and it may replace the worst individual of the parent population.

Between these two extreme replacement strategies, many distinct schemes exist that involve replacement of a given number, λ ($1 < \lambda < \mu$), of individuals in the population. The pseudocode of the GA is shown in Algorithm 2.

Algorithm 2 Pseudocode of GA

Input:
    Bounds of solution space, h_b, l_b
    Population size, `NP`
    Chromosome size, `ND`
    Objective function $f(*)$
    Maximum no. of generations, `NG`
    Crossover probability, pc
    Mutation probability, pm
Generate initial population $pop \leftarrow rand(\texttt{NP}, \texttt{ND}) \times (h_b - l_b) + l_b$, where $rand(\texttt{NP}, \texttt{ND})$ creates an $\texttt{NP} \times \texttt{ND}$ random matrix, where each element is a random number between $[0, 1]$.
Calculate fitness value for initial population
for $i = 1 : \texttt{NP}$ **do**
    $val(i) \leftarrow f(pop(i))$
end for
repeat
    while no. of offspring $\leq$ required **do**
        Select parents from current population based on their fitness
        if $pc > rand$ **then**
            Offspring $\leftarrow$ crossover (selected parents)
        end if
        if $pm > rand$ **then**
            Offspring $\leftarrow$ mutate (offspring)
        end if
    end while
    Generate new population using some replacement strategy
until stopping criterion is met

In addition to the traditional crossover (e.g. uniform and single-point) and mutation (e.g. Gaussian and uniform) operators, new operators have been introduced to improve the quality of the solution. Some of the crossover operators are as follows:

- Unimodal normally distributed crossover: in this crossover, three parents are used to generate two or more children.
- Simulated binary crossover (SBX): it is used to generate offspring close to the parents.
- Hybrid crossover with multiple descendants
- Independent component-analysis-based crossover: it shows good search ability.

Variations proposed in the mutation operator are as follows:

- Directed mutation operator that introduces a bias towards promising areas.
- Principal component analysis-based mutation operator: it is used to homogenise the components such that a situation where only a few principal components are important (and the rest negligible) can be avoided to ensure diversity.

Genetic algorithms are able to evaluate the fitness of multiple individuals simultaneously and have advantages in solving large-scale optimisation problems. However, if those important parameters, such as the selection of the objective function, population size, and the probability of crossover and mutation, are not well determined, the algorithm may converge to a local optimum early in the computation. The choice of a larger population size and a larger number of generations can solve this problem to some extent, but there is a trade-off between finding a better solution and consuming suitable computation time.

2.4.3.2 Particle Swarm Optimisation

PSO is a biomimetic algorithm, proposed by Kennedy & Eberhart (1995). It is designed to mimic the process of a flock of birds searching

for food. It is widely used in optimisation areas because the programming is simple and easy to understand. Each particle in the population adjusts its velocity based on its own experience and the experience of the best-performing particle in the swarm (pBest). The velocity update equation (Equation 2.8) incorporates the inertia weight (w), acceleration constants (c_1 and c_2), and random numbers (r_1 and $r_2 \in [0, 1]$). The position of each particle is then updated using Equation 2.9.

$$v(t+1) = w(t) \times v(t) + c_1 \times r_1(gBest - x(t)) + c_2 \times r_2(pBest - x(t)) \quad (2.8)$$

$$x(t+1) = x(t) + v(t+1) \quad (2.9)$$

Researchers have made modifications to the original PSO algorithm, such as limiting the velocity increment to prevent particles from flying out of the problem space. Fuzzy techniques have been applied to control the inertia weights, and local neighborhood-based structures have been used to compute $gBest$, analogous to local search methods.

The inertia weight decides the expanse of the search space explored. A high value of inertia weight w, at the start of iteration helps in exploring a wide search space. As iterations proceed, w is reduced to intensify the search. Acceleration factors are used to make the particles follow the best obtained solution. Low values of acceleration constants allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement towards the target regions. The performance of PSO is relatively insensitive to swarm size, provided it is not too small.

2.4.3.3 Ant Colony Optimisation

ACO is an algorithm inspired by the cooperative foraging behavior of ants. It was proposed by Dorigo (1992). ACO mimics the process in which ants deposit pheromones on their paths to find the shortest route to food sources. This behavior can be exploited to find optimal solutions in optimisation problems.

The ACO starts by generating a random set of ants (solutions), whose fitness is then evaluated based on the value of the objective function. As the ants move, the pheromone concentration on each path is updated using the pheromone update equation (Equation 2.10). Here, $\Gamma_{ij}(t)$ represents the pheromone concentration at iteration t, $\Delta\Gamma_{ij}$ represents the change in pheromone concentration between iterations, and ρ is the pheromone evaporation rate.

$$\Gamma_{ij}(t) = \rho \times \Gamma_{ij}(t-1) + \Delta\Gamma_{ij} \tag{2.10}$$

By increasing the pheromone concentration on better paths, ACO reinforces good solutions. Evaporation of pheromones is necessary to prevent the algorithm from getting trapped in local optima.

Modifications have been made to the traditional ACO algorithm to enhance its performance. An elitist strategy was introduced in Drigo (1996). The elitist strategy provides a strong additional weight to the best route since the beginning. This helps the search process move towards the best solution and also increases the convergence speed. We can also introduce a limit to the value of pheromone trails.

ACO has inherent parallelism with an in-built positive feedback mechanism that helps reach the optimum solution faster. The evaporation (of pheromones) phenomenon is used to prevent premature convergence. The

value of the evaporation constant ranges from 0.01 to 0.2 (Talbi, 2009). ACO can be used to find solutions for complex optimisation problems if the parameters of the algorithm are carefully chosen.

2.4.4 Application of Metaheuristics

Metaheuristic algorithms are used in a range of real-world optimisation problems. The application areas of metaheuristics include communication, image and signal processing, scheduling problems, very-large-scale integration design and financial planning. Table 2.2 lists several application areas of metaheuristic algorithms.

Metaheuristic algorithms have garnered immense interest in the past two decades because of their applicability to real-world optimisation problems. Metaheuristics are well-suited to solving NP-hard as well as multi-objective optimisation problems. Traditional optimisation methods fail to deliver satisfactory results in such cases because of the complexity of the problem structure and the size of the problem instance.

The salient features of metaheuristic algorithms that make them better suited to real-world optimisation problems are mentioned below.

- General applicability

  Exact methods and heuristic techniques are problem-specific methodologies. For each problem, a new mechanism needs to be adopted. Also, learning from one application cannot be easily applied to another problem. However, in the case of metaheuristics, the task is to just adapt the algorithm to the specific problem rather than starting from scratch.

- Reasonable computation time

  Even for highly complex problem structures, metaheuristic algorithms

Table 2.2: Application area(s) of Metaheuristic

Algorithm	Area (s) of Application
Simulated Annealing (SA)	• Job shop scheduling in production system (Zhang & Wu, 2011) • Vehicle routing in transport and logistics management (Vincent et al., 2017) • Communication: mobile network design, routing, channel allocation (Girgis et al., 2014)
Tabu Search (TS)	• Resource allocation in industry, university etc. (Servranckx & Vanhoucke, 2019) • Engineering technology: cell placement in VLSI, power distribution, structural design (Sait & Arafeh, 2014) • Artificial Intelligence: pattern recognition , data mining, clustering (Cao et al., 2015)
Genetic Algorithm (GA)	• Financial planning, stock predictions (Inthachot et al., 2016) • Image processing: compression, segmentation (Chakrapani & Rajan, 2009) • Sequencing in FMS (flexible management systems) (Godinho Filho et al., 2014)
Particle Swarm Optimisation (PSO)	• Telecommunications: network design, routing (Khan & Engelbrecht, 2012) • Neural network training (Das et al., 2014) • System simulation and identification (Alfi & Modares, 2011) • Decision making and planning (Chen et al., 2020) • Signal processing (Ababneh & Bataineh, 2008)
Ant Colony Optimisation (ACO)	• Set problems: set partitioning and covering, maximum independent set, bin packing (Bidgoli & Haghighi, 2020) • Scheduling: flow shop scheduling, process planning (Xing et al., 2010) • Bioinformatics: protein folding, DNA sequencing (Yu et al., 2013)

can generate near-optimal solutions in a reasonable amount of time for real-time applications.

- Global exploration

  They are more suited to problems with multiple local minima than exact methods that have higher chances of being stuck at local optima. Metaheuristic algorithms create a good balance between intensification and diversification of the search space.

However, metaheuristic algorithms also have their own set of limitations. They are listed below.

- Metaheuristic algorithms do not guarantee optimal solutions.
- It is not easy to theoretically prove the efficiency of algorithms. Studies usually rely on empirical results to prove the same.
- Development times of algorithmic frameworks are often high.
- Each metaheuristic algorithm has its own set of advantages and limitations that make it more suitable for a particular kind of application. Finding the best-suited algorithm is a challenging task.

In general, S-metaheuristic algorithms are aimed at intensification. They are good at exploring promising areas of the search space. However, they are local search algorithms and are thus not able to efficiently explore the search space. This drawback is overcome by population-based methods. They iterate a set of solutions, which leads to better coverage and exploration of the search space. The drawbacks of both can be eliminated by the use of hybrid metaheuristics. This approach aims to combine more than one metaheuristic method for problem solving. Another possibility is the use of hyper-heuristics, which help in deciding on an optimum sequence of metaheuristic algorithms, combining the advantages of each, to get the

best solution possible. Thus, it can be safely said that metaheuristics, hybrid metaheuristics and hyper-heuristics provide an efficient way of dealing with highly complex optimisation problems encountered in industrial and scientific domains.

2.5 Hyper-heuristics

This section presents a survey of hyper-heuristics which is one of the key techniques used in this thesis. The goal is not only to review the existing research in the related topics but also to put the contributions of this thesis into a wider context.

The term hyper-heuristic was first used in Denzinger & Fuchs (1996) to describe a protocol that combines several artificial intelligence methods for automated theorem proving. It was independently used in 2000 (Cowling et al., 2000) to describe 'heuristics to choose heuristics' in the context of combinatorial optimisation. However, the idea of automating the heuristic design process is not new. Actually, it can be traced back to the early 1960s (Fisher & Thompson, 1963), and was independently developed by a number of authors during the 1990s (Fang et al., 1993; Hart et al., 1998). A hyper-heuristic either selects a heuristic from a set of predefined low-level heuristics or generates a heuristic based on pre-existing low-level heuristic components for solving a given problem (Burke et al., 2013; Pillay & Qu, 2018). In the context of hyper-heuristics, a primary goal is to develop search heuristics that are easy-to-implement and applicable to various optimisation problems (Burke et al., 2013). In contrast, the problem-dependent parameters in traditional search heuristics require expertise to tune in order to reach the desired performance level. Hyper-heuristic approaches aim to provide relatively good solutions, with or without little change, on a range of optimisation problems by leveraging the capabilities of multiple low-level

heuristics. This level of generality to achieve desirable results on various problems is inspired by the demand to develop easy-to-maintain algorithmic frameworks that could work on different instances of a certain problem or even across various problems, and provide solutions of acceptable quality (Burke et al., 2013).

2.5.1 A Classification of Hyper-heuristic Methodologies

Hyper-heuristics are broadly divided into two main types: algorithms for *selecting* heuristics from a predefined set of low-level heuristics, and algorithms for *generating* heuristics from a predefined set of low-level heuristic components with the aim of solving a given problem (Burke et al., 2010a). Based on this classification, the following subsections are structured as shown in Figure 2.5.

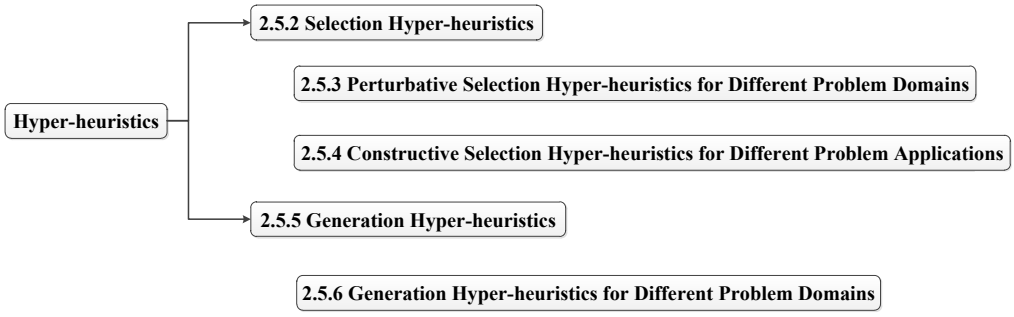

Figure 2.5: *The structure of the section of hyper-heuristics.*

Hyper-heuristics can also be broken down according to the learning methodologies into three categories: online-learning, offline-learning, and no-learning hyper-heuristics.

- Online-learning hyper-heuristics interact with the search environment and learn from the feedback while solving a given problem.

- Offline-learning hyper-heuristics are those hyper-heuristics that learn

on some training instances before the actual search starts.

- No-learning hyper-heuristics do not use any learning mechanism.

2.5.2 Selection Hyper-heuristics

A selection hyper-heuristic applies a chosen low-level heuristic to the current (partial) solution to output a new (partial) solution at each step of a search, before deciding whether to accept or reject the newly created solution. The traditional hyper-heuristics were developed from the algorithmic framework proposed by Cowling et al. (2000). The framework requires only limited information, such as the number of low-level heuristics, the direction of the optimisation process (maximisation or minimisation) and the value of the objective function for a given solution, to complete the optimisation. This modular design and the use of the concept of domain barriers prohibit the hyper-heuristic from utilising any problem or domain-specific information, making them a more general approach to computational search. A selection hyper-heuristic or its components can be reused (with no or only minor modifications) across problems. Low-level heuristics (LLHs) typically implement simple neighbourhood moves, such as swap, shift or basic local search operations. Meanwhile, more complex heuristics, such as metaheuristics, are also available as LLHs.

Selection hyper-heuristics can also be further classified into different categories based on the following characteristics. (Drake et al., 2020)

2.5.2.1 Low-level Heuristics

Nature of the Search Structure As mentioned in Burke et al. (2010a), according to the nature of the search structure, the low-level heuristics consist of constructive and perturbative heuristics. A typical perturbative low-level heuristic can be defined as a function that returns a complete

solution given one or more complete solutions. While, a constructive low-level heuristic is a function that returns a partial or complete solution given one or more partial or empty solutions.

Nature of the Low-level Heuristic Set The types of low-level heuristics in selection hyper-heuristics can be very diverse, such as mutational, ruin-recreate, local search (hill climbing), crossover or metaheuristics. Meanwhile, the size of the low-level heuristic set can be changed. For a reduced set of low-level heuristics, it may exclude a (some) particular type(s) of heuristics (e.g., crossover operators) and for an increased set, it may create new heuristics by combining existing heuristics. A selection hyper-heuristic can be designed to manage the original whole set of predefined (e.g., unary, binary, n-nary) low-level heuristics, a reduced set, or an increased set.

2.5.2.2 Nature of Move Acceptance

The nature of move acceptance depends on the nature of accept/reject decisions as well as the parameter-setting method used by the move acceptance method (Jackson et al., 2018). The move acceptance mechanism in selection hyper-heuristics can be classified as stochastic if a probabilistic framework is considered while making the accept/reject decision (e.g., Simulated Annealing), or non-stochastic otherwise. Non-stochastic move acceptance methods can be further classified into basic methods, such as accepting All Moves (AM), accepting Improving or Equal moves (IE) and accepting Only Improving moves (OI), and threshold acceptance methods (e.g., Great Deluge, Late Acceptance Strategy, etc.).

2.5.2.3 Nature of Parameter Setting

Heuristic selection, move acceptance or low-level heuristics often carry parameters to be set or controlled. The algorithmic parameters for the

heuristic selection and move acceptance methods need to be handled at the hyper-heuristic level, while the setting of low-level heuristic parameters can be handled by either the hyper-heuristic or the low-level heuristic itself. Eiben et al. (1999) provided three categories of parameter control for evolutionary algorithms: deterministic, adaptive and self-adaptive. The parameters can be set statically to a fixed value prior to the search process, dynamically, allowing the value to change in a predefined manner, or adaptively allowing the value to change in a reactive manner during the search process.

2.5.3 Perturbative Selection Hyper-heuristics for Different Problem Domains

A large number of problem domains have been addressed by different researchers. This thesis focuses on three of them: puzzles and games, scheduling and timetabling. The final class comprises problems that do not fall into any of these categories.

2.5.3.1 Puzzles and Games

It is hardly unexpected that puzzles and games have attracted the interests of academics working on selection hyper-heuristics since they have long been a favourite playground of artificial intelligence researchers. The Eternity II Puzzle is an edge-matching puzzle requiring the placement of patterned tiles on a grid so that the edges of neighbouring tiles have a similar pattern. Wauters et al. (2012) submitted the winning entry for a global competition to solve this problem, where the objective is to maximize the number of matching edges. Using a series of low-level heuristics that swap and rotate tiles, Simple Random heuristic selection was paired with various acceptance criteria, with supplementary objective functions employed to

direct the search.

Li & Kendall (2015) introduced hyper-heuristic players for a number of games, defining high-level strategies using a variety of low-level heuristics, including iterated prisoners dilemma and repeated *Goofspiel*. The hyper-heuristic game players were shown to outperform their constituent low-level heuristics, using dynamic strategies over time.

2.5.3.2 Scheduling

In its brief history, selection hyper-heuristics have been frequently used to scheduling difficulties. Asta et al. (2016b) applied a lifelong learning multi-stage hyper-heuristic to personnel scheduling. This approach employs two selection hyper-heuristics both embedding Simple Random heuristic selection, but with different move acceptance methods. The algorithm consists of four phases overall which are cycled through periodically, performing tensor analysis in the first three phases to configure the algorithm to be used in the final phase. Tensor analysis is used to discover which low-level heuristics perform well with which parameter settings and move acceptance method. In the last phase, an iterated multi-stage algorithm randomly chooses between Simple Random-Improving and Equal or Simple Random-Naive Acceptance and runs that hyper-heuristic using the previously learned configuration for a fixed duration. This process is repeated until the time allocated for the final phase ends. Two variants of the approach with different memory settings for learning are tested on nurse rostering benchmarks. The results showed that including a 'forgetting' mechanism performs slightly better than remembering everything from the start of the search process, improving upon the best known results for four instances. Nurse rostering was also considered by Rahimian et al. (2017). Their work presented a hybrid approach, combining a solution construction stage with variable neighbour-

hood descent operating over a search space five low-level heuristics, before a final Integer Programming-based ruin-and-recreate phase. The proposed approach was shown to outperform two state-of-the-art approaches from the literature in many cases on well-known existing benchmarks.

Selection hyper-heuristics for maintenance scheduling have also been researched. Pour et al. (2018) uses a Choice Function hyper-heuristic to designate the working regions of engineers executing maintenance operations on the Danish rail network. At each phase, tasks are redistributed from one region to another using one of five low-level heuristics chosen based on Choice Function results. Chen et al. (2017) managed a series of six low-level heuristics to build maintenance plans for preventative and corrective maintenance of an urban water drainage system across a rolling time horizon using binary exponential backoff, a Tabu-based greedy technique.

A hyper-heuristic GA, which selects from multiple crossover and mutation operators at each generation, was used by Wu et al. (2016) to solve software project scheduling problems. Koulinas et al. (2014) used a Particle Swarm Optimisation-based hyper-heuristic operating over eight low-level heuristics to solve well-known benchmarks for resource constrained project scheduling. A tabu search-based hyper-heuristic algorithm was presented by Koulinas & Anagnostopoulos (2013) to solve special cases of the resource-constrained project scheduling problem in the context of the construction industry. The 'Dominance-based and Roulette Wheel with an Adaptive Threshold Acceptance' (Asta et al., 2016a) hyper-heuristic was used as a part of a hybrid approach that won the MISTA 2013 challenge on 'multi-mode resource-constrained multi-project scheduling problem'. This multi-stage selection hyper-heuristic method combines two different stages. The first stage evaluates all low-level heuristics and maintains an active subset of the best-performing heuristics, using a dominance-based mechanism that

keeps heuristics that are not dominated by any other low-level heuristics in the subset. The heuristic dominance strategy measures the improvement in the objective function value and the number of steps required to achieve that improvement. This stage also assigns a score to each low-level heuristic. The subsequent hyper-heuristic stage is invoked, using the active subset of heuristics with their associated scores. The proposed multi-stage hyper-heuristic is used as a local search method for improvement as a part of a population-based hybrid approach.

2.5.3.3 Timetabling

Timetabling problems, which are closely connected to scheduling, have often been handled using selection hyper-heuristics.

Burke et al. (2014) investigated two-stage techniques for exam timetabling, which first develop a plausible solution and then use a hyper-heuristic to sequence perturbative low-level heuristics for improvement. The evaluated methods sequentially apply each heuristic from an evolving list. A method that adaptively modifies the length and content of sequences performs very well on the Toronto benchmarks but badly on the ITC2007 dataset.

Soria-Alcaraz et al. (2014) evaluated a hyper-heuristic-enhanced form of Iterated Local Search for post-enrollment course scheduling. During the improvement phase, the hyper-heuristic utilises an Adaptive Operator Selection technique based on credit assignment to regulate nine low-level heuristics. The suggested technique performed less well than state-of-the-art approaches, yet it produced the new best solution for one of the ITC2007 problem instances. Soria-Alcaraz et al. (2017) enhanced their approach by identifying a collection of low-level heuristics utilising non-parametric statistical testing and fitness landscape probing methods, which performed the best among their Iterated Local Search algorithms for post-enrollment

course timetabling. They also tried this strategy for vehicle routing, although it did not outperform the current best methods.

Da Fonseca et al. (2016) proposed a three-stage approach referred to as GOAL (Group of Optimisation and Algorithms) which won the third international timetabling competition (ITC2011), based on a real-world high school timetabling problem. The first stage of the approach constructs an initial solution which is then fed into a hyper-heuristic controlling six low-level heuristics for improvement. The heuristic selection component of this hyper-heuristic chooses a heuristic with a prefixed probability. Simulated Annealing with reheating is used for move acceptance. In the last stage, an Iterated Local Search algorithm is used, employing two low-level heuristics, each selected at random during the search and accepting improving moves only. Kheiri & Keedwell (2017) suggested a hidden Markov model-based sequence-based selection hyper-heuristic. This adaptive method keeps scores indicating the likelihood of selecting a low-level heuristic based on the previously invoked heuristic, therefore learning successful sequences of low-level heuristics to apply. During the search phase, the scores are updated using a Reinforcement Learning approach. The suggested hyper-heuristic beats GOAL by achieving the new best results in seven cases while matching the best-known outcomes in four instances from the ITC2011 benchmark.

2.5.3.4 Other Application Domains

In addition to the aforementioned well-known problem domains, this section reviews a number of additional problem domains addressed in the literature by selection hyper-heuristics.

In retail, shelf space allocation, where the objective is to maximise the use of search space according to some quality metric, is a significant problem. Bai et al. (2013) examined a two-dimensional shelf space alloca-

tion model using Simple Random and Reinforcement Learning heuristics with Simulated Annealing acceptance. Inspired by this study, Zhao et al. (2016) offered an expanded model intended to represent the practicality of the real-world scenario more accurately. To overcome this challenge, they used a Simple Random - Simulated Annealing hyper-heuristic acting over a collection of low-level heuristics limited to viable regions of the search space.

The area of telecommunications has a daily influence on the lives of the majority of people throughout the globe. Yang et al. (2014) presented a hyper-heuristic Genetic Algorithm to handle the problem of frequency assignment (FAP). This method generates sequences of low-level heuristics to apply to the current solution during the search. To solve the challenge of picking a cluster head in wireless sensor networks, Tsai et al. (2017) employed various metaheuristics, randomly chosen and applied for a defined number of iterations.

Several problems in industrial design have been solved using selection hyper-heuristics, the majority of which are continuous rather than discrete optimisation problems. Allen et al. (2013) applied a number of selection hyper-heuristics to the challenge of optimising aircraft structural design.

Splines are piecewise polynomial functions that can be constructed from a set of control points. Using a set of low-level heuristics operating over a space of control points, representing unmanned aerial vehicle routes, Akar et al. (2014) compared a number of well-known selection hyper-heuristics and a Genetic Algorithm. Using the OneMax and GapPath functions as examples, Lehre & Özcan (2013) analysed the expected runtime of a simple selection hyper-heuristic. This work concluded that in the case of some problem domains, mixing low-level heuristics can be more effective than using a single low-level heuristic. Alanazi & Lehre (2014) also analysed the

runtime of selection hyper-heuristics, comparing different learning mechanisms commonly used in the literature. Using the simple Leading Ones function as an example, similar performance was observed for all four learning mechanisms tested. Sabar et al. (2017) tuned deep belief networks for image recognition utilising a Multi-armed Bandit Monte Carlo selection hyper-heuristic, exceeding results reported by conventional meta-heuristic approaches. The Kampouridis et al. (2013) proposed a method for financial forecasting that uses low-level heuristics for Genetic Programming trees. This system picks from a collection of up to fourteen low-level heuristics to change decision trees, using a Reinforcement Learning method based on a roulette wheel, and shows superior performance to a well-known existing tool.

2.5.4 Constructive Selection Hyper-heuristics for Different Problem Applications

The majority of the selection hyper-heuristics discussed previously in this subsection work over sets of perturbative low-level heuristics conducting the search with complete solutions to optimisation problems. In addition, a limited number of published articles propose selection hyper-heuristics that choose from a range of constructive low-level heuristics to create solutions from empty or incomplete solutions. Numerous techniques of this sort use an evolutionary algorithm to generate sequences of low-level heuristics for solution creation.

2.5.4.1 Graph-Colouring Hyper-Heuristic for Timetabling

Firstly, we introduce a representative example of a constructive selection hyper-heuristic. In educational timetabling, the scheduler must arrange a collection of courses or examinations into the available time slots while

adhering to certain limits. In some studies, the problem is modelled as a graph colouring problem. In particular, events (such as courses) are represented as nodes in a graph, whereas edges denote conflicts between events. Graph heuristics in timetabling select the events according to their properties, such as the number of conflicts with other events, and then place them sequentially in the time slots. For a particular ordering strategy, events are assigned with varying degrees of difficulty. Therefore, the planner assigns the most challenging event first. The graph-based hyper-heuristic suggested in Burke et al. (2007c) consists of the following six heuristics:

- Largest Degree (LD): Orders the events in decreasing order based on the number of conflicts the event has with the other events.
- Largest Weighted Degree (LW): The same as LD, but the events are weighted by the number of students involved.
- Colour Degree (CD): Orders the events in decreasing order in terms of the number of conflicts (events with common students involved) each event has with those already scheduled.
- Largest Enrolment (RO): Orders the events in decreasing order based on the number of enrolments.
- Saturation Degree (SD): Orders the events in increasing order based on the number of timeslots available for each event in the timetable at that time.

A potential solution is composed of a series of low-level heuristics. A solution (timetable) is constructed via iteration. At the ith iteration, the ith graph-colouring heuristic in the sequence is used to order the unassigned events. The initial events e in the ordered series are then allocated to the time slots with the lowest cost.

By applying Tabu Search as the top-level search approach, good sequences of graph-colouring heuristics are found. A complete solution is constructed by successively using the graph-colouring heuristics in order to sort the remaining events and choose the relevant events to put in the timeslots. Since each heuristic in the list is used to schedule a number e of events, the length of the heuristic list is n/e, where n is the number of events to be scheduled. Tests were performed for $e = 1, \ldots, 5$ (details can be found in Burke et al. (2007c)). This work also highlights the existence of two search spaces in constructive hyper-heuristics (the heuristic space and the problem solution space). The approach was tested on both course and exam timetabling benchmark instances with competitive results. Later, this graph-based hyper-heuristic was further extended in Qu & Burke (2009), where a formal definition of the framework is given. Additionally, the authors compare the performance of a number of high-level heuristics that act on the search space of heuristics. In particular, a best-improvement hill-climber iterated local search and variable neighbourhood search are implemented and contrasted with the previously implemented tabu search.

2.5.4.2 Other Problem Applications

Qu et al. (2015) used a simple Estimation of Distribution Algorithm, a Univariate Marginal Distribution Algorithm (UMDA), to generate probability distributions from which to derive sequences of low-level heuristics at different stages of a search. Using five well-known graph colouring low-level heuristics, each low-level heuristic in a sequence is applied consecutively to assign an exam to a time slot. The quality of the solutions found was shown to be competitive with existing hyper-heuristic approaches to construct timetables for the Carter benchmark set.

Another area that has previously seen a high level of research interest

for constructive selection hyper-heuristics is constraint satisfaction problems (CSPs). The order in which variables are selected to be instantiated can have a significant impact on the cost of computing the solution of a CSP. However, although many variable ordering heuristics exist, predicting the performance of a heuristic for a particular problem in advance can be difficult. Ortiz-Bayliss et al. (2013) used learning vector quantisation (LVQ), a type of supervised neural network, to learn a series of rules mapping between the features of the region of search space currently being explored and an appropriate heuristic action to take at that point. Based on the constraint density and the constraint tightness of the current problem state, one of four low-level heuristics for variable ordering is used. Based on a similar overall framework, Ortiz-Bayliss et al. (2016) used a Genetic Algorithm with variable length, encoding increasingly sophisticated rules with a larger number of features and heuristic actions. Each chromosome in the Genetic Algorithm has 10 values, nine of which correspond to landscape characteristics, and the remaining one specifies which of seven low-level heuristics for variable ordering should be used when these landscape aspects are met. When solving a CSP, the Genetic Algorithm chromosome with landscape characteristics closest to the present solution state is identified, and the matched low-level heuristic is executed. Following this work, by performing heuristic filtering, Gutierrez-Rodríguez et al. (2017) reduced the size of the low-level heuristic set. Their experiments revealed two heuristics that greatly outperformed the other heuristics studied, both separately and when combined inside a hyper-heuristic framework. Rosales-Pérez et al. (2017) provided a co-evolutionary method for heuristic selection that uses supervised multilabel classification to identify subsets of heuristics that perform well for certain circumstances. This method outperformed prior work (Ortiz-Bayliss et al., 2016), greatly lowering the time required to solve a

collection of well-known CSP benchmarks.

Crawford et al. (2013) used Choice Function variants to adaptively rank eight enumeration strategies during the process of solving CSPs, where the set of enumeration strategies consists of combinations of variable and value selection heuristics. Each Choice Function variant is composed of a number of weighted indicators (such as the number of visited nodes, number of backtracks, number of steps, etc.), used to assess the performance at intermediate stages of the search process. The weighting for each of these indicators is controlled by Particle Swarm Optimisation. The proposed method is able to find good solutions on average across different problems (N-queens, Magic Square and Latin Square).

López-Camacho et al. (2014) presented a unified framework for solving one and two-dimensional, regular and irregular bin packing problems. A set of six low-level heuristics, to decide which object to place next, and where to place it, were used to iteratively construct solutions. A Genetic Algorithm was used at the high-level, evolving a set of rules to govern which heuristic to apply in a given solution state. The framework was able to generalise well across a variety of problem instances without additional parameter tuning, outperforming the constituent low-level heuristics. Thomas & Chaudhari (2014) also considered two-dimensional bin packing, using a Genetic Algorithm to select a subset of items forming a sub-problem, then selecting a placement strategy from three low-level heuristics via a greedy method. Sequences of low-level heuristics were also evolved using a Genetic Algorithm by Salhi & Vázquez Rodríguez (2014), this time for constructing solutions for a flow show scheduling problem variant. This method was shown to perform very well, outperforming a large number of existing strategies from the literature. When addressing multi-objective two-dimensional bin packing problems, Gomez & Terashima-Marín (2018) generated rules

for low-level heuristic selection. On the basis of a comparable structure to that used by Ortiz-Bayliss et al. (2016) for generating solutions to CSPs, a set of rules were evolved that map distinct areas of the search space with certain attributes to heuristic actions. Depending on the current state of the solution, one of forty operators consisting of an ordering heuristic and a packing heuristic is picked and used to pack the next item. Compared to the use of a single heuristic, the performance of an approach utilising many heuristics was significantly enhanced.

Li et al. (2014) introduced an Ant Colony Optimisation-based hyper-heuristic. Their research explored a range of assignment and sequencing low-level heuristics to provide solutions for a production scheduling problem in cellular manufacturing systems needing 'intercell' transfers. The performance of the hyper-heuristic technique scaled much better than that of CPLEX as the size of the problem instance grew. A similar question was resolved by Li et al. (2015), who offered a bi-level strategy to create and choose combinations of heuristic rules. Genetic Programming was used to generate candidate rules in order to create a search space for a Genetic Algorithm-based selection hyper-heuristic to operate across. In terms of both solution quality and execution time, the developed system demonstrated remarkable performance.

2.5.5 Generation Hyper-heuristics

This section introduces examples of methods that have the capacity to produce heuristics automatically for a specific problem. Numerous methodologies documented in the literature employ genetic programming (GP) (Burke et al., 2009), a subfield of evolutionary computation that can generate computer programs automatically, to generate heuristics. GP draws inspiration from the principles of natural selection and genetics to iteratively

improve the performance of programs.

At its core, GP involves the creation and evolution of a population of candidate programs. These programs are typically represented as *syntax trees*, where the nodes represent functions or operations, and the leaves represent variables or constants. For example, Figure 2.6 shows the tree representation of the program $max(a * b, b - 2 * a)$.

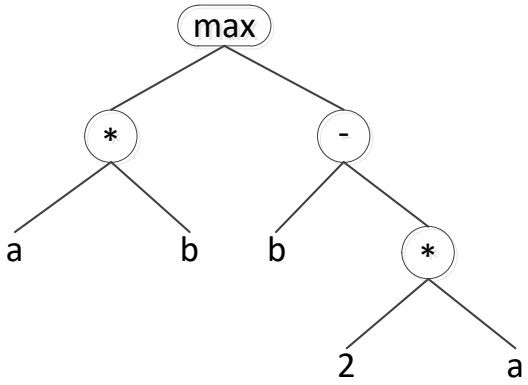

Figure 2.6: *GP syntax tree representing max(a*b, b-2*a).*

The first step of GP is to obtain the initial population. It can be done randomly or with domain-specific knowledge. The next step in GP is fitness evaluation, where each program's fitness is assessed. A fitness function is defined to measure how well a program performs on a given problem. This function quantifies the program's ability to solve the problem and serves as the basis for selection.

Selection is a crucial aspect of GP, as it determines which programs will be chosen for reproduction and the creation of the next generation. Various selection methods can be employed, such as tournament selection or fitness proportionate selection. The goal is to favor programs with higher fitness scores, as they are more likely to produce better offspring.

Once the selection process is complete, genetic operators like *crossover* and *mutation* are applied to the selected programs. Crossover involves exchanging genetic material between two parent programs, creating new offspring with a combination of their traits. Mutation introduces random

changes in the offspring by altering or replacing certain traits.

After the genetic operators are applied, the new offspring form the next generation of the population. This iterative process of selection, crossover, and mutation continues until a termination condition is met. Termination conditions can include reaching a maximum number of generations, achieving a satisfactory fitness level, or reaching a predefined time limit.

Throughout the iterations, GP aims to evolve programs that exhibit improved performance on the given problem. By exploring a vast search space and gradually converging towards optimal or near-optimal solutions, GP demonstrates its effectiveness in solving difficult COPs, including Boolean satisfiability (Bader-El-Den & Poli, 2007; Fukunaga, 2008), bin packing (Burke et al., 2010b; López-Camacho et al., 2014; Sim et al., 2015), the travelling salesman problem (Keller & Poli, 2007a,b) and some real-world problems, such as container terminal truck dispatching (Chen et al., 2020), production scheduling (Hart & Sim, 2016; Yska et al., 2018; Zhang et al., 2021a), medical applications such as ambulance dispatching (MacLachlan et al., 2022) and observing satellite scheduling (Zhang et al., 2018) in astronomy geography.

Some GP-based hyper-heuristics have evolved local search heuristics (Bader-El-Den & Poli, 2007; Fukunaga, 2008; Keller & Poli, 2007a,b) or even evolutionary algorithms (Oltean, 2005). An alternative approach is to utilise genetic programming to develop a program that represents a function that is part of the processing of a given problem-specific construction heuristic (Burke et al., 2007b; Tay & Ho, 2008). Most examples of employing genetic programming as a hyper-heuristic are offline in the sense that a training set is used to generate a heuristic program, which is then applied to unseen instances of the same problem. Thus, the objective is to produce reusable heuristics. However, research has also been undertaken on dis-

posable heuristics (Bader-El-Den & Poli, 2007; Keller & Poli, 2007a,b). In other words, heuristics are developed to solve a specific or a small number of problem instances with a high degree of similarity.

2.5.6 Generation Hyper-heuristics for Different Problem Domains

2.5.6.1 A Representative Example

A typical example of heuristic generation using genetic programming is given in this section. It evolves packing heuristics that use a constructive way to solve a one-dimensional bin packing problem. This task entails packing a collection of n things into bins of a specified capacity C using the fewest feasible bins. In the online version of the problem, neither the quantity nor the dimensions of the objects are known in advance. In contrast to the offline form of the problem, where the list of objects to be packed is provided at the outset, the offline version of the problem does not have this characteristic. First-fit, a construction heuristic used in online bin packing, sequentially packs a group of objects in the order in which they are supplied. The heuristic iterates over the available bins, placing the current item in the first bin that it fits.

In Burke et al. (2006, 2007b), the online bin packing problem was solved by a constructive hyper-heuristics. The evolved heuristics, depicted as trees, function within a fixed framework that mimics the action of the first-fit heuristic. The core idea is to utilise the qualities of the items and bin capacities, which reflect the problem state, to develop functions (expressions) that would drive the packing process. Each evolved function (GP tree) is applied in turn to the available bins, returning a value. If the value is less than or equal to zero, then the system moves on to the next bin. But if the value is positive, the item is packed into the current bin. In

this way, the function decides when to stop the search for a suitable bin for the item. The algorithm (depicted in Algorithm 3) then repeats the process for each of the other items until all the items have been packed.

Algorithm 3 Pseudocode of genetic programming framework for bin packing

```
1: for each item p do
2:     for each item b do
3:         output = value obtained by evaluating Heuristic
4:         if output ≥ 0 then
5:             place item p in bin b
6:             break
7:         end if
8:     end for
9: end for
```

A genetic programming framework requires the specification of terminals and functions. The hyper-heuristic architecture for online bin packing utilises features that are related to the problem state to define terminals. In Burke et al. (2006), the following terminals are chosen:

- S, the size of the current item,
- C, the capacity of a bin (this is a constant for the problem) and,
- L, the load of a bin (i.e., the total size of all the items occupying that bin).

Later (Burke et al., 2007a), these three attributes were replaced by only two attributes: S, the size of the current item and E $(= C - L)$, the residual capacity of a bin (i.e. how much space is remaining in the bin). The function set used in Burke et al. (2006, 2007b) consists of $\leq$ $, +, -, \times, \%$, where $\%$ is the 'protected divide function'. Results from Burke et al. (2006) demonstrate that a basic genetic programming system can uncover human-designed heuristics such as first-fit, whereas Burke et al. (2007a) developed heuristics that surpassed first-fit. In Burke et al. (2007b),

it was also shown experimentally that the selection of the training instances (classified according to the item size distribution) influences the trade-off between the performance and generality of the produced heuristics and their application to new situations.

2.5.6.2 Generation Hyper-heuristics for Other Problem Domains

The most prevalent method for generating heuristics so far has been tree-based genetic programming. Recently other genetic programming methods have been developed such as grammar-based (Sosa-Ascencio et al., 2016), gene expression programming (Sabar et al., 2014a,b), and grammatical evolution (Burke et al., 2011; Sabar et al., 2013).

Other machine learning methods and representations have also been adopted. The researchers devised a general evolutionary algorithm that produces and searches heuristics in the form of policy matrices while interacting with an online bin packing simulator for assessment (Asta et al., 2016c; Özcan & Parkes, 2011). The results demonstrate that the produced heuristics are tailored to the distribution of item sizes and outperform those devised by humans. In Sim et al. (2015), a method for lifelong learning is described in which an artificial immune system is integrated with genetic programming in a system that continually generates new heuristics and samples problems from the environment. The system obtained good results on a large number of examples generated by the dynamically changing one-dimensional bin packing environment.

Fan et al. (2021) presents a novel approach to the dynamic job shop scheduling problem with extended technical precedence constraints. The authors propose a mathematical programming model and employ a construction heuristic known as dispatching rules (DR), which is automatically generated by GP as a hyper-heuristic. The strength of this approach lies in

its ability to automate the generation of scheduling rules, eliminating the need for manual design and adjustment. However, the method may require substantial computational resources and time for training and optimising the DR.

Zhang et al. (2021b) propose a correlation coefficient-based recombinative guidance for GP hyper-heuristics (GP-HH) to solve flexible job shop scheduling (FJSS), a problem that has significant implications in manufacturing and service industries. Their method is a two-fold process. First, they employ a correlation coefficient to measure the relationship between different heuristics, which allows them to identify which heuristics are more likely to produce effective offspring when recombined. Second, this information is used to guide the recombination process in GP, thereby enhancing the efficiency and effectiveness of the GP-HH.

Luo et al. (2022) propose a GP-based approach to automatically design priority rule-heuristics for the resource-constrained project scheduling problem (RCPSP). They use a new fitness function that considers the total weighted tardiness of the project schedule and the results show the high generalization ability and superior performance of GP in designing priority rules for RCPSP.

Tan et al. (2020) propose a cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds. This approach is based on the idea of coevolution and can allocate resources more efficiently than other methods. Additionally, it can learn the workload patterns and virtual machine types from historical workload traces.

2.6 Deep Reinforcement Learning (DRL)

Since the methods proposed in this thesis are DRL-based hyper-heuristics, it is imperative to provide an introduction to DRL. This will facilitate a comprehensive understanding for readers of the approach being presented.

The idea of reinforcement learning (RL) has its roots in behavioural psychology. According to Thorndike's Law of Effect (Thorndike, 1927), in a given specific situation, if an animal behaves in a way that makes it feel comfortable, the connection between the behaviour and the specific situation will be reinforced, and the animal will be more likely to behave in that way again when that specific situation arises, resulting in pleasurable feedback. On the contrary, if the animal behaves in such a way that makes it feel uncomfortable, it will be more likely to avoid the behaviour when the situation arises again. In other words, the mapping between behaviour and a situation depends on the feedback it receives in that situation. RL is a process of behavioural adjustment that simulates the interaction of animals with the environment in which they live. Two distinguishing features distinguish it from supervised learning. Firstly, RL is not based on static labelled data but on data generated by agent-environment interaction over time. Secondly, the feedback obtained is not a clear-cut evaluation of whether it is right or wrong but rather a vague evaluation. RL acquires knowledge from interactive actions in the environment and the corresponding evaluations and improves action strategies to adapt to the environment to achieve the desired goal (Minsky, 1954). The immediate objective of RL is to learn a mapping from the environment to an optimal sequence of actions.

The theoretical framework of RL was formed in the 1980s. After combining operations research and control theory, breakthroughs were made in both theory and algorithms, and good applications were achieved in se-

quential decision problems such as automated control and optimal decision making (Frank & Claus, 2006). RL usually uses trial-and-error mechanisms. Since rewards are often delayed, classical RL can only improve its action policies based on the sparse rewards in learning, while the exploration is mainly based on random trial-and-error without effective strategies for action selecting. Thus, in classical RL, "infinite" and "sufficient" access to the state-action pairs is required to make the exploration converge. In more complex problems, dimensional catastrophes can occur due to a large number of state-action pairs. At this stage, the dominant approach to solving the Curse of Dimensionality (Keogh & Mueen, 2017) is to abstract and generalise the state-action space.

Recently, RL has been further developed in combination with deep learning. Deep neural networks can automatically perceive and extract advanced features from data, while RL can iteratively improve decision-making by 'trial and error' interactions but is poor at directly perceiving information from complex environments. In 2015, Mnih et al. (2015) combined convolutional neural networks and Q-learning to successfully open up research in deep reinforcement learning (DRL).

In order to apply RL to solve a problem, the problem needs to be modeled as a sequential decision-making process, where the agent engages with the environment by executing a series of actions to find a solution. The Markov Decision Process (MDP) emerges as a prevalent mathematical framework for modeling problems as a sequential decision-making process. (Bellman, 1957). MDP can be defined as a five-element-tuple, namely (S, A, R, P, γ), where:

- S is the state space, $s \in S$ represents a specific state.
- A is the action set. $S \to A$ denotes the mapping of states to actions, i.e. the set of actions available to the agent in state s.

- R: the set of instant rewards, $r(s, a)$ denotes the reward obtained by taking action a in state s.
- P denotes the state transfer probability matrix.
- γ is the discounted factor, adjusting the RL agent to balance immediate and future rewards.

Broadly, the RL algorithms can be split into the *model-based* and *model-free* categories, as shown in Figure 2.7.

- *Model-based* methods focus on the environments, where transition functions are known or can be learned, and can be utilised by the algorithm when making decisions. This group includes Monte-Carlo Tree Search (MCTS) algorithms such as AlphaZero (Silver et al., 2016) and (Schrittwieser et al., 2020).
- *Model-free* methods do not rely on the availability of the transition functions of the environment and utilise solely the experience collected by the agent.

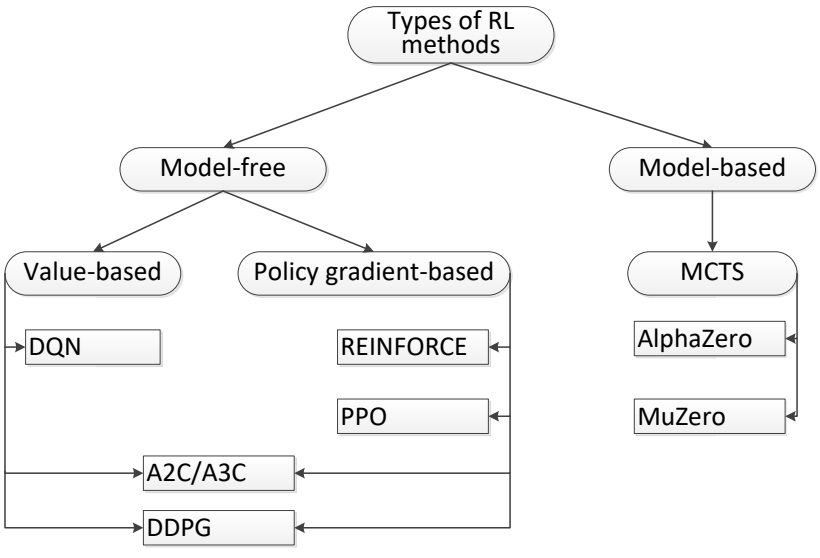

Figure 2.7: *A classification of reinforcement learning methods.*

Furthermore, *model-free* methods can be split into two big groups of RL algorithms, namely policy gradient-based and value-based methods. The value-based DRL fits a value function with a deep neural network, and policy gradient-based DRL fits the policy directly and uses the policy gradient method to find the optimal policy. Additionally, there are RL algorithms, such as actor-critic methods (A2C/A3C) (Mnih et al., 2016) and deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), that combine policy gradient-based methods with value-based methods.

As value function-based DRL is applied in this study, we will not go into too much detail about the policy gradient-based DRL algorithms. We first introduce the Q-learning algorithm and the basic concepts involved in the RL algorithm. Several typical DRL algorithms are then introduced.

2.6.1 Q-learning

RL agents are able to maximise their cumulative rewards by exploring different action sequences in a specific environment. In other words, RL agents can learn from their own experience and achieve the desired performance through feedback in interaction with the environment. Thus, an RL algorithm is considered a machine learning algorithm based on a trial-and-error mechanism and reward feedback.

The RL agent needs to explore the environment and make decisions in the current state s, with the purpose of maximising the cumulative rewards. Mostly used action selection policies are $\epsilon-greedy$ and $softmax$ algorithms. When choosing an action, $\epsilon-greedy$ can effectively balance the *exploration* and *exploitation*. Similar to the concepts of *exploration* and *exploitation* in other algorithms or domains, they refer to exploring new solution spaces and exploiting known promising solutions, with the only difference being the implementation. For example, the $\epsilon-greedy$ algorithm can be expressed

as:

$$a = \begin{cases} \arg\max_a Q & \text{if } \beta \geq \epsilon, \\ a \ random \ action & \text{if } \beta < \epsilon \end{cases} \tag{2.11}$$

where $\beta \in [0, 1]$, ϵ denotes the probability of exploration. Specifically, to obtain a better policy, in the initial state, the RL agent randomly selects an action with probability ϵ. In the subsequent decision points, the Q-value of each action is first calculated. Then, the action with the highest Q-value is selected with probability $1 - \epsilon$ (*exploitation*) and the action is selected randomly with probability ϵ (*exploration*) according to the greedy algorithm. Larger values of ϵ lead to larger variance and fluctuations of the Q-values. However, smaller values of ϵ can cause the RL algorithm to fall into a local optimum. Therefore, it is crucial to choose a proper ϵ value. It is important to note that during training, the current optimal action is not truly optimal until the RL model converges. In this case, the advantage of the $\epsilon - greedy$ is highlighted. Namely, it allows the RL agent to choose a random action with probability ϵ. Therefore, it does not blindly trust the current "optimal" action, thus providing the opportunity to jump out of the local optimum and explore new actions.

After taking an action, the state moves to the next one. Meanwhile, the RL agent receives an immediate reward, which will directly determine the RL agent's long-term cumulative reward. For convenience, the long-term cumulative reward based on state s and action a is represented by the state value function $Q(s, a)$, denoted as Equation 2.12.

$$Q(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a] \tag{2.12}$$

The goal of an RL model is to learn the optimal policy to maximize the long-term cumulative reward. In other words, the goal of an RL agent is to

find the optimal $Q^\star(s, a)$ as shown in Equation 2.13.

$$Q^\star(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^\star(s', a')|s, a] \tag{2.13}$$

Where s' and a' denote the state and action of the next time step, respectively. Clearly, searching $Q^\star(s, a)$ is a Bellman optimality problem and can be solved by dynamic programming.

The Q-learning algorithm is a basic RL algorithm that features a Q-table to store the algorithm's core, i.e. the state-action values (Q-values). Specifically, each time it interacts with the environment, the Q-learning agent first checks whether the current state already exists in the Q-table. If so, it updates the corresponding state-action values in the Q-table according to Equation 2.14.

$$Q(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a')|s, a] \tag{2.14}$$

If not, it creates a new item in the Q-table to store all the state-action values of the current state and updates the corresponding state-action values in the Q-table. The Q-learning agent continuously interacts with the environment until the algorithm converges. Figure 2.8 illustrates the structure of the Q-learning algorithm, in which the top left is the Q-table used in Q-learning. The Q-learning algorithm allows for easy and fast updating of the Q-table. However, when it has a large state-action space, the Q-table can be very large, which will lead to inefficient searching for pre-existing state-action values. Furthermore, the Q-table is powerless for estimating non-existing state-action values. Therefore, the capability of RL algorithms that use Q-tables to store state-action values is limited.

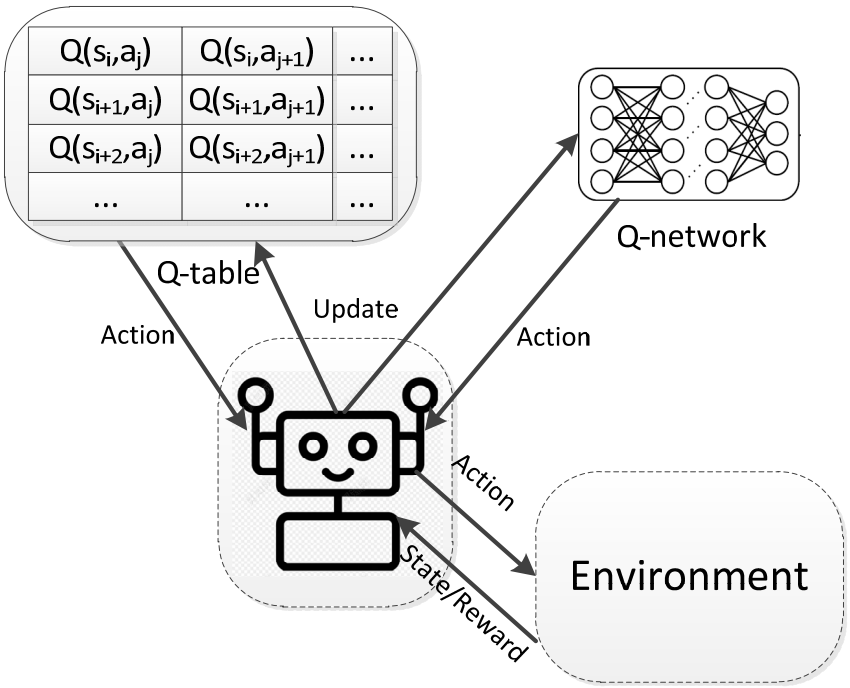

Figure 2.8: *The structure of the Q-learning and DQN algorithm*

2.6.2 Deep Q-Network

The Deep Q-Network (DQN) uses the powerful approximation capacity of deep learning, i.e. the neural network, to replace the original Q-table of the Q-learning algorithm, thus solving the limitations of the Q-learning algorithm. The DQN can efficiently estimate the state-action values with the help of neural networks which have powerful perception capabilities. Figure 2.8 illustrates the structure of the DQN, too. Specifically, each time it interacts with the environment, the DQN agent will input a state obtained from the environment into the neural network, which then estimates all the state-action values of the current state (the agent can choose between multiple actions in a state). After that, the agent selects the action with the largest state-action value. When the action is taken, the environment updates the state and gives the corresponding reward to the agent. Finally, the agent uses the reward to update all neural network parameters.

Since the DQN uses a neural network to replace the Q-table of the Q-learning, the state-action function $Q(s, a)$ of the Q-learning is converted to

$Q(s, a; \theta')$, where θ' represents all neural network parameters. Furthermore, the state-action values of the DQN are updated according to Equation 2.15.

$$Q(s, a; \theta') = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta')|s, a] \tag{2.15}$$

The loss function of the neural network is Equation 2.16.

$$J(\theta') = (Q(s, a; \theta') - (r + \gamma \max_{a'} Q(s', a'; \theta')))^2 \tag{2.16}$$

Where $Q(s, a; \theta')$ denotes the direct output of the neural network of inputted state-action pair (the estimated value), $(r + \gamma \max_{a'} Q(s', a'; \theta'))$ denotes the state-action value of the current time step calculated by the state-action value of the next time step and the reward obtained in the current time step, which is a more 'real' value. The neural network parameters are updated by Equation 2.17.

$$\theta' = \theta' + (\frac{\partial J(\theta')}{\partial \theta'}) \tag{2.17}$$

Although the DQN addressed the limitations of the Q-Learning algorithm by using a neural network, it only utilizes one neural network to estimate the values of state-action pairs. This leads to the outputted state-action values changing significantly as the neural network parameters are updated, resulting in unstable training of this algorithm. Meanwhile, owing to the max operator in Equation 2.15, the DQN agent is more likely to select overestimated values, leading to over-optimistic value estimates.

2.6.3 Double Deep Q-Network

(Van Hasselt et al., 2016) designed the Double DQN (DDQN) algorithm to solve the training instability of DQN. Additionally, it can reduce overestimations by decomposing the maximising operation in the target into

action selection and action evaluation. DDQN utilises two neural networks, namely the estimation network and the target network, to estimate the state-action values. The target network makes the fitted targets, namely the state-action values, more stable over a certain period. Specifically, the DDQN agent uses Equation 2.18 for state-action value estimation.

$$Q(s,a;\theta') = \mathbb{E}_{s'}[(1-\Gamma)Q(s,a;\theta') + \Gamma(r + \gamma \max_{a'} Q(s',a';\theta))] \tag{2.18}$$

Where Γ, θ' and θ denote the learning rate and the parameters of the estimation network and the target network, respectively. The loss function for the estimation network is given by Equation 2.19, where the fitted target in DDQN is Equation 2.20.

$$J(\theta') = (Q(s,a;\theta') - (r + \gamma \max_{a'} Q(s',a';\theta)))^2 \tag{2.19}$$

$$Y_t^{DDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a} Q(S_{t+1},a;\theta_t);\theta_t^-) \tag{2.20}$$

The parameters of the estimation network are also updated by Equation 2.17. However, the parameters of the target network are only updated every τ steps by copying from the estimation network. Here, τ is a parameter that defines how frequently the target network parameters θ are updated. The specific parameter update formula is expressed as Equation 2.21.

$$\theta = \alpha\theta + (1-\alpha)\theta' \tag{2.21}$$

Where α is the learning rate for updating the target network parameters. Obviously, DDQN effectively solves the algorithm training instability problem by clever use of the target network. However, if there is a large number of actions in the DRL model, DDQN still suffers from a slow con-

vergence speed.

2.6.4 Dueling Double Deep Q-Network

The Dueling Double Deep Q-Network (D3QN) algorithm improves the previous DRL algorithm by changing the structure of the neural networks. It divides the Q-network into the state value function, denoted as $V(s, \theta', \alpha)$ and the advantage function, denoted as $A(s, a, \theta', \beta)$. The former is only related to the state s and not to the specific action to be taken, while the latter is related to both the state s and the action a. The ultimate value function can be expressed as Equation 2.22, where θ' represents the common network parameters, α represents the unique part of the network parameters of the value function, and β represents the unique part of the network parameters of the advantage function.

$$Q(s, a, \theta', \alpha, \beta) = V(s, \theta', \alpha) + A(s, a, \theta', \beta) \tag{2.22}$$

In order to clarify the respective roles of the state value function and the advantage function, and improve their identifiability, the actual formula used is changed to Equation 2.23. Minus $\frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s, a', \theta', \beta)$ is a centralisation of the advantage function and will lead to more stable training.

$$Q(s, a, \theta', \alpha, \beta) = V(s, \theta', \alpha) + (A(s, a, \theta', \beta) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s, a', \theta', \beta)) \tag{2.23}$$

2.6.5 Experience Replay and Prioritized Replay Buffer

In a DRL system, data is relatively expensive to acquire because it requires the interaction of the agent with the environment. It is not desirable to discard data after only one-time use. Meanwhile, the training data obtained from the interaction between the DRL agent and the environment

are correlated rather than independently and identically distributed. However, as supervised learning models, deep neural networks require that the data be independently and identically distributed. *Experience replay* (Lin, 1993) solves this problem by breaking the correlation between the data by storing the experience in a pool and sampling them during training. Specifically, at each time step, a piece of data e is obtained, indicated as a tuple (s, a, r, s'). e is then added to the data pool. During the training process, a mini-batch of training data (a random set of experiences) is sampled from the data pool. When using mini-batch stochastic gradient descent (SGD) for optimisation, it is generally assumed that the samples are relatively independent of each other, thus making the noise in the data in each batch cancel each other out. At the same time, experience replay also improves data utilisation, allowing the algorithm to converge faster. The flowchart of experience replay is shown in Figure 2.9.

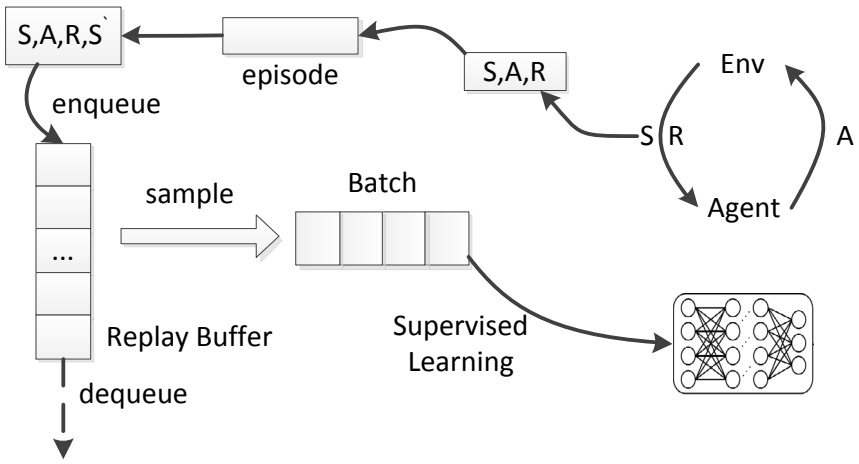

Figure 2.9: *The flowchart of experience replay.*

In traditional *experience replay*, the mini-batch data is randomly sampled from the experience pool with equal probability, but in practice, the performance improvement gained by the agent is different for different sam-

ples. The *Priority Replay Buffer* adds a priority to each experience e: the greater the TD-Error of an experience, the more we need to sample it for training, i.e., this experience has higher priority; conversely, this experience has lower priority. Taking the DDQN as an example, the TD-Error is expressed as Equation 2.24, where Y_t^{DDQN} is the fitted target shown in Equation 2.20.

$$TD - Error = Q(s, a; \theta') - Y_t^{DDQN} \tag{2.24}$$

However, using TD-Error directly as a priority indicator makes the sampling only focus on experiences with high TD-Error. This narrow sampling is more likely to cause overfitting. To solve this problem, Schaul et al. (2015) proposes a new stochastic sampling method, namely *priority replay*. The sampling probability of transition i is calculated by Equation 2.25,

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.25}$$

where $p_i > 0$ is the priority of transition i. α is a hyperparameter that adjusts the degree of priority. When $\alpha = 0$, this equation is equivalent to the uniform case.

In summary, research on DRL was started by DQN, which solves the low search efficiency of the Q-table and the poor generalisation ability of Q-learning when the state-action space is large. DDQN introduces a target network, which solves the overestimation problem of DQN and makes the training more stable. D3QN has improved the structure of the neural network. It decouples the previous Q-network into a state value network and an advantage network, making the training more efficient. The state value network is more concerned with changes in state, while the advantage network is more sensitive to the differences between different actions in a

given state. The introduction of ER both breaks down the correlation between the data and increases the utilisation of the data, and PR provides a more subtle sampling method related to expected learning progress, making training even more efficient.

2.6.6 DRL for COPs

In the context of the COP field, a straightforward approach to categorising RL is based on the specific family of techniques employed to address the given problem. As depicted in Figure 2.7, the RL methods can be classified at two levels: the first level distinguishes between model-based and model-free methods, while the second level encompasses policy gradient-based, value-based, and Monte-Carlo Tree Search approaches.

Also, RL approaches can be categorised based on the way the solution is searched by the learned agent. Specifically, methods can be classified into *constructive* methods or *improvement* methods. The former constructs solutions incrementally by employing the learned policy to select each element for inclusion in a partial solution, and the latter commences with an arbitrary complete solution and iteratively learns a policy to enhance it.

Furthermore, an additional division can be made based on the type of encoders utilised to represent the states of the MDP (Mazyavkina et al., 2021). In order to process the input structure S (e.g., graphs) of COPs, a mapping is present from S to a d-dimensional space $\mathbb{R}^d$. Such a mapping is called an *encoder* as it encodes the original input space. The encoders vary depending on the particular type of the space S. A lot of effort has been made to explore utilising different encoders in RL over the last years to solve COPs.

As mentioned before, RL methods focused on solving TSP and VRP, which are relatively close to problems solved in this thesis, are mainly re-

viewed. An initial attempt to apply RL to COPs was made by Bello et al. (2016). In their approach to solving TSP, the MDP representation involves a state vector that represents the current tour of nodes at time step t. The action is selecting a node not included in the current state. The initial state, s_0, is the embedding of the starting node. The transition function, $P(s, a, s')$, returns the next node of the constructed tour until all nodes have been visited. The negative tour length is chosen as the reward of the RL framework. The pointer network, proposed by Vinyals et al. (2015), is used as the encoder to encode the input sequence, and the solution is constructed sequentially. The training process is parallel and asynchronous, similar to Mnih et al. (2016), and the REINFORCE algorithm is used to update the parameters of the controller.

Later works, such as Khalil et al. (2017), have built upon Bello et al. (2016). In Khalil et al. (2017), the MDP for solving the TSP is similar to Bello et al. (2016), with the exception of the reward function. The reward is defined as the difference in cost functions after transitioning from state s to state s' when taking action a. The cost function is a negative weighted sum of the tour length since the weighted variant of the TSP is solved. They use the *structure2vec* (Dai et al., 2016) as the encoder for encoding the partial solutions and DQN as the chosen RL algorithm for updating the network's parameters.

Nazari et al. (2018), also motivated by Bello et al. (2016), focuses on solving VRP. Since the approach proposed in Bello et al. (2016) cannot be applied directly to solve VRP due to its dynamic nature. To overcome this issue, they extend the previous methods used for TSP to solve VRP and its stochastic variant. The state representation is modified to include the coordinates of the customer's location and their demand at the current time step. The action is to select the next node to visit in the vehicle

route. The reward is the negative total route length, given to the agent only after satisfying all customers' demands, which is the terminal state of the MDP. In terms of the encoder, they simplify the encoder of Bello et al. (2016) by replacing the LSTM unit with 1-d convolutional embedding layers. This modification allows the model to handle dynamic state changes. The learning process is performed using the REINFORCE algorithm for both TSP and VRP, while A3C is used for stochastic VRP.

In a similar vein to Nazari et al. (2018), Deudon et al. (2018) adopts a similar approach to Bello et al. (2016) but with a modified encoder-decoder network architecture. While the MDP remains unchanged from Bello et al. (2016), the Graph Neural Network (GNN) encoder architecture in Deudon et al. (2018) relies solely on attention mechanisms, encoding the input as a set rather than a sequence. The decoder, however, remains the same as in the Pointer Network case. The REINFORCE algorithm is employed to update the parameters of the described encoder-decoder network.

Kool et al. (2018) proposes a REINFORCE method to solve TSP, two variants of VRP (Capacitated VRP and Split Delivery VRP), Prize Collecting TSP (PCTSP) and Stochastic PCTSP (SPCTSP). In this study, they have employed an encoder-decoder architecture comparable to that of Deudon et al. (2018), specifically utilising a Transformer-like (Vaswani et al., 2017) attention-based encoder. However, they have made slight modifications to the training procedure and have opted for a simple rollout baseline instead of a critic-learned baseline, resulting in improved performance. The MDP formulation in this case aligns with the approaches used by both Deudon et al. (2018) and Bello et al. (2016).

Cappart et al. (2021) integrates two approaches, RL and constraint programming (CP), to address TSP with time windows. To encode the problem, a dynamic programming formulation is proposed and it also served

as a connection between RL and CP. This formulation can be presented as both an MDP and a CP problem. A state s comprises three components: the remaining cities yet to be visited, the last visited city, and the current time. An action a corresponds to the selection of a city. The reward $r(s, a)$, is defined as the negative travel time between two cities. This MDP can subsequently be transformed into a dynamic programming model. Finally, DQN and PPO algorithms have been chosen to train the neural network.

In contrast to prior studies that customise their approaches for specific COPs, Drori et al. (2020) present a novel approach that offers a general framework for model-free RL. This framework utilises a GNN representation, which can adapt to various problem classes by solely modifying the objective function. The proposed framework represents problems using the edge-to-vertex line graph and formulates them within a single-player game framework. The MDPs for TSP and VRP align with those introduced in Bello et al. (2016). And a graph attention network (GAT) is used as the encoder.

Lu et al. (2019) learns an *improvement* method which iteratively improves the complete solutions for capacitated VRP. The policy is parameterised by a GAT and is trained with the REINFORCE algorithm. They have developed an intrinsic MDP that integrates both the features of the current solution and the historical information. The action selection involves deciding between two sets of operators that modify the current solution, for instance, the 2-Opt heuristic, which involves removing two edges and reconnecting their endpoints. Specifically, these two sets of operators consist of improvement operators, chosen based on a learned policy, and perturbation operators, chosen when encountering a local minimum.

In summary, in recent years, DRL approaches for COPs have explored various encoders, such as Pointer Network, S2V and GAT. Although these

encoders entail numerous intricate details, it is crucial to grasp that all of them are the differentiable functions optimised by the gradient descent that return the encoded vector representations, which then can be used by the RL agent. Researchers have also employed constructive or improvement methods to search the solutions, and utilised different RL algorithms such as REINFORCE, DQN, and PPO. These studies have demonstrated the potential of DRL in solving challenging COPs and have paved the way for further advancements in this field.

2.7 Summary

This chapter provides the preliminary for the problems addressed in this thesis and related methods employed. Basically, DRL-based Hyper-Heuristics is utilised to address real-world online truck dispatching problems in ports, which involve varying degrees of uncertainties. To understand the context of the studies conducted, it is important to have prior knowledge, including the definition of (online) COPs, hyper-heuristics, and DRL. Besides, because metaheuristics are commonly used methods for solving COPs, and the third sub-study in this thesis explores an integration method that combines DRL-based hyper-heuristics with one of the metaheuristics, specifically GA, to tackle a relatively more challenging problem, this chapter also introduces metaheuristics.

We begin by providing a general definition of COPs. Subsequently, some important concepts, such as dynamic or static, deterministic or stochastic, online or offline and uncertainties are introduced through a classic COP (VRP). By leveraging these concepts, we can effectively classify the solved problem within the realm of COPs and identify the characteristics of this category of problems. Then, we focus on introducing the online container truck dispatching problem addressed in this thesis and highlight that it is

an online, dynamic and stochastic problem.

Next, metaheuristics and hyper-heuristics are introduced separately. For metaheuristics, we first provide different classification methods for them, followed by some typical examples, such as SA (Single-solution-based metaheuristics), GA and PSO (Population-based metaheuristics). For hyper-heuristics, we have also provided various classification methods for them. The following subsections are organized based on the classification (heuristics to *select* low-level heuristics or heuristics to *generate* heuristics from a predefined set of low-level heuristics) and followed by applications of the two groups of methods in COPS in various fields.

Finally, we introduced the last important technique used in this thesis, which is DRL. DRL can be categorised into model-based methods and model-free methods. The latter can further be divided into policy gradient-based and value-based methods. Since one of the widely used value-based methods, Double DQN, is adopted in this thesis, only a few value-based methods have been introduced. Then, two important techniques for enhancing DRL, namely experience replay and prioritized replay buffer, were discussed.

2.7.1 Summary of Literature Review

For the literature review, we have reviewed four types of existing methods for the yard truck dispatching problem, including exact methods, rules and simple heuristics, optimisation techniques such as metaheuristics, and machine learning-based methods. Section 2.3.2 presents the merits and demerits associated with each of these approaches.

In particular, relevant literature has been further reviewed in the sections on hyper-heuristics and DRL. In the context of addressing online COPs with uncertainties, particularly with regard to the online container truck

dispatching problem, we provide a concise summary of the characteristics, advantages, and limitations of these approaches.

Hyper-heuristics Hyper-heuristics, as a methodology for solving COPs, have a notable advantage, namely providing a general problem-solving framework. Unlike traditional problem-specific heuristics, hyper-heuristics operate at a higher level of abstraction, allowing for the development of reusable and adaptable algorithms. This characteristic enables hyper-heuristics to tackle a wide range of COPs without the need for extensive problem-specific modifications.

However, when addressing online COPs with uncertainties, there are also certain limitations associated with hyper-heuristics. This type of problem is typically formulated as sequential decision-making processes, where decisions need to be made rapidly at each decision point, and once made, these decisions cannot be altered in the future. Consequently, at a given decision point, when crucial dynamic (real-time) information that is vital for decision-making has not yet emerged, the decisions made are often of low quality.

Regarding the first category of hyper-heuristics, namely *perturbative hyper-heuristics*, they operate on complete solutions. This implies that they need to provide a complete solution before crucial dynamic (real-time) information becomes available. Using sampling values based on the historical distribution of this crucial dynamic (real-time) information is not helpful in solving online COPs with uncertainties. The reason is that in this type of problem, the temporal order of information arrival is crucial. Consequently, perturbative hyper-heuristics might not be usually suitable for addressing online COPs with uncertainties, directly.

For *constructive hyper-heuristics*, a decision is made step by step at each decision point, which perfectly fits the sequential decision-making pat-

tern for solving online COPs with uncertainties. However, both *constructive selection* and *constructive generation* hyper-heuristics exhibit certain limitations.

Firstly, for *constructive selection* hyper-heuristics, they typically require knowledge of all future information in order to perform well. For example, in online 2-D bin packing problems, the dimensions of all the items (rectangles) need to be known in advance because the algorithm necessitates an initial sorting of all the rectangles. Then, based on this sorting, it chooses a proper rule (low-level heuristic) to select a rectangle to pack. However, for online 2-D bin packing problems, rectangles are revealed one by one, and the dimensions of future rectangles cannot be known in advance. Therefore, constructive selection hyper-heuristics cannot provide high-quality solutions for this problem. Similarly, in timetabling problems, all the events that are not yet scheduled need to be known in advance and are then ordered by certain low-level heuristics. For instance, events are ordered according to the number of feasible time-slots available in the partial solution at that time, or in terms of the number of conflicts they have with those already scheduled in the timetable. For real-world online problems, this required information is quite often not available.

Secondly, for *constructive generation* hyper-heuristics, they are indeed a type of good approach for solving online COPs with uncertainties, with Genetic Programming (GP) serving as a prominent representative within this category. By expanding the size of the terminal set and operation set while simultaneously relaxing the depth limitation imposed on evolved programs (GP trees), the evolved GP trees can express highly intricate mapping relationships and become more powerful. However, this expansion also leads to a substantial increase in the search space, therefore, more computational time. On the other hand, when compared to DRL, GP necessitates a

longer training time. This is due to the time-consuming nature of evaluating the fitness of evolved programs in the context of solving online COPs with uncertainties. For a given problem instance, such evaluation requires using programs to guide the solution construction process step by step at each decision point, either within a simulator or a real-world environment. For GP, the entire evolutionary process generates a substantial number of programs (population size * number of evolved generations). Similarly, when employing DRL to guide the solution construction process, a complete solution is gradually built at each episode. However, for the same problem, the number of episodes required during the training process of DRL is significantly smaller compared to the number of evolved programs generated throughout the entire evolutionary process of GP. Consequently, GP necessitates a more extensive training time. Recently, Zhang et al. (2022a) have proposed some interesting acceleration mechanisms for data-driven GP. Some initial trials suggest that such methods occasionally encounter convergence issues (i.e., the populations fail to converge due to imprecise fitness evaluations).

Overall, the main strength of GPHH lies in its explicitness of the rule expressions that provide a relatively higher degree of interpretability compared to deep neural networks. However, in practice, limited by the width and height of the GP trees and the size of the terminal set and the operation set, GP's expressive power may be weaker than deep neural networks in expressing complex nonlinear and high-dimensional relationships. There are a few interesting early developments by combining DRL with GP approaches within a hyper-heuristic framework, which can be seen as a valid extension of our work.

DRL Inspired by the success witnessed in playing games, numerous studies in recent years have ventured into exploring the utilisation of DRL for addressing COPs. Several points highlight the strengths of using DRL in

this context: Firstly, the main pro of using DRL algorithms to solve COPs is the considerable reduction in running time compared to the ones obtained by the metaheuristics and solvers (Mazyavkina et al., 2021). Secondly, DRL allows for flexible problem representation by utilising neural networks as function approximators. This enables the model to learn complex mappings between problem states and corresponding actions. Thirdly, DRL is adept at solving sequential decision-making problems, making it well-suited for addressing online COPs with uncertainties. Besides, DRL models can adapt to the dynamic environments by continuously learning and updating their policies, allowing them to maintain optimal or near-optimal solutions even in changing conditions.

The studies conducted in this thesis focus on designing good DRL-based hyper-heuristics for solving online COPs with uncertainties. The first sub-study attempted to use DRL directly to solve the online container truck dispatching problem. The second sub-study aims to put DRL into the hyper-heuristic framework, thereby achieving a certain level of problem-agnostic generality while also leveraging the aforementioned advantages of DRL. Meanwhile, it enhances the interpretability of the solutions it provides. In the third sub-study, we explored using DRL-based hyper-heuristics to serve as an integration approach of DRL and metaheuristics. Despite its satisfactory performance in specific problem scenarios, when used alone, the metaheuristic is difficult to effectively and efficiently handle intricate COPs (Wang et al., 2021). Therefore, it is necessary to explore some mechanisms, such as a learning mechanism, to combine the advantages of both metaheuristics and DRL-based hyper-heuristics to improve search efficiency. The subsequent three chapters will provide a comprehensive exposition of the three sub-studies conducted in this thesis.

Chapter 3

A DRL method to solve Online Container Truck Routing Problem with Uncertainties

3.1 Introduction

In a container terminal at a port, container trucks are important horizontal transport tools for terminal operations, and their efficiency directly impacts the operational efficiency of the terminal. Due to changes in the internal and external environment of the container terminal, there are several uncertainties in the scheduling of container terminal resources, such as vehicle breakdowns and other mechanical equipment failures, the impact of external vehicles on traffic within the container terminal, and fluctuations in the efficiency of individual machines (e.g. quay crane) due to the operator (e.g. driver and crane operator). Sometimes, these uncertainties may have a decisive impact on optimisation solutions. In order to be closer to the real-world port environment, uncertainties are increasingly being taken into account in port optimisation and scheduling problems.

In this chapter, we have solved the challenging real-world online con-

tainer truck routing problem presented in Section 2.2. Since DRL is designed to solve sequential decision problems and is able to maximise the cumulative reward through its learning mechanism, it has an inherent advantage in solving online problems. Besides, it has several other advantages when used to solve COPs (as summarised in Section 2.7.1). Therefore, we have firstly applied the traditional DRL approach, Double DQN (see details in Section 2.6.3), to solve the problem and obtained better results than the manual crafted heuristic presented in Chen et al. (2016).

3.2 The Double DQN based Method for Online Container Truck Routing Problem

In this section, we use DDQN (introduced in Section 2.6.3) to solve the online container truck routing problem proposed in Section 2.2. Although the availabilities of other DRL methods that may be better than DDQN are noted, the focus in this research is on the interactions between DRL and online combinatorial optimisation problems, rather than exploring the best DRL method. The particular focus is centred on the hybridisation of data-driven and model-driven schemes. When applying a DRL algorithm to the port truck scheduling problem, the latter needs to be transformed into a Markov Decision Process (MDP). Firstly, we need to define a set of features describing the states of the dispatching system. These features are then used as input states S for the MDP. After that, the action set A is designed according to the dynamics of the system. Finally, the reward function, which is associated with the optimisation objective of the system, needs to be designed. There is only one central *agent* in this framework, namely the task dispatching planner. It is activated whenever an idle truck appears, then it randomly selects an action or chooses the action with the highest

Q value according to Equation 2.11. The proposed DDQN framework is shown in Figure 3.1.

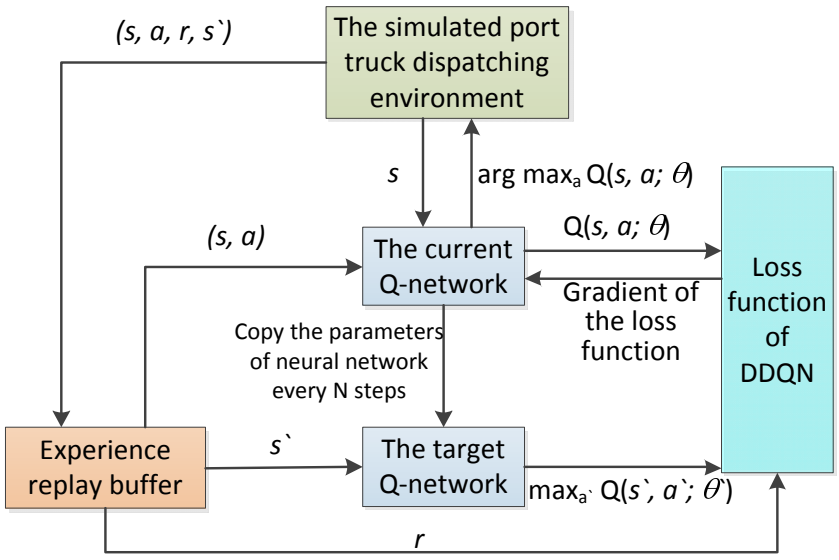

Figure 3.1: *The proposed DDQN framework*

3.2.1 State Design

In the previous fully data-driven DRL approach, which is used to solve many reinforcement learning problems, such as playing Atari games, the screenshot images are directly fed as a state to the neural network. However, we propose a different idea of state design. First, although we cannot directly obtain an overall snapshot of the port truck dispatching environment (like the screenshots of Atari games), we can intuitively observe some important features, such as the number of tasks remaining in each work queue, from the port scheduling system. We call these features the *explicit partial solution states*, which are similar to the screenshots in playing Atari games. Secondly, unlike playing Atari games, most port scheduling systems are self-developed and are highly open to scheduling managers. This means that we have access to the internal dynamics of the port scheduling system model. Based on the dynamics of the model, we can obtain values for some

features that need to be predicted. We call such features derived from the dynamics of the scheduling system model the *model-derived solution states*.

This state design can be extended to different problems. Generally, the first state vector (the explicit partial solution states) contains all the necessary information about the current partial solution, including any constraints and efficiency indicators of key resources of the problem under concern. The second state vector (the model-derived solution states) is on the projected solution states at any future point, being estimated through the deterministic model of the problem.

For the real-world container truck routing problem faced we solved in this chapter, it requires professional advice on selecting features to encode a state. The following features have been seen as vital by our collaborators in one of the largest ports in the world. Therefore, when dispatching a task, we selected them subsequently to be part of the state.

- Feature 1: the remaining number of tasks each QC needs to finish (i.e. the length of the work queue),
- Feature 2: the distance between the current position of the truck to be dispatched and the source nodes of the first tasks of every work queue,
- Feature 3: the predicted number of trucks to serve every QC, including trucks already dispatched and to be dispatched in the near future (i.e. the supply),
- Feature 4: the predicted number of tasks to be finished at every QC in the near future (i.e. demand). In this application, this is set to 10 minutes.

The first two features are *explicit partial solution states* which can be directly acquired from the problem environment, denoted as $[rn_1, rn_2, \cdots, rn_m]$

and $[d_1, d_2, \cdots, d_m]$, respectively. The latter two need to be estimated by the problem mathematical model, which are the *model-derived solution states*. For example, a moving-average method is applied to predict the service time of a task in a QC. Algorithm 4 denotes how the problem mathematical model calculates the predicted number of tasks to be finished at every QC in 10 minutes.

The last two state vectors can be expressed as $[wn_1, wn_2, \cdots, wn_m]$ and $[pn_1, pn_2, \cdots, pn_m]$, respectively. Thus, the final state is a concatenation of all four items, represented as:

$[[rn_1, rn_2, \cdots, rn_m], [[d_1, d_2, \cdots, d_m], [wn_1, wn_2, \cdots, wn_m], [pn_1, pn_2, \cdots, pn_m]]$.

Algorithm 4 Calculation of the predicted number of tasks to be finished at every QC in 10 minutes.

```
timespan = 10 min
t = 0
//store the results, the number of elements in the list is equal to the number of work queues predict_numbers = [0, 0, ··· , 0]
for each work queue, wq_i do
    for each task t_j in wq_i do
        t_service = service time of t_j
        t = t + t_service
        if t ≤ timespan then
            predict_numbers[i] += 1
        else
            Break
        end if
    end for
end for
return predict_numbers
```

3.2.2 Action Design

As mentioned in section 2.2, The online dispatch pattern that we used in this formulation is *truck-oriented dispatching*, which means that a new dispatching request should be invoked each time a truck is idle. Specifically,

when a new truck has just joined the system or a truck has just completed its current task, the planner is activated, and a task is selected and dispatched to the idle truck. As described in Figure 2.2, all undispatched tasks are first grouped into different work queues. There is a constraint (2.3) for the work queues, namely all tasks are dispatched exactly in the order given by the work queues. Meanwhile, the task sequencing in each work queue is already decided separately and is not part of the optimisation in our problem. This means that we cannot skip the first task of each work queue and execute the later ones. Therefore, the action in the DRL framework is to select a suitable work queue from all work queues ($WQ_l, l \in N^{ship}$). In other words, once the planner (the DRL agent) has selected a work queue, the corresponding task is determined. Thus, the action set is denoted as $A = \{i \mid i \in (0, 1, 2, \cdots, n_{ship} - 1)\}$, where n_{ship} is the number of QCs.

3.2.3 Reward Design

Typically, an immediate reward r_t is a scalar value that the agent receives after taking the chosen action in the environment at each time step t. Since the objective of this problem is to minimise the aggregated QC waiting times between two consecutive tasks, when a task q_l^{h+1} is selected, we set the reward as the time gap (i.e. crane idle time) between the completion time of the previous task q_l^h and the start time of the current task q_l^{h+1}, i.e. $T(q_l^{h+1})^{ship} - T(q_l^h)^{ship} - t(q_l^h)^{ship}$, where $T(q_l^{h+1})^{ship}$ and $T(q_l^h)^{ship}$ are the operation start times of task q_l^{h+1} and q_l^h, $t(q_l^h)^{ship}$ is the task q_l^h's operation time at the QC. Since this problem is a minimisation problem and DRL normally aims to maximise the accumulative reward, we chose the negative time gap as the reward in order to minimise the accumulated QC idle time between tasks.

Note that when computing the reward of a specific task assignment,

the corresponding QC waiting time cannot be computed immediately after the assignment because the previous task may not have been completed yet, or the current truck has not reached the assigned QC. Therefore, the evaluation is done episodically. That is, at each episode, when all the tasks in a given data set are dispatched and finished (i.e. an episode is finished), the rewards are calculated retrospectively.

It should be noted that the agent may choose an action that violates some of the constraints (e.g. going to the work queue without tasks). If this happens, the agent heavily makes the decision extremely unpopular, namely by giving a big negative reward.

3.2.4 The Proposed DRL Algorithm

Notably, in a DRL framework, the training data including the states, actions and rewards, is generated during the interactions between the agent and the dispatching system model. Thus, the difficulty in building the training data or the issues of low-quality labels in supervised learning can be avoided.

The proposed DRL-based method capitalises on both the problem mathematical model of the dispatching system (via *model-derived solution states*) as domain knowledge and the large amount of training data possibly available from previous real-life experience and/or simulations. The mathematical model provides the main structures and properties of the problem while the uncertainties are not modelled mathematically. Instead, it is assumed that all the information of uncertainties is implicitly given in the form of the training data and the proposed algorithm is expected to perform well across all uncertain scenarios. Thus, this framework is believed to be relatively easier to train than the previous data-driven methods due to the use of additional information as pre-knowledge from the mathematical

model. The details of the training process are shown in Algorithm 5.

Algorithm 5 DDQN for an online container truck routing problem

Initialise parameters of both the target Q-network and the current Q-network with random values $\boldsymbol{\theta^-}$ and $\boldsymbol{\theta}$
Initialise the experience replay buffer $\boldsymbol{M}$ with capacity $\boldsymbol{L}$
for each episode **do**
    obtain a state $\boldsymbol{s_a}$ from the problem model directly
    calculate $\boldsymbol{s_b}$ according to the problem model
    concatenate $\boldsymbol{s_a}$ and $\boldsymbol{s_b}$ to generate state $\boldsymbol{s}$
    for each step in the episode **do**
        choose an action $\boldsymbol{a}$ (a work queue) according to ε-greedy policy
        take the action $\boldsymbol{a}$ and acquire a reward $\boldsymbol{r}$ and next state $\boldsymbol{s'}$ which also includes $\boldsymbol{s'_a}$, $\boldsymbol{s'_b}$
        store transition $(\boldsymbol{s, a, r, s'})$ in the experience replay buffer, $\boldsymbol{M}$
        $\boldsymbol{s} \leftarrow \boldsymbol{s'}$
        $\boldsymbol{b} \leftarrow$ a randomly sampled mini-batch of transitions from the experience replay buffer, $\boldsymbol{M}$
        for each transition $(\boldsymbol{s_j, a_j, r_j, s'_j})$ in $\boldsymbol{b}$ **do**
            if $\boldsymbol{s'_j}$ is a terminal state **then**
                $y_j = r_j$
            else
                $y_j = r_j + \gamma Q(s'_j, \arg\max_a Q(s'_j, a; \theta); \theta^-)$
            end if
            for the loss function $L(\theta) = (y_j - Q(s_j, a; \theta))^2$, utilize gradient decent to update parameters θ
        end for
    end for
end for

In this research, during the interactions with the problem model with random parameters, at each time step t, the DRL agent acquires an explicit partial solution state s_a from the problem model directly; then the model derived solution state s_b is calculated pursuant to the dynamics of the problem model, as discussed in subsection 3.2.1. After that, the state s is denoted as the concatenation of s_a and s_b. In accordance with the state, the DRL agent takes an action a and obtains a reward r. Meanwhile, the action takes the partial solution to a new state s', which contains s'_a and s'_b, as mentioned above. In this case, a piece of data e is obtained, indicated as a tuple (s, a, r, s'). At each time step, e is added to the experience

replay buffer M. During the training process, the experience replay mechanism is applied. Every time, a mini-batch of training data (a random set of experiences) is sampled from the experience replay buffer M.

3.3 Experiment

To evaluate the effectiveness of the introduced approach, in this section, we conduct experiments to solve the real-world problem described in section 2.2. Firstly, we introduce the training datasets and the details of the utilised deep neural networks (DNN). Then more details, such as the software and hardware of the experimental setting, are given to provide a better understanding. Chen et al. (2016) first modelled this problem as a dynamic combinatorial optimisation problem and proposed an efficient manually crafted heuristic (MCH), which we used as a baseline to evaluate the performance of the proposed method.

3.3.1 MCH

As mentioned earlier, in this formulation of the problem, a new dispatching request should be invoked each time a truck is idle. The truck will be dispatched to the most suitable crane based on a utility function that incorporates heuristics. The truck will then execute the first task of the work queue related to the most suitable crane. MCH is derived from the accumulated work experience of experienced operators at container terminals, which Chen et al. (2016) has transformed into a heuristic. In this algorithm, some user parameters are applied, as shown in Table 3.1. It should be noted that the parameters *QC.taskNum*, *d(QC.f_tsk_source, truck)*, *QC.p_num_trks* and *QC.p_num_tsk* correspond to the state Features 1, 2, 3 and 4 introduced in Section 3.2.1. Considering the operational requirements of the real container terminal, each QC is assigned a specific priority value, namely *QC.priority*

in the MCH, but this feature was not considered in the DRL framework.

Table 3.1: User parameters of the MCH.

User parameters	**Description**
QC.p_num_trks	The predicted number of trucks to serve the QC, including trucks already arrived and to be arrived in the near future.
QC.p_num_tsk	The predicted number of tasks to be finished at the QC in the near future.
QC.f_tsk_source	The source node of the first task of the QC.
d(QC.f_tsk_source, truck)	The distance between the current position of the truck and QC.f_tsk_source.
QC.taskNum	The remaining number of tasks the QC needs to finish.
QC.a_num_trks	The available number of trucks that have already arrived at the QC.
QC.max_num_limit	The maximum allowable number of assigned trucks to any QC.
QC.priority	A priority related to the QC.

When the algorithm is running, real-time parameters will be used to calculate a specific score for each QC. Based on the scores for each QC, the algorithm will decide whether to dispatch the idle truck to a QC, as shown in Algorithm 6. Line 4 of Algorithm 6 invokes the utility, function which is detailed in Algorithm 7. The *QC.tasks[0]* in line 7 refers to the first task of the work queue related to the QC.

3.3.2 Dataset

The experiment datasets were drawn from real-world problems with a small adaptation. Two datasets (Small_Basic and Big_Basic) were used in the initial experiments, both containing 120 problem instances. In the scalability test experiments, 10 further datasets were generated based on Small_Basic and Big_Basic. In all problem instances, the crane operation times $(t(i)^{ship}, t(i)^{yard})$ are drawn from four different Gaussian distributions

Algorithm 6 The MCH proposed by Chen et al. (2016)

Require: Truck *truck*, QC *QCs*, Heuristic *heuristic*

function *dispatch*(*truck*)
    $min\_score \leftarrow +\infty$, $dispatch\_task \leftarrow NULL$
    for *QC* in *QCs* **do**
        $QC.score \leftarrow heuristic(QC, truck)$
        if $QC.score < min\_score$ **then**
            $min\_score \leftarrow QC.score$
            $dispatch\_task \leftarrow QC.tasks[0]$
        end if
    end for
    return *dispatch_task*
end function

Algorithm 7 The utility function of MCH

Require: Truck *truck*, User Parameters *parameters*

function *heuristic*(*QC, truck*)
    if $QC.a\_num\_trks < QC.p\_num\_tsk$ **then**
        $score \leftarrow d(QC.f\_tsk\_source, truck) * (QC.a\_num\_trks - QC.priority)$
    else
        $score \leftarrow d(QC.f\_tsk\_source, truck) * QC.p\_num\_tsk$
    end if
    if $QC.p\_num\_trks \geq QC.max\_num\_limit$ **then**
        $score \leftarrow score + 200000$
    end if
    if $QC.taskNum \leq 0$ **then**
        $score \leftarrow score + 500000$
    end if
    return *score*
end function

to sufficiently simulate the complexity of the real-life data. Their real values are revealed dynamically over time. Table 3.2 shows more details about the datasets. Basically, they are divided into two groups: Small and Big datasets, and are sorted in descending order according to the number of tasks per instance (or the number of QCs) respectively.

Table 3.2: Details of datasets used in various experiments.

Datasets	no. of tasks per instance	no. of QCs
Small96T	96	4
Small_Basic	72	4
Small48T	48	4
Small3WQ	54	3
Small2WQ	36	2
Big288T	288	6
Big_Basic	216	6
Big144T	144	6
Big5WQ	180	5
Big4WQ	144	4
Big3WQ	108	3
Big2WQ	72	2

Each problem instance in the 'Small_Basic' task dataset contained 72 tasks which are distributed in 4 QCs, while every problem instance in the 'Big_Basic' task dataset contained 216 tasks which are distributed in 6 QCs. In the 120 problem instances of both 'Small_Basic' and 'Big_Basic' task sets, 20 of them are randomly sampled for training and the remaining 100 for the test.

We then regenerate new tasks for problem instances in all the other task sets, but the operation nodes involved are the same as those in 'Small_Basic', and 'Big_Basic' task sets. The number of tasks of a problem instance in 'Small48T' and 'Small96T' changed to 48 and 96 from 72 (in 'Small_Basic'), respectively. With similar changes, 'Big144T' and 'Big288T' are generated.

For problem instances in 'Small3WQ', 'Small2WQ', 'Big5WQ', 'Big4WQ',

'Big3WQ' and 'Big2WQ' task sets, the numbers of work queues involved were changed. For the 'Small_Basic' task sets, the numbers of work queues were changed from 4 to 3 and 2. For the 'Big_Basic' task sets, the numbers of work queues were changed from 6 to 5, 4, 3 and 2.

As mentioned at the end of Section 2.2, the service times of the tasks are uncertain. In this port, the service time of tasks obeys normal distributions, and their average value is 120 seconds. To reflect the various situations in the real world as much as possible, we chose 4 different distributions ($\mathcal{N}(120, 5^2)$, $\mathcal{N}(120, 10^2)$, $\mathcal{N}(120, 15^2)$ and $\mathcal{N}(120, 20^2)$, respectively) to generate the service times of the tasks for the problem instances in all the task sets. The truncated ranges of them are [105, 135], [90, 150], [75, 165] and [60, 180] seconds, respectively.

3.3.3 Details of the DNNs

We adopted a four-layer DRL framework similar to the one used in Chen & Tian (2019), where a DRL was used to solve some deterministic combinatorial optimisation problems. In the three problems they solved, their neural networks include one, two, and two hidden layers, respectively.

For our implementation, the numbers of neurons in every layer in the two neural networks (NN) are shown in Table 4.4. The inputs of the NNs are the 'states'. Before feeding them to the NNs, the features in the states are normalised to the range [-50, 50]. The numbers of dimensions of them are related to the number of QCs (or work queues). Therefore, the number of neurons in the input layers of the two NNs are different. Since the outputs of the deep neural networks are the values $Q(s, a_i)$ $i \in (0, 1, 2, \cdots, n_{ship} - 1)$ in a specific state s, the dimension of the output layer is equal to the number of actions: for 'Small_Basic' and 'Big_Basic', they are 4 and 6, respectively.

We used the rectified linear unit function (ReLU) as the activation

Table 3.3: The number of neurons in our DRL for different datasets.

	Layer 1 (Input)	Layer 2	Layer 3	Layer 4 (Output)
small sets	16	100	160	4
big sets	24	100	160	6

function for each hidden layer. Since the NNs approximate real-valued future rewards, there is no activation function on the output layer, and we did not use one-hot encoding, either.

For ε-greedy policy used to choose actions (see line 8 in algorithm 5, ε was set to 0.9 in the training process and was set to 1 during the test process. The reward decay γ shown in line 17 in algorithm 5 equaled to 0.8. The size of the data pool $\mathbf{M}$ (in line 2 in algorithm 5) was 3000 and the size of the mini-batches (in line 12 in algorithm 5) was set to 60. i.e. for every training step, 60 transition experiences (s_j, a_j, r_j, s'_j) were fed to the neural network of the DRL agent to train the model.

The loss function is shown in line 19 algorithm 5. We chose the RMSprop optimiser with the learning rate equalled to 0.001 to minimise the loss.

Other settings related to the software and hardware are shown in Table 3.4.

Table 3.4: Hardware and software environment of experiments

Name	Detailed configuration
Processor	Intel Core i7-8700K
Memory	48GB
Graphics	GeForce GTX 1080 Ti
Linux	Ubuntu 16.04 LTS
Python	2.7.12
Tensorflow	1.0.0

3.4 Results Analysis

Considering there are stochastic components in the proposed algorithm, such as the initial values of the weights, biases in the neural networks, all the experiments (i.e. both training and testing) are repeated 10 times. During the training, our DRL agent converges after around 8000 episodes on both Small_Basic and Big_Basic. In each episode, the entire training dataset (20 problem instances) was used to train the DNNs. In the experiments, when the agent selects an action that violates some of the constraints, such as going to a work queue without tasks, we give it rewards of -10000 and -20000 which are far smaller than a normal reward signal, for Small_Basic and Big_Basic datasets, respectively.

When the training is completed, we evaluated the well-trained DRL agent for its performance and scalability (which includes two directions). In each test experiment, the algorithms were used to solve 100 test instances. For each test instance, the algorithms were run 200 times with different random seeds.

The average total crane waiting time over 100 problem instances was adopted as the performance indicator. We compared the results obtained from a manually crafted heuristic (MCH) (Chen et al., 2016) and the proposed DRL method, as shown in Table 3.5. It can be seen that the proposed DRL obtained better results than MCH on average. We also conducted a Wilcoxon Rank Sum test on the results obtained from MCH and the proposed DRL in this experiment and the following two groups of scalability experiments (see Section 3.4.1). All the obtained p-values are less than 0.01, indicating statistically significant differences between the results obtained from the two methods.

To further explore whether the proposed DRL performed well on all

Table 3.5: The Performance of the proposed DRL in comparison with the manual crafted heuristic (MCH). In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev		Imp% of the proposed DRL over MCH
	MCH (baseline)	the proposed DRL	
Small_Basic	1808.02/15.72	1665.24/10.29	7.91%
Big_Basic	6392.29/56.78	5890.38/60.33	7.85%

the test instances, we compared the total crane waiting time of all 100 test instances obtained from both methods. We put the results on two figures (Figure 3.2 and Figure 3.3) to present them more clearly. As can be seen from the figures, in 76 of 100 test instances, the proposed DRL performed better than MCH.

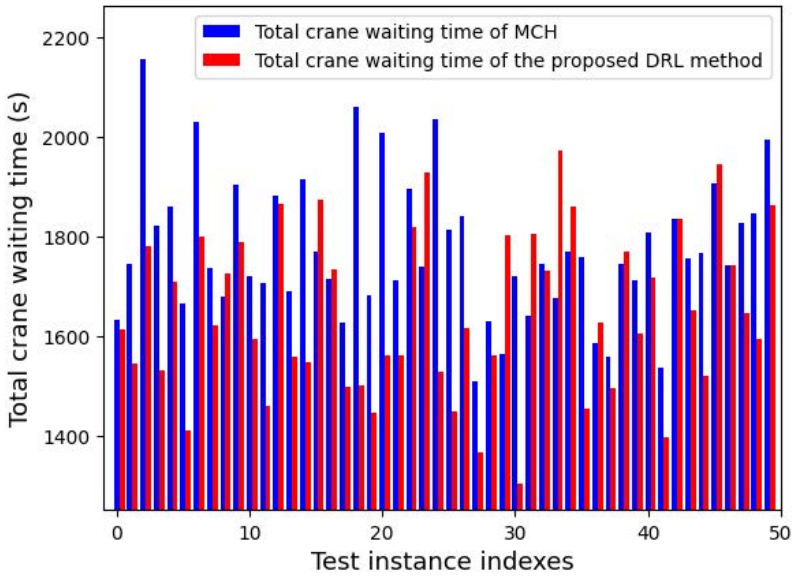

Figure 3.2: *Total crane waiting time of the two methods when tested on different test instances (a)*

3.4.1 Scalability Evaluations

In real-world scenarios, the generality of the trained model is of high importance. In the problem faced by Ningbo Port, the number of tasks in

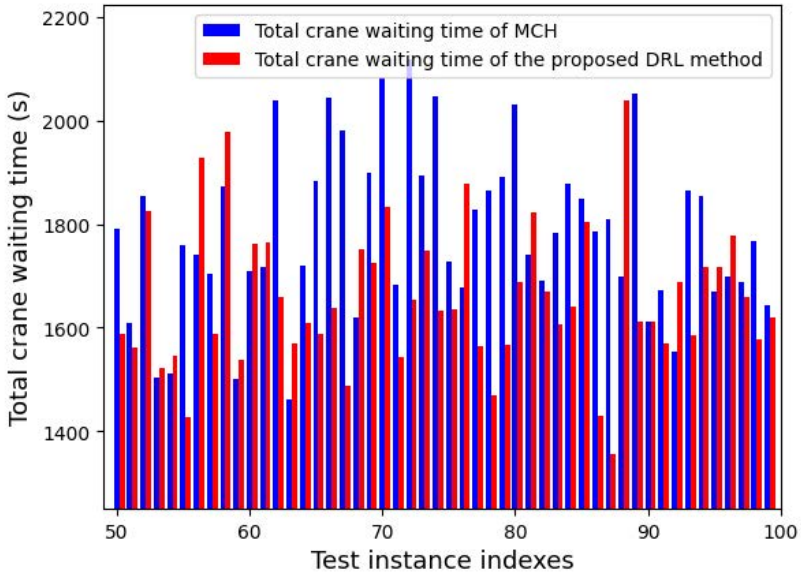

Figure 3.3: *Total crane waiting time of the two methods when tested on different test instances (b)*

the datasets and the number of work queues (which is equal to the number of QCs) may change at some point. Therefore, two groups of experiments (Scalability Experiment 1 and Scalability Experiment 2) were conducted to evaluate the generality of the trained DRL agent.

In Scalability Experiment 1, the trained DRL models were tested on 'Small48T', 'Small96T', 'Big144T' and 'Big288T', where the number of tasks in a problem instance is different from those of the training instances. The experimental results are presented in Table 3.6.

Table 3.6: Results of scalability experiment 1. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev		Imp% of the proposed DRL over MCH
	MCH (baseline)	the proposed DRL	
Small48T	1017.06/7.52	965.32/10.37	5.09%
Small96T	2207.33/16.77	2136.52/17.17	3.21%
Big144T	3601.26/22.77	3446.76/31.44	4.30%
Big288T	8489.78/45.56	8190.73/74.59	3.52%

In Scalability Experiment 2, the trained DRL models were tested on

'Small3WQ', 'Small2WQ', 'Big5WQ', 'Big4WQ', 'Big3WQ' and 'Big2WQ', where the number of work queues involved is decreased. The results are shown in Table 3.7. Note that here we did not test our trained algorithm on instances with increased QCs due to two reasons: First, we foresee poor results because the DRL agent knows nothing about these newly added QCs during the training. Second, for a real-world port, the maximum number of QCs stays unchanged. If we train our DRL-HH with the maximum QCs, the model would still perform well for instances with fewer QCs because it has seen all the information.

Table 3.7: Results of scalability experiment 2. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev		Imp% of the proposed DRL over MCH
	MCH (baseline)	the proposed DRL	
Small3WQ	1360.45/8.47	1308.44/13.16	3.82%
Small2WQ	895.43/5.36	866.05/9.52	3.28%
Big5WQ	4850.87/35.68	4706.91/54.96	2.97%
Big4WQ	3920.40/29.45	3750.24/37.65	4.34%
Big3WQ	2620.34/21.48	2510.72/25.62	4.18%
Big2WQ	1745.55/14.73	1689.22/19.70	3.23%

In these two groups of scalability experiments, the changes in both the number of tasks in the datasets and the number of work queues render significant differences in the problem structure. Hence, obtaining a model that can perform well in different situations is considerably difficult. However, the results of the proposed DRL method are still better than MCH in both of the two groups of scalability experiments.

Although the proposed DRL method does not seem to improve the results significantly, it is actually an important improvement in a real-world port. There are two reasons for this. First, the time required to complete the problem instances (1600 to about 1800 seconds) in our experiments is very short. If we zoom in the time period to 1 year, then the time saved

by our algorithm will be 647 hours compared with the time spent by MCH. Second, the number of QCs involved in our datasets is small compared to the total number of QCs in an international port like Ningbo Port. If we train an agent that includes all QCs in the port in the future, even if the total QC waiting time is reduced by about 7%, it will bring great economic benefits. It is very time-consuming to train a big agent. Therefore, the scalability experiment in this research only tests the scalability at a moderate level.

In a DRL framework, since the DNN can learn non-linear relationships and encode kinds of knowledge in different problem instances, good performance can be achieved by consuming only a certain amount of computing power and training time. In contrast, when using a heuristic to solve complex combinatorial optimisation problems, it takes a lot of expertise and time to develop a well-performing heuristic. Moreover, such a heuristic is usually problem-specific and cannot be well generalised.

3.4.2 Computational Times

For an online algorithm, in addition to performance, we are also concerned about its computation time. The proposed DRL method spent 324 minutes (7740 episodes) and 1640 minutes (8864 episodes) training the DRL agent for the Small_Basic and Big_Basic task sets, respectively.

It is worth noting that although it takes a lot of time to train a model, it is very fast in testing/execution once trained. The average execution time for a trained DRL agent to solve a small instance (containing 72 tasks) is 0.159 seconds and 0.536 seconds for a large instance (with 216 tasks). Therefore, in terms of computing time, the proposed method is adequate for solving the online problem and can be applied to the operation of real-world ports.

3.5 Summary

This chapter presented a conventional DRL method for solving online combinatorial optimisation problems and evaluated it by solving a real-world online container truck routing problem. Some important problem parameters, such as the crane service time in the online container truck routing problem, are uncertain and cannot be obtained in advance. Therefore, online algorithms need to be developed to solve these problems. The proposed method has three advantages over the previous manually crafted heuristic method.

- First, the DRL method has an inherent advantage in solving sequential decision problems: its trial-and-error learning mechanism and maximisation of cumulative rewards allow the DRL agent to improve its capabilities as it interacts with its environment automatically. Since online combinatorial optimisation problems are essentially sequential decision problems, the DRL is an appropriate choice for solving them.

- Secondly, DRL can learn in dynamic environments (data with uncertainties). The DNN in the DRL framework can learn non-linear relationships and encode kinds of knowledge in different problem instances.

- Thirdly, unlike developing a manually crafted heuristic, the proposed method does not require much problem-specific knowledge and a lot of manpower and time to polish itself in order to improve its performance.

We can draw the following conclusions from the computational evidence. Firstly, the proposed method outperforms the manually crafted heuristic on datasets containing 100 problem instances of different sizes.

This is probably because the proposed algorithm makes more balanced decisions jointly considering both short-term rewards and forecasts of discounted future gains (parameterised by the discount rate) through the deep Q-networks, while the manually crafted heuristic relies on fixed rules that are derived from experience with the most common cases but may suffer performance drop for less popular cases perceived by scheduling experts. Secondly, the scalability experiments show that the proposed method is better than the manually crafted heuristic when the number of tasks in the problem instances or the number of work queues changes. Finally, after taking a certain amount of computational power and time for training, the obtained model can respond in a very short time in practice, which can be fully adapted to the online port scenario.

Despite its high performance and good generality, the proposed method suffers from poor interpretability of the solution as a machine learning algorithm, which hinders its application in practice. Meanwhile, the training time is relatively long. In the next chapter, we introduce a new method that improves the interpretability of the solutions and reduces the training time, too.

Chapter 4

A DRL-Based Hyper-heuristic for Online Container Truck Routing Problem with Uncertainties

4.1 Introduction

In most real-life COPs, accurate information of all the problem characteristics cannot be acquired in advance. Instead, the real values of problem parameters are often sequentially revealed over time during decision making. Thus, it is crucial to develop alternative methodologies that can accommodate uncertainties as well as make decisions that can sufficiently balance the conflicts between solution optimality and its resilience to unpredictable (sometimes even disruptive) changes.

This research focuses on adaptive algorithms that are trained offline but deployed in real-time, so that decisions are dynamically made in sequence pursuant to the actual situations revealed. This provides decision-makers with the maximum flexibility to react to changes while maintaining the high

quality from the resulting solution.

This work is also motivated by the demand of advanced algorithms for solving challenging online COPs, taking advantage of both the known problem structures as well as a large amount of unlabelled historical data reflecting the uncertainties. To be more adaptable to industrial applications, the proposed algorithms must also cater to a certain level of *interpretability*. Bearing these requirements in mind, we propose a new hyper-heuristic method that uses a double deep Q-network (DDQN) (Van Hasselt et al., 2016) to train a heuristic selection module from a set of low-level, human-interpretable heuristics in different problem-solving scenarios.

Different from the DRL methods in Mnih et al. (2015) and Van Hasselt et al. (2016), which are designed to solve gaming problems without obvious mathematical formulations, our DRL-based hyper-heuristic (DRL-HH) is applied to classical COPs which have rich literature, especially for their offline version of the problems. Meanwhile, compared with previous hyper-heuristics, the simple Q-learning mechanism is replaced with a much more powerful DDQN, which provides much better ability to handle high dimensionality data. Besides, compared to online genetic programming (GP) hyper-heuristic methods (e.g. MacLachlan et al. (2019) and Chen et al. (2020)), the proposed method could offer better performance in handling a large amount of training data with much higher dimensionality of features, on which the evolution of a GP decision tree is often highly challenging (if not impossible) (as analised in Section 2.7.1). At last, compared to supervised learning methods heavily used in data predictions and pattern recognition, the proposed method does not require pre-labelled data by experts. In fact, most of the problem instances addressed in this paper do not normally have ground-truth solutions due to their difficulty. The decisions made in practice are not optimal because of the problem complexity.

The remainder of this chapter is organised as follows: Section 4.2 describes the proposed DRL hyper-heuristic framework. In Section 4.3, the proposed framework is evaluated by solving the same online container truck routing problem with uncertainties described in Section 2.2, followed with discussions of the experimental design and results analysis. Finally, Section 4.4 concludes this chapter.

4.2 The Proposed DRL Hyper-heuristic Framework

According to the classification introduced in Section 2.5.1, the proposed DRL-based hyper-heuristic belongs to the offline-learning constructive selection hyper-heuristics. As depicted in Figure 4.1, selection constructive hyper-heuristics operate upon a set of constructive low-level heuristics to incrementally build solutions. A constructive heuristic can be certain rules, suitable building blocks/patterns and partial optimal solutions premised on certain mathematical models. There can also be some random assignments in certain cases to diversify the search. A key decision in constructive hyper-heuristics is to intelligently select the most suitable constructive heuristic(s) in the heuristic space at each step of solution construction while at the same time satisfying various constraints. The ultimate goal is to acquire an optimal (or near-optimal) sequence of constructive heuristics that builds a high-quality solution in the solution space. The challenge is to build a reliable mapping between the problem states and constructive heuristics. The interface between the solution space and hyper-heuristics renders the possibility of building a system that performs effectively across various solution spaces.

In the proposed DRL-HH framework, a DRL is introduced into an ex-

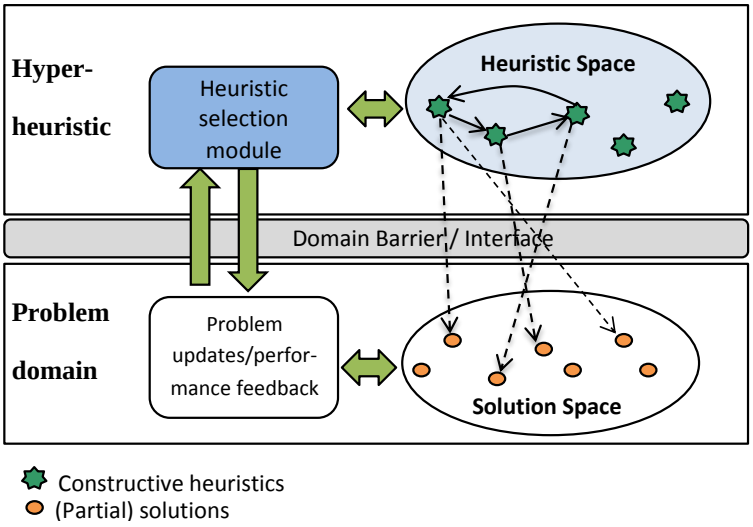

Figure 4.1: *A selection constructive hyper-heuristic framework*

isting selection constructive hyper-heuristic framework. As in the previous chapter, we also adopted the widely used DDQN as the DRL algorithm to train the present heuristic selection module exhibited in Figure 4.1. Details are described in the following sub-sections. As discussed in last chapter, DRL has good perception and the ability to automatically enhance policies based on its trial-and-error learning. Compared with some greedy and myopic online algorithms, such as `best-fit` for online bin-packing and `nearest neighbour` heuristic for TSP, the proposed method can strategically give up some current rewards to obtain a bigger reward in the future thanks to the intelligence built in the DRL agent through offline training based on the large amount of data containing hidden information regarding uncertainties. It can, therefore, more effectively handle problems with uncertainties.

4.2.1 A DRL-based Hyper-heuristic

The proposed DRL-based constructive hyper-heuristic framework is depicted in Figure 4.2. In the framework, the actions are a set of parameterised heuristics (often referred to as low-level heuristics in hyper-heuristics). The selection of these actions/heuristics is based on two state-vectors, namely

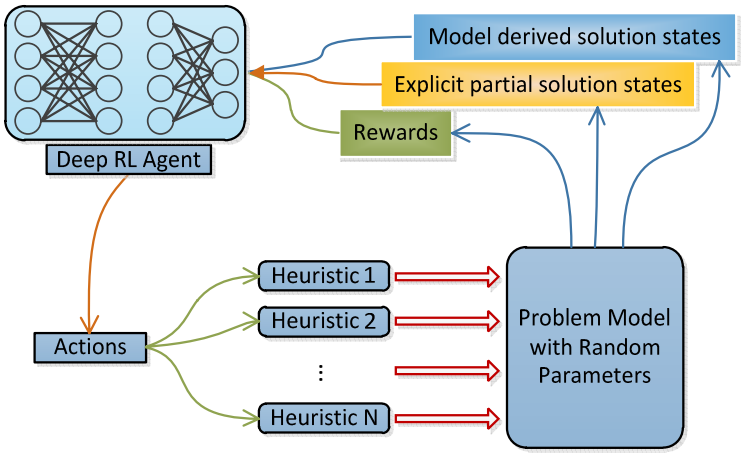

Figure 4.2: *The DRL hyper-heuristic framework*

the *explicit partial solution states* and *model-derived solution states*, which are the same as that introduced in section 3.2.1 and based on the historical experience of the DRL agent.

The DRL agent is represented as a value function $Q(s, a)$ with respect to state s and action a, and corresponds to the heuristic selection module in the constructive hyper-heuristic. At each decision point, the agent selects and then executes an action in accordance with the states of the partial solution, and acquires reward feedback from the problem model. The Q function is defined as the expectation of discounted cumulative rewards, as denoted in Equation 4.1, where γ is the discounted factor, t is the time step, R is the reward and π is the policy.

$$Q^{\pi}(s, a) = \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, A_t = a, \pi] \tag{4.1}$$

The proposed DRL-based constructive hyper-heuristic method also makes use of the mathematical problem model (via model derived solution states) as domain knowledge. The mathematical model only simulates the main structures and properties of the problem, while the uncertainty of the environment is then simulated by the diversity of data. Decoupling the design of the mathematical model from the uncertainty of the data has two ad-

vantages: firstly, it makes the mathematical model construction simpler; secondly, the diversity of data can be produced independently without considering the interaction between the data and the mathematical model. Also, as the framework uses additional information from the mathematical model, it can be trained more efficiently than the previous data-driven methods.

The offline training of the framework is more or less the same as the training of the DDQN in last chapter (as shown in Figure 3.1). In this study, during the interactions with the problem model with random parameters, at each time step t, the DRL agent acquires an explicit partial solution state s_a from the problem model directly; then the model derived solution state s_b is calculated pursuant to the dynamics of the problem model. After that, the state s is denoted as the concatenation of s_a and s_b. In accordance with the state, the DRL agent takes an action a (a low-level heuristic) and obtains a reward r. Meanwhile, the action takes the partial solution to a new state s', which contains s'_a and s'_b, as mentioned above. In this case, a piece of data e is obtained, indicated as a tuple (s, a, r, s'). At each time step, e is added into the data pool M. During the training process, the experience replay mechanism is applied. Every time, a mini-batch of training data (a random set of experiences) is sampled from the data pool M. The details of the training process are shown in Algorithm 8.

4.2.2 Solution State of the Hyper-heuristic

The proposed DRL hyper-heuristic constructs a solution step by step by repeatedly calling a chosen constructive heuristic. At each step, two state vectors on the current partial solution are passed to the hyper-heuristic as reference information for decision making (i.e. choice of the most appropriate constructive heuristic from the set of heuristics at the disposal thereof).

Algorithm 8 DDQN hyper-heuristic

Initialise parameters of both the target network and online network with random values $\boldsymbol{\theta}$
Initialise data pool $\boldsymbol{M}$ with capacity $\boldsymbol{L}$
for each episode **do**
    obtain a state $\boldsymbol{s_a}$ from the problem model directly
    calculate $\boldsymbol{s_b}$ according to the problem model
    concatenate $\boldsymbol{s_a}$ and $\boldsymbol{s_b}$ to generate $\boldsymbol{s}$
    for each step in the episode **do**
        choose agent action $\boldsymbol{a}$ (a low-level heuristic) according to ε-greedy policy
        calculate a heuristic action $\boldsymbol{A_{heu}}$ by the chosen low-level heuristic
        take the heuristic action $\boldsymbol{A_{heu}}$,
        acquire a reward $\boldsymbol{r}$ and next state $\boldsymbol{s'}$ which also includes $\boldsymbol{s'_a}$, $\boldsymbol{s'_b}$
        store transition $(\boldsymbol{s, a, r, s'})$ in the data pool, $\boldsymbol{M}$
        $\boldsymbol{s} \leftarrow \boldsymbol{s'}$
        $\boldsymbol{b} \leftarrow$ a randomly sampled mini-batch of transitions from the data pool, $\boldsymbol{M}$
        for each transition $(\boldsymbol{s_j, a_j, r_j, s'_j})$ in $\boldsymbol{b}$ **do**
            if $\boldsymbol{s'_j}$ is a terminal state **then**
                $y_j = r_j$
            else
                $y_j = r_j + \gamma Q(s'_j, \arg\max_a Q(s'_j, a; \theta); \theta^-)$
            end if
            for the loss function $L(\theta) = (y_j - Q(s_j, a; \theta))^2$, utilize gradient decent to update parameters θ
        end for
    end for
end for

The first state vector (the explicit partial solution states) contains all the necessary information about the current partial solution, including any constraints and efficiency indicators of key resources of the problem under concern. The second state vector (the model derived solution states) is on the projected solution states at any future point, being estimated through the deterministic model of the problem. In an effort to raise the generality of hyper-heuristics, the use of both explicit state vector and model-derived state vector are considered as a distinctive algorithm design philosophy.

4.2.3 Actions of the Hyper-heuristic

In the context of hyper-heuristics, the actions are, in most cases, various heuristic rules used in practice. The actions can also be more sophisticated model-based strategies for making multiple decisions at each step. Action set design is problem-specific, and there is no generic design suitable for all problems. As a general guideline, an action set design should satisfy both reachability and interpretability. Reachability means that there should exist at least one combination of these heuristics through which the optimal solution can be reached. Meanwhile, interpretability requires a certain level of convenience to interpret and evaluate these heuristics.

In the following section, we demonstrate how the proposed hyper-heuristic method can be used to solve the online container truck routing problem with uncertainties proposed in Section 2.2 and evaluate its performance in comparison with some existing methods.

4.2.4 Implementation Details of the New Framework

This section describes the implementation details of the proposed DDQN-based hyper-heuristic method, including the state design, the action sets (low-level heuristics) and the reward design.

4.2.4.1 State Design

In order to maintain a certain level of consistency among different approaches for the purpose of performance comparison in experiments, the same state as introduced in Section 3.2.1 is employed in this method. It can be represented as:
$[[rn_1, rn_2, \cdots, rn_m], [[d_1, d_2, \cdots, d_m], [wn_1, wn_2, \cdots, wn_m], [pn_1, pn_2, \cdots, pn_m]]$, which is a concatenation of the four features (Feature 1 to 4) in the state in Section 3.2.1.

4.2.4.2 Actions

In this proposed framework, there are two level of actions. An agent performs an *agent action* to select a low-level heuristic, i.e. $A_{agent} = \{agent\ action \quad i \mid i \in (0, 1, 2, \cdots, 9)\}$. Once agent action i is selected, the corresponding *heuristic action* is taken to assign a task to the current truck under consideration.

The set of low-level heuristics used in this paper is inspired by the manual heuristic in Chen et al. (2016). Essentially, these heuristics are different rules to sort the active work queues ($WQ_l, l \in N^{ship}$) attached to QCs. This is because the task sequencing in each work queue is already decided separately and is not part of optimisation in our problem. The low-level heuristics considered three factors: distances between the current action-triggering truck and the candidate work queues, the degree of unbalance of work queues, and the degree of urgency of all candidate work queues. For each factor, three possible thresholds are assigned to decide whether the corresponding factor becomes active in work queue sorting. This leads to $3 * 3 = 9$ low-level heuristics. Finally, Chen et al. (2016)'s manual heuristic is also included as a low-level heuristic in our DRL-HH.

4.2.4.3 Low-level Heuristics

The low-level heuristics used in this study are inspired by the manual heuristic proposed in Chen et al. (2016), which considers three factors:

- the distance between the current position of the truck and the source node of the first task of a particular work queue,
- the *unbalanced degree* which is defined as the difference in the remaining tasks between the work queue under consideration and the shortest work queue (i.e. with the least number of remaining tasks).

- the *urgent degree* which means the difference between the demand and supply of a work queue. Specifically, the demand refers to the predicted number of tasks to be finished at a QC in the near future, and the supply refers to the number of these tasks that have already been dispatched by or assigned to some trucks within the same period.

In this research, we use a set of thresholds for each of the three factors to become active in the decision making. These different thresholds are applied to different low-level heuristics so that different low-level heuristics have different sensitivities to the above-mentioned factors. Then the DRL-HH can choose a low-level heuristic with a good threshold at each step. Since the changing ranges of these three factors are different, we normalised them to $[0, 1]$. For each of the three configurable parameters, we set three values $\{0.3, 0.6, 0.9\}$ for it. Each low-level heuristic only considers one factor, and a corresponding threshold is given as input data. Therefore, there are a total of 9 low-level heuristics in addition to the one proposed by Chen et al. (2016) as depicted in Algorithm 7.

The first group of low-level heuristics are distance-first heuristics exhibited in Algorithm 9. This heuristic prefers to choose the first task of a particular work queue if the current position of the truck dispatched to finish the said task is the closest to the source node of said task. When there are multiple tasks with the distances between the truck's current position and their source nodes being smaller than the given distance threshold d_t, this heuristic operates under the assumption that these tasks are all considerably close and randomly chooses between these close tasks.

The second group of heuristics are urgent degree-first heuristics presented in Algorithm 10. This heuristic tends to choose the first task of the work queue with the biggest urgent degree. Yet, if there are multiple work queues of which urgent degrees are greater than the given urgent de-

Algorithm 9 Distance-first heuristic

Input: Truck $\boldsymbol{v}$, a distance threshold $\boldsymbol{d_t}$
Initialise the distance list $\boldsymbol{dlist}$ with an empty list
for each work queue $\boldsymbol{wq_i}$ **do**
    calculate the distance $\boldsymbol{d_i}$ between the current position of the truck $\boldsymbol{v}$ and the source node of the first task of $\boldsymbol{wq_i}$
    append $\boldsymbol{d_i}$ to $\boldsymbol{dlist}$
end for
Initialise list $\boldsymbol{dl}$ with an empty list
for each distance $\boldsymbol{d_i}$ in $\boldsymbol{dlist}$ **do**
    if $\boldsymbol{d_i} < \boldsymbol{d_t}$ **then**
        append i in $\boldsymbol{dl}$
    end if
end for
if $\boldsymbol{dl}$ is not empty **then**
    select an element $\boldsymbol{dl}[\boldsymbol{i}]$ from $\boldsymbol{dl}$ randomly.
    $\boldsymbol{k} = \boldsymbol{dl}[\boldsymbol{i}]$
else
    $\boldsymbol{k}$ = the index of the smallest element of $\boldsymbol{dlist}$
end if
return the first task of work queue $\boldsymbol{wq_k}$

gree threshold u_t, this heuristic operates under the assumption that these work queues are all considerably urgent. Further, the first tasks in these considerably urgent work queues are randomly chosen.

The third group of heuristics are unbalanced degree-first heuristics denoted in Algorithm 11. This heuristic first tends to choose the first task of the work queue with the biggest unbalanced degree. If the unbalanced degrees of multiple work queues are greater than the given unbalanced degree ubd_t, these work queues are all considered as considerably unbalanced, and the first tasks in these considerably unbalanced work queues are randomly chosen.

4.2.4.4 Rewards

The Reward design is the same as that introduced in section 3.2.3, namely the negative time gap (i.e. crane idle time) between the completion time of the previous task q_l^h and the start time of the current task q_l^{h+1}, i.e.

Algorithm 10 Urgent degree-first heuristic

1: **Input**: an urgent degree threshold $\boldsymbol{u_t}$
2: Initialise the urgent degree list $\boldsymbol{ulist}$ with an empty list
3: **for** each work queue,$\boldsymbol{wq_i}$ **do**
4: calculate the predicting demanding number($\boldsymbol{d_n}$) of trucks in 10 minutes
5: calculate the sum ($\boldsymbol{s_i}$) of the number of trucks already lining up in $\boldsymbol{wq_i}$ and the predicted number of trucks which can reach in 10 minutes
6: calculate the urgent degrees $\boldsymbol{u_{d_i}}$ (the difference between $\boldsymbol{d_n}$ and $\boldsymbol{s_i}$)
7: append $\boldsymbol{u_{d_i}}$ to $\boldsymbol{ulist}$
8: **end for**
9: Initialise list $\boldsymbol{ul}$ with an empty list
10: **for** each $\boldsymbol{u_{d_i}}$ in $\boldsymbol{ulist}$ **do**
11: **if** $\boldsymbol{u_{d_i}} > \boldsymbol{u_t}$ **then**
12: append i in $\boldsymbol{ul}$
13: **end if**
14: **end for**
15: **if** $\boldsymbol{ul}$ is not empty **then**
16: select an element $\boldsymbol{ul}[\boldsymbol{i}]$ from $\boldsymbol{ul}$ randomly.
17: $\boldsymbol{k} = \boldsymbol{ul}[\boldsymbol{i}]$
18: **else**
19: $\boldsymbol{k}$ = the index of the biggest element of $\boldsymbol{ulist}$
20: **end if**
21: **return** the first task of work queue $\boldsymbol{wq_k}$

$T(q_l^{h+1})^{ship} - T(q_l^h)^{ship} - t(q_l^h)^{ship}$, where $T(q_l^{h+1})^{ship}$ and $T(q_l^h)^{ship}$ are the operation start times of task q_l^{h+1} and q_l^h, $t(q_l^h)^{ship}$ is the task q_l^h's operation time at the QC.

The difference is that in this framework, there is no need to heavily penalize those actions that violate some of the constraints (e.g. going to work queue without tasks), which the method proposed in the last chapter has done in order to make the agent avoid choosing such harmful actions. This is because the low-level heuristics used in this constructive hyper-heuristic framework can not only reduce the search space and improve the interpretability of the actions, but also they are equivalent to constraints added to the original actions (*heuristic actions*), thus preventing the agent from selecting those extremely bad actions.

Algorithm 11 Unbalanced degree-first heuristic

Input: an unbalanced degree threshold $\boldsymbol{ubd_t}$
Initialise the remaining tasks list $\boldsymbol{rlist}$ with an empty list
Initialise the unbanced degree list $\boldsymbol{ubdlist}$ with an empty list
for each work queue,$\boldsymbol{wq_i}$ **do**
    calculate the number of the remaining tasks, $\boldsymbol{rt_i}$ in this work queue
    append $\boldsymbol{rt_i}$ to $\boldsymbol{rlist}$
end for
$\boldsymbol{rt_{min}}$ = min($\boldsymbol{rlist}$)
for each $\boldsymbol{rt_i}$ in $\boldsymbol{rlist}$ **do**
    append unbanced degree $\boldsymbol{ubd_i}$ = $\boldsymbol{rt_i}$ - $\boldsymbol{rt_{min}}$ to $\boldsymbol{ubdlist}$
end for
Initialise list $\boldsymbol{ubdl}$ with an empty list
for each $\boldsymbol{ubd_i}$ in $\boldsymbol{ubdlist}$ **do**
    if $\boldsymbol{ubd_i} > \boldsymbol{ubd_t}$ **then**
        append i in $\boldsymbol{ubdl}$
    end if
end for
if $\boldsymbol{ubdl}$ is not empty **then**
    select an element $\boldsymbol{ubdl}[\boldsymbol{i}]$ from $\boldsymbol{ubdl}$ randomly.
    $\boldsymbol{k} = \boldsymbol{ubdl}[\boldsymbol{i}]$
else
    $\boldsymbol{k}$ = the index of the biggest element of $\boldsymbol{ubdlist}$
end if
return the first task of work queue $\boldsymbol{wq_k}$

4.3 Experiment Design and Results Analysis

To evaluate the performance and robustness of the proposed DRL hyper- heuristic (DRL-HH), several benchmark problem instances of different sizes are extracted from real-life data to serve as a test bed. The methods to solve this kind of online COP are scarce due to its huge solution space and a relatively low response time required. The manually crafted heuristic (MCH) (Chen et al., 2016) showed to generate solutions that are superior to those used in practice, and is used as a baseline for our proposed method. The proposed DRL-HH is also compared with the data-driven genetic programming hyper-heuristic (Data-driven GP) (Chen et al., 2020).

4.3.1 Data-driven GP

The data-driven GP proposed by Chen et al. (2020) is basically a GP algorithm that evolves programs to dispatch container trucks for an online truck dispatching problem at each decision point. The programs, namely individuals in GP populations, are represented as trees as introduced in Figure 2.6. The terminals and operations are shown in Table 4.1. As introduced before, the *QC.f_tsk_source* in *d(QC.f_tsk_source, truck)* refers to the source node of the first task of the QC.

Table 4.1: The terminals and operations of the GP tree in Data-driven GP.

Name	**Description**
+	Add operation
−	Minus operation
∗	Multiplication operation
/	Division operation
> (≥)	Greater than (Greater than or equal to)
< (≤)	Less than (Less than or equal to)
&&	Logic and
||	Logic or
QC.a_num_trks	The available number of trucks that have already arrived at the QC.
QC.p_num_trks	The predicted number of trucks to serve the QC, including trucks already arrived and to be arrived in the near future.
QC.p_num_tsk	The predicted number of tasks to be finished at the QC in the near future.
d(QC.f_tsk_source, truck)	The distance between the current position of the truck and QC.f_tsk_source.
QC.taskNum	The remaining number of tasks the QC needs to finish.
QC.max_num_limit	The maximum allowable number of assigned trucks to any QC.
QC.priority	A priority related to the QC.
Constant	Random constant number.

Operations such as crossover, mutation, and selection are employed as evolutionary methods between generations of individuals. The *sub-tree*

crossover is used as the crossover operation. Given two parents, it involves the random and independent selection of a crossover point in each parent tree. Subsequently, the offspring is generated by replacing the subtree rooted at the crossover point in a duplicate of the first parent, with a duplicate of the subtree rooted at the crossover point in the second parent.

The fitness of an individual is the obtained average total crane waiting time over 20 problem instances in the training set when the individual is used to guide the solution construction process step by step. Specifically speaking, at each decision point, an individual (a GP tree) is used to calculate a score for each work queue (QC). Subsequently, the first task from the work queue with the lowest score is selected.

The hyperparameters are shown in Table 4.2, and the evolutionary process of the population is illustrated in Algorithm 12. The population is evolved until the max generation is reached.

Table 4.2: Hyperparameters in Data-driven GP.

Name	**Value**
Population Size	500
Max Generation	100
Crossover Probability	0.45
Mutation Probability	0.15
Reproduction Probability	0.4
Tree Initialisation Method	Ramped half-and-half
Selection Method	Tournament selection, size 9
Depth Restriction	12

4.3.2 Datasets

The exact same datasets as introduced in section 3.3.2 were selected for these new experiments. This facilitates the comparison of the proposed DRL-HH method with the DRL based method presented in the previous chapter.

Algorithm 12 The Data-driven GP for dynamic truck dispatching

Require: *Initial Population*, *max_generation*
1: $p \leftarrow$ *Initial Population*
2: *generation* $\leftarrow$ *0*
3: **while** *generation* $<$ *max_generation* **do**
4: *p.calculate_fitness()*
5: *p.penalize_long_individuals()*
6: *next_generation* $\leftarrow$ *Population with 0 individuals*
7: **while** *next_generation.size()* $<$ *p.size()* **do**
8: Insert an *individual* to *next_generation* by Crossover, Mutation,
9: or Reproduction in p
10: **end while**
11: $p \leftarrow$ *next_generation*
12: *generation* $\leftarrow$ *generation* $+ 1$
13: **end while**

4.3.3 Experiment Settings

Table 4.3 lists the hardware and software settings for the experiments.

Table 4.3: Hardware and software settings for the experiments

Name	Detailed configuration	Name	Detailed configuration
Processor	Intel Core i7-8700K	Linux	Ubuntu 16.04 LTS
Memory	48GB	Python	2.7.12
Graphics	GeForce GTX 1080 Ti	Tensorflow	1.0.0

We adopted a four-layer DRL similar to the one used in Chen & Tian (2019), where a DRL was used to solve some deterministic COPs. In the three problems they solved, their neural networks include one, two, and two hidden layers, respectively.

For our implementation, the number of neurons in every layer in the two neural networks (NN) are shown in Table 4.4. The inputs of the NNs are the 'states'. Before feeding them to the NNs, the features in the states are normalised to the range [-50, 50]. The number of dimensions of them are related to the number of QCs. Therefore, the number of neurons in the input layers of the two NNs are different.

The output layers of the NNs outputs the Q-values for each agent action

Table 4.4: The number of neurons in our DRL for different datasets.

	Layer 1 (Input)	Layer 2	Layer 3	Layer 4 (Output)
Small sets	16	100	160	10
Big sets	24	100	160	10

which is a low-level heuristic. Therefore, the number of neurons in the output layers of the two NNs are the same since it is equal to the number of low-level heuristics.

We used the rectified linear unit function (ReLU) as the activation function for each hidden layer. Since the NNs approximate real-valued future rewards, there is no activation function on the output layer, and we did not use one-hot encoding, too.

In DRL-HH, ε-greedy policy was used to choose actions (see line 8 of algorithm 8). ε was set to 0.9 in the training process and was set to 1 during the test process. The reward decay γ shown in line 19 of algorithm 8 equaled to 0.8. The size of the data pool $\mathbf{M}$ (in line 2 of algorithm 8) was 3000 and the size of the mini-batches (in line 14 of algorithm 8)was set to 60. i.e. for every training step, 60 transition experiences (s_j, a_j, r_j, s'_j) were fed to the neural network of the DRL agent to train the model.

The loss function is shown in line 21 of algorithm 8. We chose the RMSprop optimiser with the learning rate equalled to 0.001 to minimise the loss.

4.3.4 Termination Criteria

When solving complex COPs, the determination of termination criteria is a crucial aspect in designing and evaluating algorithms. Termination criteria refer to the conditions that dictate when the solving process of an algorithm should cease. These criteria are typically defined to ensure that the algorithm has achieved its intended goal, that further computation is

unlikely to yield meaningful improvements or the running time reaches a certain time limit.

For the MCH, it is a rule-based method that can be directly used without the need for training or evolution. Therefore, when employing this method, there is no need to consider termination criteria.

For DRL-HH, we consider the convergence of performance as the termination criterion, which implies that the DRL agent's performance stabilises over a certain number of episodes. Specifically speaking, when the standard deviation of the average total crane waiting time across 500 consecutive episodes is less than or equal to 50 seconds and 150 seconds for Small_Basic and Big_Basic, respectively, the training of the DRL-HH agent is terminated.

For Data-driven GP, when the maximum number of generations is reached, the algorithm terminates, returning the current best solution. In our experiments, the maximum number of generations is set to 100, which is smaller compared to the values typically set during GP evolution (e.g., 500, 1000, or more). However, given the excessive time consumption, it becomes impractical and unacceptable to evolve a larger number of generations, such as 500 generations. For example, it takes 0.6 seconds for each individual to solve a problem instance in Big_Basic. Since the training set consists of 20 problem instances, the evaluation of each individual would consume approximately $0.6 * 20 = 12$ seconds. Therefore, when the maximum number of generations is set to 100 and the population size is 500, the entire evolutionary process would require a total of $12 * 100 * 500/3600 \approx 167$ hours. This is approximately 25 times the time consumption of DRL-HH. Because in the training of DRL-HH, each episode it does solution construction for 20 problem instances, which takes approximately 0.6 * 20 = 12 seconds, too. And the training of DRL-HH reaches performance convergence in an average of

1992 episodes, as shown in the following experiment results. Therefore, the training of DRL-HH only requires about $12 * 1992/3600 = 6.6$ hours.

When discussing the fairness of comparing various methods, firstly, in comparison to DRL-HH, the Data-driven GP method has spent a longer training time. Meanwhile, it is important to acknowledge that, under similar software and hardware environments, the training of DRL-HH also consumes additional computational power on the GPU of high-performance graphics cards. Although the neural network employed may not be particularly deep, it should still be taken into consideration when considering fairness. However, quantifying this additional computational power and further converting it into CPU power for a completely fair comparison of computational power and time consumption between the two methods is very challenging. While this may be a direction that could be explored in the future, it currently does not fall within the central focus of this study.

4.3.5 Experimental Results

There are stochastic components in our algorithm, such as the initial values of the weights, biases in the neural networks and the randomness in the low-level heuristics. We therefore repeated the experiment (i.e. both training and testing) 10 times. During the training, our DRL agent converges after about 2000 episodes on both Small_Basic and Big_Basic. In each episode, the entire training dataset (20 problem instances) was used to train the agents.

Once the training is completed, the resulting DRL-HH method is evaluated for its performance, scalability and relative performance against the simple DRL method. In each test experiment, the algorithms were used to solve 100 test instances. For each test instance, the algorithms were run 200 times with different random seeds.

The average total crane waiting time over 100 problem instances was adopted as the performance indicator. The results obtained from the proposed method and the other two benchmark methods are shown in Table 4.5. Meanwhile, Wilcoxon Rank Sum tests were conducted to compare the results obtained from any two out of the three methods in this experiment and the following two groups of scalability experiments (see Section 4.3.5.1). All of the obtained p-values are less than 0.03, except for one p-value obtained from the results between MCH and Data-driven GP, which is greater than 0.05 (the corresponding results are displayed in italics in Table 4.8). In general, this indicates statistically significant differences between the results obtained from all the three methods.

Table 4.5: The performance of DRL-HH in comparison with manual heuristic and a data-driven GP method. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev			Imp% of DRL-HH over MCH
	MCH (baseline)	Data-driven GP	DRL-HH	
Small_Basic	1808.02/15.72	1786.32/9.97	1674.40/14.58	7.39%
Big_Basic	6392.29/56.78	6333.48/36.60	5868.30/66.68	8.20%

It can be seen that the data-driven GP is marginally better than the manual heuristic, while DRL-HH performs the best among the three methods. This was expected because our DRL-HH can balance the long-term and short-term rewards with DQN training. Both the manual heuristic and data-driven GP may suffer from solving extreme test instances. On average, DRL-HH obtains a significant improvement when compared with manual heuristic (7.37% for Small_Basic and 8.20% for Big_Basic).

It was shown that combining multiple heuristics is beneficial than applying any of them alone (Burke et al., 2007c; Pillay & Qu, 2021). The comparison of our proposed DRL-HH against each individual heuristic (introduced in Section 4.2.4.3) is presented in Table 4.6, and results confirm

this finding in the literature.

Since a certain degree of randomness (as shown in line 14 of algorithm 9, line 16 of algorithm 10 and line 19 of algorithm 11) exists in these low-level heuristics, we run the experiments ten times. In each of them, all the 100 problem instances in the datasets were tested.

Table 4.6: Results of the low-level heuristics

Datasets	Low-level heuristics	Configurable parameters	Average total crane waiting time(s) / StdDev
Small_Basic	Distance heuristics	0.3	1907.88 (15.34)
		0.6	1923.38 (16.22)
		0.9	1945.33 (23.12)
	Urgent degree-first heuristics	0.3	1888.25 (22.13)
		0.6	**1872.13 (17.85)**
		0.9	1962.26 (12.45)
	Unbalanced degree-first heuristics	0.3	1898.27 (17.62)
		0.6	1886.95 (13.66)
		0.9	1878.42 (11.82)
Big_Basic	Distance heuristics	0.3	6850.12 (42.14)
		0.6	6628.73 (40.83)
		0.9	6623.26 (31.45)
	Urgent degree-first heuristics	0.3	6546.85 (36.83)
		0.6	6654.18 (35.94)
		0.9	**6449.52 (33.25)**
	Unbalanced degree-first heuristics	0.3	6733.49 (45.61)
		0.6	6579.34 (38.26)
		0.9	6728.11 (33.42)

4.3.5.1 Scalability Evaluations

In real-world scenarios, the generality of the trained model is of high importance. In the problem faced by Ningbo Port, the number of tasks in the datasets and the number of work queues (which is equal to the number of QCs) may change at some point. Therefore, two groups of experiments (Scalability Experiment 1 and Scalability Experiment 2) were conducted to evaluate the generality of the trained DRL-HH agent.

In Scalability Experiment 1, DRL-HH was tested on 'Small48T', 'Small96T', 'Big144T' and 'Big288T', where the number of tasks in a problem instance is different from those of the training instances (See Table 3.2 for details). The experimental results are presented in Table 4.7.

Table 4.7: Results of scalability experiment 1. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev			Imp% of DRL-HH over MCH
	MCH (baseline)	Data-driven GP	DRL-HH	
Small48T	1017.06/7.52	1037.40/5.97	971.44/10.49	4.52%
Small96T	2207.33/16.77	2192.76/18.73	2130.67/19.79	3.51%
Big144T	3601.26/22.77	3580.37/31.28	3450.30/32.67	4.20%
Big288T	8489.78/45.56	8423.56/58.12	8175.28/54.37	3.71%

In Scalability Experiment 2, DRL-HH was tested on 'Small3WQ', 'Small2WQ', 'Big5WQ', 'Big4WQ', 'Big3WQ' and 'Big2WQ', where the number of work queues involved is decreased (See Table 3.2 for details), results shown in Table 4.8.

Table 4.8: Results of scalability experiment 2. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets	Average total crane waiting time (s) / StdDev			Imp% of DRL-HH over MCH
	MCH (baseline)	Data-driven GP	DRL-HH	
Small3WQ	1360.45/8.47	1354.74/7.65	1313.10/10.84	3.42%
Small2WQ	895.43/5.36	889.79/7.07	869.92/9.48	2.93%
Big5WQ	4850.87/35.68	4805.76/29.94	4646.37/40.24	4.22%
Big4WQ	3920.40/29.45	3898.45/31.39	3774.21/39.71	3.76%
Big3WQ	*2620.34/21.48*	*2612.49/20.22*	2533.54/22.77	3.32%
Big2WQ	1745.55/14.73	1735.95/14.94	1692.46/18.38	3.08%

In these two groups of scalability experiments, the changes of both the number of tasks in the datasets and the number of work queues rendered significant changes in the problem structure. For instance, changes in the number of work queues (QCs) imply alterations in the structure of

the graph $G = (N, A)$ that describes the problem (i.e., variations in the number of nodes). Hence, obtaining a trained model that could perform well in different situations was considerably difficult. The results of manual heuristic and data-driven GP are still relatively close: data-driven GP is slightly better than the manual heuristic (except for the task set with 48 tasks in Table 4.7).

Notably, in Table 4.7 and Table 4.8, the results of data-driven GP in every row were obtained by rules developed separately for different situations. Using the results obtained by a single rule in the data-driven GP would be unfair. Yet, in real world, when faced with continuously changing scenarios, data-driven GP would experience difficulties in automatically making a choice to call different rules. In DRL-HH, since the deep neural network can learn non-linear relationships and encode kinds of knowledge in different problem instances, good performance with only one well-trained model could be achieved. DRL-HH is better than manual heuristic in all experiments (from 2.9% to about 4.5%), as observed from the two tables.

4.3.5.2 Discussion about Generality

The generality of the model has been partially validated through the aforementioned performance tests and the two groups of scalability experiments.

Firstly, from the results of the first group of experiments, namely the performance test, it can be observed that the trained model performs well on unseen problem instances of the same type as those used for training. Therefore, at the level of problem instances of the same type, the proposed method exhibits a high degree of generality.

Secondly, in our formulation, the trained model can also adapt well to situations where there are changes in the number of tasks and reductions in

work queues, without the need for retraining. However, when the problem instance involves an increase in the number of work queues, it implies the need to augment the dimensionality of the state in DRL to account for the features related to the increased work queues. Consequently, it also necessitates an expansion of the input dimensionality of the Q network. In this case, it is necessary to retrain the model.

Thirdly, in terms of generality at the cross-problem level, it is worth noting that DRL-HH can be applied to various online COPs formulated as sequential decision-making problems, indicating a certain level of cross-problem generality. However, it should be acknowledged that this approach relies on problem-specific features and low-level heuristics, which imposes certain limitations on its cross-problem generality. When applying DRL-HH to address new problems, it is necessary to incorporate some problem-specific design and also retrain the model.

4.3.5.3 DRL VS. DRL-HH

Finally, the direct use of DRL to choose work queues (the method proposed in the last chapter) was tested against the DRL-HH for both the Small_Basic and Big_Basic task sets. In the experiment where the DRL directly chooses work queue, all the hyper-parameters are the same as those in the DRL-HH experiments. Again, the simple DRL was run 10 times with the random seeds. In the simple DRL method, the agent may choose an action that violates some of the constraints (e.g. going to work queue without tasks). If this happens, the agent heavily to make the decision extremely unpopular, namely giving a big negative reward. In this experiments, we give the agent rewards of -10000 and -20000 which are far smaller than a normal reward signal, for Small_Basic and Big_Basic datasets, respectively.

However, although the punishment strategy addresses this problem to a

certain extent, the training time increases greatly, as shown in Table 4.9 for simple DRL to obtain similar results to those by DRL-HH. The simple DRL consumes about 3.7 (for Small_Basic) and 4.1 (for Big_Basic) times training time, respectively, compared with that of DRL-HH. The simple DRL also requires about 4 and 4.5 times training iterations compared with that of DRL-HH. Using DRL directly suffers from considerably slow convergence.

Table 4.9: Average training time and number of training episodes for two methods to achieve the same results

Task sets	Methods	Average training time (hour) / StdDev	Average num of episodes / StdDev
Small_Basic	DRL-HH	1.45/0.1	1933/61
	Simple DRL	5.40/0.18	7740/209
Big_Basic	DRL-HH	6.62/0.28	1992/75
	Simple DRL	27.33/0.75	8864/218

It is worth noting that although the training of DRL-HH takes a lot of time, it is very fast in testing/execution once trained. The average execution time for a trained DRL-HH agent to solve a small instance (containing 72 tasks) is 0.165 seconds and 0.575 seconds for a large instance (with 216 tasks).

4.3.6 States Spectral Analysis

Models trained with data-driven methods are often concerned with their interpretability. Complex models acquired through traditional machine learning methods, such as deep neural networks, are difficult to comprehend. The low-level heuristics in the present hyper-heuristics were manually designed. The proposed DRL hyper-heuristic thus provides a certain level of interpretability. To further understand the trained model, in the test phase, spectrum analysis of the states was conducted to identify possible patterns between the states and the corresponding actions.

Spectrum analysis is a method used to analyse the spectral character-

istics of signals. It involves transforming signals into the frequency domain to study their frequency components and energy distribution. It has extensive applications in fields such as signal processing, medical diagnosis, and astronomy. In DRL, the collection of states in which a specific action is selected can be regarded as a form of 'signal', thereby spectral analysis can be performed on them. It can assist us in understanding the common characteristics of environmental states when selecting specific actions or further identifying some patterns between the states and the corresponding actions.

We collected 12,000 states and removed duplication. The remaining 11,035 states were then partitioned into nine groups pursuant to their corresponding actions executed, as shown in Figure 4.3. It can be observed that out of the total 11035 states, actions 0, 1, 3 and 6 are among the most frequently used heuristics.

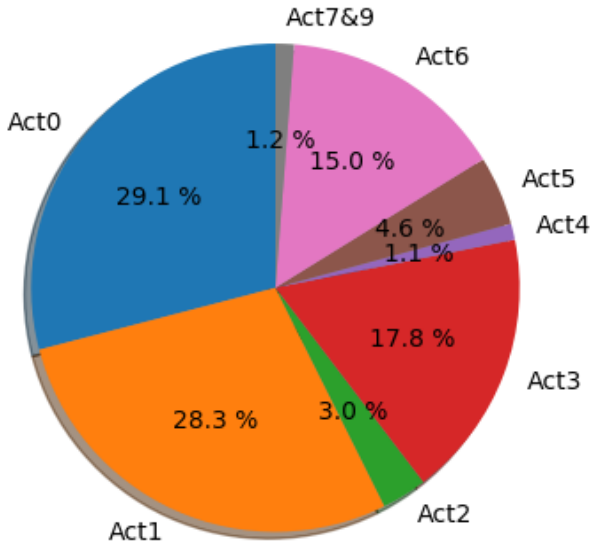

Figure 4.3: *The distribution of the actions corresponding to the 11035 states.*

Spectrum analysis was conducted on each group of these classified states. Taking action 0 in Figure 4.4 as an example, where the x-axis represents the state elements discussed in Section 4.2.4.1. The urgent degree was employed to replace the supply and the demand as it is a more intuitive indicator and more conducive to discover certain patterns. The range of val-

ues of different state elements was normalised to [-50, 50]. For the elements of work queue length, the closer it was to -50, the smaller the work queue length was. When it was equal to -50, the work queue was empty. The colour represents the frequency of the elements that fall into a specific area exhibited as the colour bar on the right side of the figure. Drawing an analogy between spectrum analysis of states in DRL and that of a signal, each state element (e.g. work queue length) in states can be regarded as a signal, while the range of values for each state element ([-50, 50]) corresponds to the frequency range of a signal. Furthermore, the varying colors associated with different values of state elements can be likened to the varying energy distribution in different frequencies in a signal.

In Figure 4.4 and Figure 4.5, when focusing on the work queue length elements, it can be observed that DRL-HH tended to choose actions 0 and 3 at the middle and late stages of an episode (work queue length in [-50, 2] and [-50, 26], respectively). However, at the early stage of an episode, actions 0 and 3 were chosen with a low frequency.

When attention shifts to the urgent degree, Figure 4.5 and Figure 4.6 indicate that compared with action 4, action 3 was more likely to be chosen when the urgent degree was high.

The results reveal that certain patterns exist between the states and the corresponding actions. These patterns are "statistical/fuzzy" patterns and cannot be used as clear-cut rules for operational-level decision making. However, they are valuable for managers to develop insights and understanding about this challenging problem. In real life, people instinctively reject decisions that are hard to understand. This presents a challenge for the use of DRL in an industrial environment like Ningbo Port. DRL-HH can fully utilise the powerful learning and exploration of DRL, but more importantly, can also provide explainable solutions to some extent, allowing

decision-makers in industry to understand and thus accept the suggestions provided by the algorithm more easily.

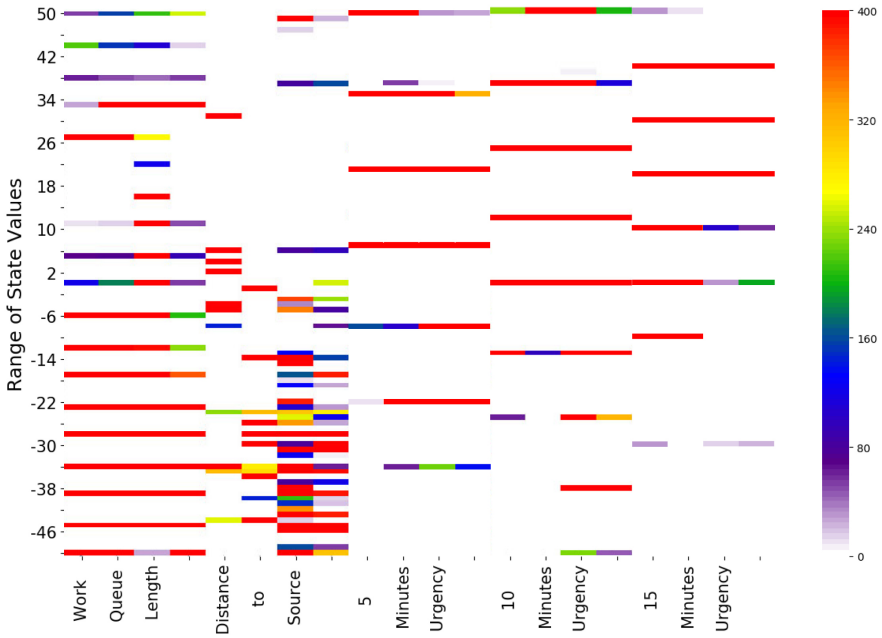

Figure 4.4: *The spectrogram of states for action 0.*

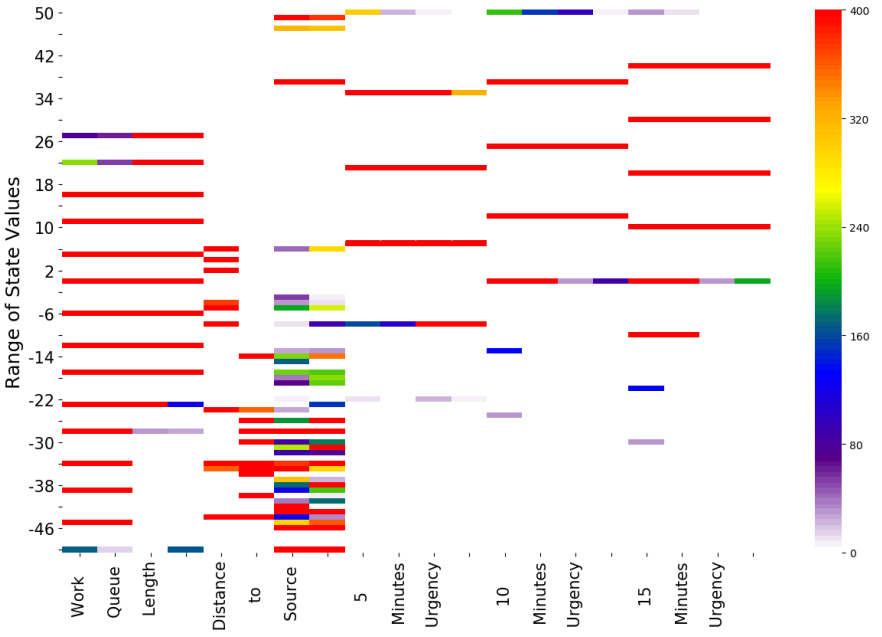

Figure 4.5: *The spectrogram of states for action 3.*

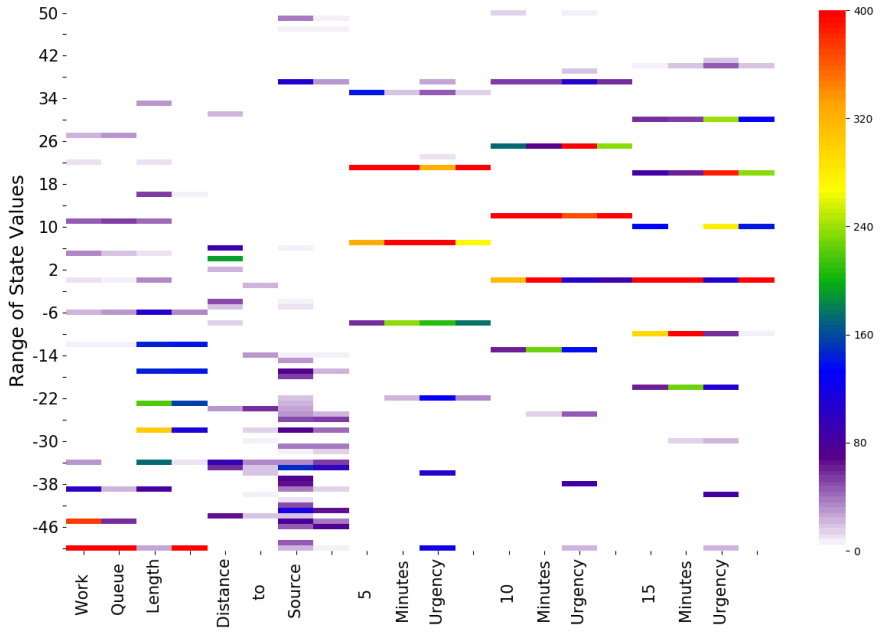

Figure 4.6: *The spectrogram of states for action 4.*

4.4 Summary

In this chapter, we propose a DRL-based hyper-heuristic framework in which DRL was introduced into a constructive hyper-heuristics framework to address the challenging online container truck dispatching problem. Experimental results highlight several advantages with this new framework. Firstly, it shows better performance compared with the existing state-of-the-art methods, namely the MCH, and the data-driven GP. The reason could still very likely be that DRL-HH can balance the long-term and short-term rewards with DQN training. Both the manual heuristic and data-driven GP may suffer from solving extreme test instances. Also, the performance of the data-driven GP is not as good as DRL, which is likely due to the limitation on the depth of the GP-tree in the experiments, resulting in the inferior capability of the GP-tree compared to the deep Q-network. However, if we do not impose restrictions on the depth of the GP-tree, it would increase the search space of GP, leading to more computational time.

Secondly, compared to the approach proposed in the previous chapter that directly uses DRL, DRL-HH does not exhibit a significant performance improvement. It primarily enhances the convergence speed of the algorithm and reduces training time. Additionally, due to the incorporation of human-designed low-level heuristics and the conducted state spectral analysis, DRL-HH demonstrates improved interpretability of the solutions, thus is more acceptable in real-life applications. Finally, DRL-HH shows good scalability when the problem instances change significantly.

In the next chapter, we have considered a more challenging online COP, a multi-scenarios online container truck routing problem. It is an extension of the problem addressed in this chapter, and it further considers the impact of external trucks on the dispatching of internal trucks, making it more closely aligned with real-world port environments. In different scenarios, the distribution of the random variable (crane service time) is no longer unique and fixed but varies over time with different randomly occurring distributions. Therefore, it is a more realistic and challenging problem that is worth exploring. A new method that combines DRL-HH and genetic algorithms (GA) has been designed to solve this problem.

Chapter 5

Online Container Truck Routing Problem with Non-stationary Multi-scenario Switching

5.1 Introduction

The real-world port is an exceedingly intricate environment, subject to various factors that give rise to diverse operational scenarios during container truck dispatch. These scenarios exhibit distinct problem characteristics, occasionally even featuring disparate optimisation objectives, necessitating diverse strategies to address the varying scenarios.

According to the types of most of the tasks (loading/unloading) during a given time period, scenarios can be classified as inbound dominated, outbound dominated or inbound/outbound balanced scenarios. The unloading tasks are typically less susceptible to disruptions caused by truck delays due to their less stringent precedence requirements. Conversely, loading tasks necessitate strict adherence to predefined sequences, making delays capable

of exponentially propagating and resulting in substantial quay crane (QC) waiting. Therefore, distinct dispatch policies are imperative for scenarios with tasks dominated by either load or unload tasks.

Also, based on the distribution of yard cranes (YCs) within the container yard involved in a given set of tasks, the problem scenario can be categorised into two scenarios: YC-concentrated and YC-dispersed scenarios. For the former, the operation nodes located at a limited number of YCs are more likely to cause conflicts as YCs need to support multiple QCs simultaneously. In the case of the latter, conflicts between trucks at YCs are highly unlikely to occur due to their dispersed nature.

Furthermore, the varying available number of trucks for dispatch may lead port managers to prioritise different optimisation objectives, thereby giving rise to distinct scenarios: truck abundance scenario and truck shortage scenario. When ample trucks are available, the objective should be to minimise empty truck travel distances; otherwise the priority shifts towards mitigating the costly QC waiting.

Lastly, the uncertainties regarding the timing and quantity of external truck (ET) arrivals at the port have varying impacts on online internal truck (IT) dispatching, giving rise to different problematic scenarios, namely mild congestion, medium congestion and heavy congestion scenarios as shown in Figure 5.1. In this study, we specifically address the multi-scenario problem arising from this factor.

ETs are the primary means of transport for containers to and from the port. The uncertainty and randomness of ET arrival times at the port lead to congestion at the truck terminal during peak hours, which not only reduces terminal operational efficiency but also contributes to pollution emissions in the port area. The port managers have adopted several methods, including truck appointment that attempts to reduce ET arrivals during

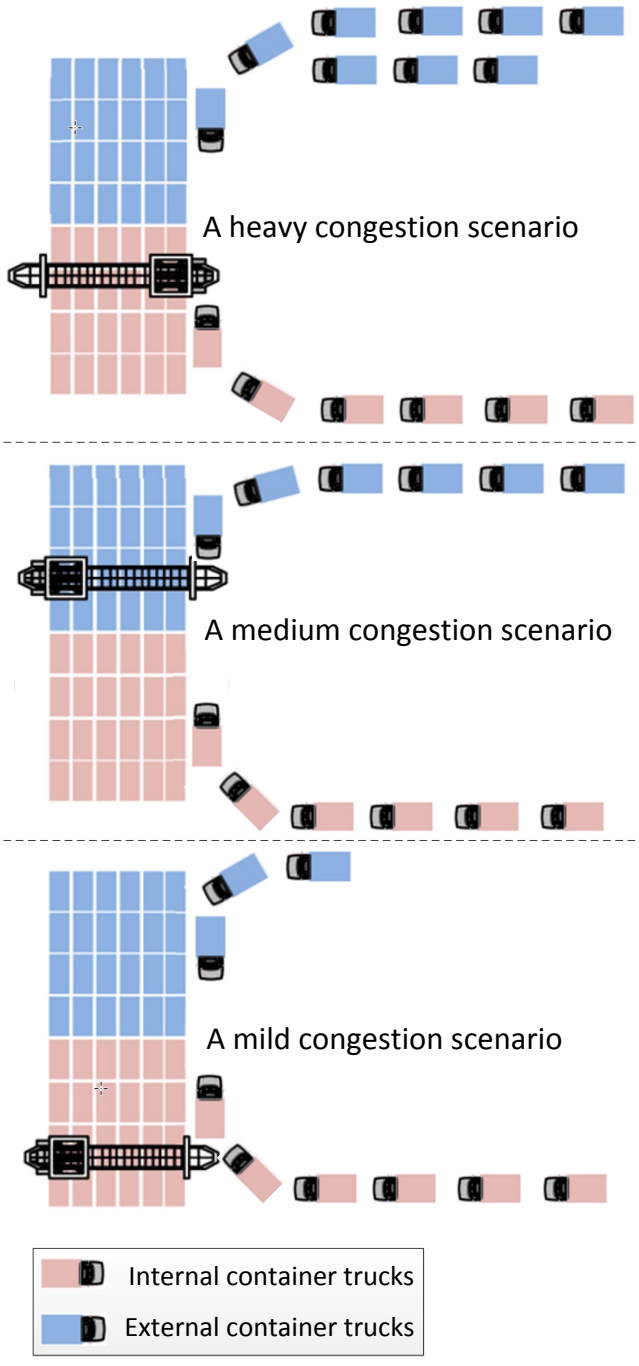

Figure 5.1: *Examples of different scenarios*

peak hours by controlling the acceptable number of ETs during the reservation period, time window management and time-varying tolls, to control the uncertainty and randomness of ETs arriving at the port. However, their uncertainty and randomness are still not well controlled and predicted (Ramírez-Nafarrate et al., 2017).

When the YCs are serving ETs, services for ITs at YCs are affected (Zehendner & Feillet, 2014). Meanwhile, to increase the turnover volume of empty containers, in recent years, many ports have offered direct empty container pick-up services, which allow ETs to arrive directly at the seaside, pick up the empty containers with the help of QCs, and transport empty containers outside the port. As a result, ETs not only have an impact on the operation of ITs in the yards but also affect the operation of ITs on the seaside. Therefore, the impact of the uncertainty caused by ETs cannot be ignored in real-world IT dispatching problems.

This problem is more challenging than the problems solved in the previous two chapters because the uncertainty of the service times of ITs in QCs and YCs due to different weather, operators and equipment in the previous two chapters is relatively small and follows fixed normal distributions. However, the uncertainties regarding the timing and number of ET arrivals at the port cause their effect on the crane service times (CST) of ITs at QCs and YCs to change over time. The more the number of ETs entering a port at a given time, the greater the impact (greater mean and variance) on the service time of ITs at QCs and YCs is, and vice versa.

In the different scenarios, the CST of ITs varies with different degrees of fluctuation, showing significantly different distributions. Since this is an online problem, different scenarios randomly emerge over time during operation. Therefore, we are unable to anticipate in advance the potential scenarios that may arise, nor can we predict the next task's CST.

In such online multi-scenario problems with uncertainties, the information that is important for decision-making, such as CST, sometimes varies lightly and sometimes dramatically, making decision-making extremely difficult. We found by analysing the data that the happening of a scenario is usually localised in time, i.e. once a scenario emerges, it usually lasts for a while and does not immediately disappear after it occurs.

This means that we can use different algorithms for different scenarios at different problem-solving stages. If the algorithms are chosen appropriately, the overall quality of the solution will be improved. While traditional meta-heuristics can obtain high-quality solutions when the CST is relatively stable, a DRL has an advantage in dealing with dynamic sequential decision problems, which means it can perform well when the CST varies significantly. Then, the remaining question is how to design a framework that can identify different scenarios while allowing meta-heuristics and DRL to collaborate well with each other.

This chapter proposes a hyper-heuristic framework combining a DRL and GA to solve the online multi-scenarios container terminal truck dispatching problem. This framework takes advantage of both DRL and GA to make appropriate decisions at different stages of problem-solving and reduces GA's response times through strategies of *variable visions* and a *surrogate network*. The remainder of this chapter is organised as follows: Section 5.2 is the problem description. Section 5.3 introduces the proposed DRL-GA-HH framework. In Section 5.4, the proposed framework is evaluated by solving the online multi-scenarios container terminal truck dispatching problem with uncertainties described in Section 5.2, followed by discussions of the experimental design and results analysis. Finally, Section 5.5 concludes this chapter.

5.2 Problem Description

In this multi-scenario problem related to ETs, the direct impact of ET on online IT dispatching is manifested through the varying degrees of queuing congestion they induce among ITs at operational nodes (QCs or YCs). In this study, we do not directly consider the effect of the queuing of ETs at these nodes. Instead, a small trick is used, which is to consider the increase in waiting time for an IT at a QC or YC, which is caused by the queuing of ETs, as an increase in service time for this IT at the QC or YC. With the help of this trick, we can easily simulate the effect of ETs on this problem while not having to simulate the extra complex dynamics of the port model caused by the queuing of ETs at the operational nodes.

Due to the advantages of the decoupling of modelling and data strategy mentioned in the previous chapter (Section 4.2.1), we can maintain the exact consistency of the problem description with Section 2.2, while making modifications solely to the training data, thereby transforming the problem into a more challenging multi-scenario problem. Specifically, only the distributions of the CST of tasks in the problem instances are changed (See details in Section 5.4.1.2).

Uncertainty about the arrival time and number of ETs at the port leads to a great impact on the dispatching of ITs. Specifically, ETs can affect the dynamic environment of a port in two respects. Firstly, it increases the density of vehicles inside the port, which reduces the speed of the ITs. Secondly, it complicates the queuing of ITs at QCs and YCs, which makes the uncertainty of the waiting and service time of the ITs in the yard even greater.

We adopted the *vacation queue model* to formulate the queuing of ETs and ITs in queues at YCs (QCs). It has been used in different areas, such

as flexible manufacturing and limited production (Ke, 2003). It uses the equipment's vacation time to complete other tasks so that its utilisation rate is increased. In this problem, when the queue length of ITs is less than a predetermined threshold, ITs and ETs are served following the FCFS policy. When the queue length of ITs is greater than the threshold, only ITs are served. From an IT viewpoint, time spent serving ETs can be regarded as YCs (QCs)' 'vacation time' or as breaks during the service of ITs. YCs return to serve ITs when the vacation (serving ETs) ends or the queue length of ITs exceeds the threshold (according to the vacation policy).

Due to the great uncertainties of ETs, it is hard to add their dynamics directly into the mathematical model of the ITs dispatch. Therefore, as mentioned above, in this modelling, we consider the effect of ETs as an uncertain disturbance of the service time of ITs. Specifically, when ITs and ETs are mixed in a queue, they are served in the order in which they arrive at the node. The model service time (ST_m) of an IT, which is used in this formulation, consists of two parts: the first is its real service time (ST_r) at the node, second is the ST_r of the consecutive ETs immediately in front of it that are inserted into the queue. For example, in Figure 5.2, the ST_m of IT_i is the sum of the ST_r of IT_i and ST_r of ET_j and ET_{j+1}.

In this multi-scenario problem, the variables, objectives, and constraints of the problem are exactly the same as those in the online container truck dispatching problem introduced in Section 2.2. The only difference between the multi-scenario problem and the previous problem is the distribution of the crane service time in both the QCs and YCs. In the previous problem, the distribution of the crane service time is a fixed normal distribution. In the multi-scenario problem, different distributions of the crane service time randomly emerge over time during operation. The different scenarios are defined by the different distributions of service times.

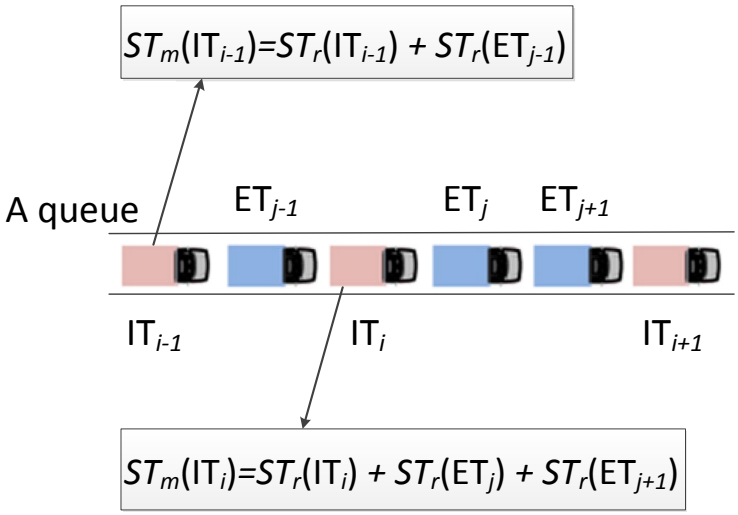

Figure 5.2: *The calculation of service time in this problem. ST_m stands for the model service time, while ST_r stands for the real service time.*

5.3 Methodology

The DRL genetic algorithm hyper-heuristic (DRL-GA-HH) proposed in this chapter is still based on the DQN framework. As shown in Figure 5.3, an additional group of GAs is added to the action set. Based on DRL's perception of the states of the environment, the framework is able to select the appropriate action from the action set to coordinate DRL and the GAs work together. In Figure 5.3, the orange lines show the processing flow of a traditional DRL. The green lines depict how the framework differs from the traditional DRL approach when collecting the agent's experience (see Section 5.3.3 for details).

In this method, the action set has been changed, containing three parts. The first is the actions that directly choose a work queue, which is used in the first study as shown in Section 3.2.2. Because these are the most basic indivisible actions, they are called atom actions in the DRL community. The second is a set of parameterised low-level heuristics used in DRL-HH (the method proposed in the last chapter). Their validity has been demonstrated in the last chapter. Therefore, they have been included in this action set.

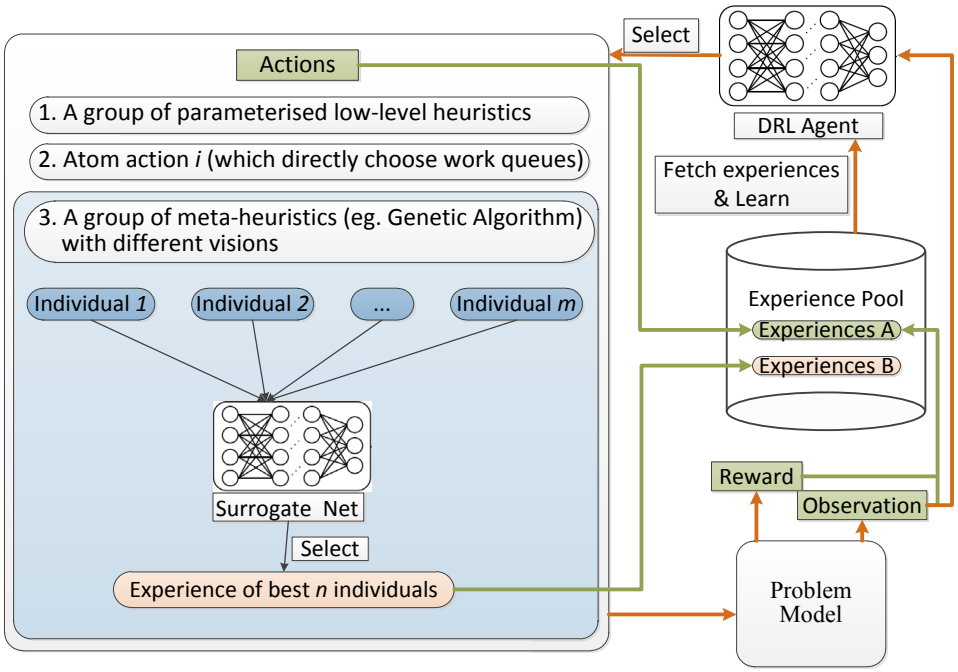

Figure 5.3: *The DRL-GA-HH framework*

The third part is a set of GAs with variable *vision* (See details about GA in Section 5.3.1).

In the whole system, GAs function as part of actions in DRL and are parallel to other actions. In this online approach, when the DRL agent selects other actions, such as an atom action (which directly chooses a work queue) and a parameterised low-level heuristic, it assigns only one task at a time. However, when the DRL agent selects a GA, the GA evolves a partial solution to provide a dispatching plan for the next n tasks (n is the length of the *vision*). This process continues until all of these tasks in the plan are dispatched. Only then does the DRL agent proceed to select an action again.

In this framework, the surrogate network shown in Figure 5.3 is a component of GA used for evaluating the fitness of GA individuals. It is an independent neural network and has no relation to the Q-network in DRL. Details about the surrogate network are introduced in Section 5.3.2.

In this multi-scenario problem, the service time of ITs shows differ-

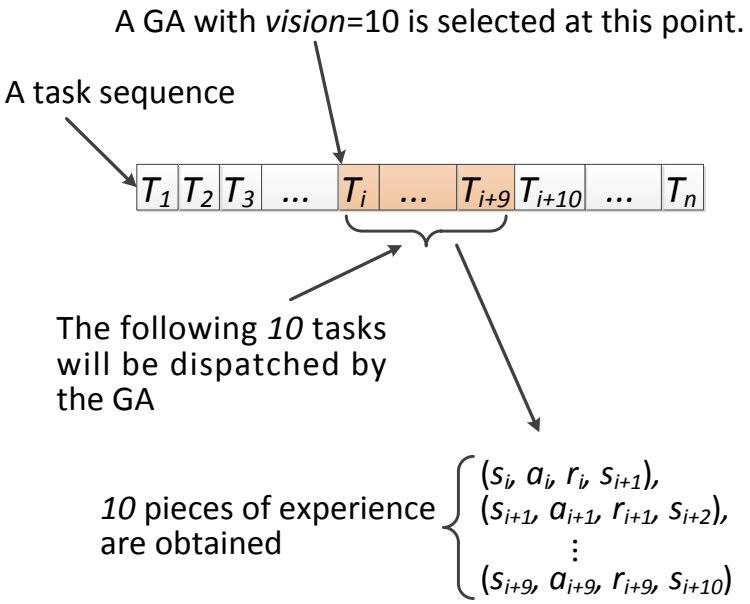

Figure 5.4: *A schematic diagram when a GA with vision=10 is selected*

ent distributions in different scenarios, and their means and variances show great differences. As shown in Figure 5.8, in mild congestion scenarios, the variances of the service time of ITs are relatively small, and GAs perform best in these scenarios. As the scenarios change from mild to heavy congestion, the variance of ITs' service time gradually becomes larger, and GA's performance gradually deteriorates. When the scenario moves completely to a heavy congestion scenario, a GA will not be a good choice.

In this method, the top-level architecture is DRL. After training, a single DRL decision model for all scenarios is obtained. It can perceptively sense scenario changes in the environment and select appropriate actions in different scenarios.

5.3.1 GA with Variable Visions

GA with variable visions is one of the tricks in this framework. In this DRL-GA-HH framework, at a decision point, if a GA is selected, its vision indicates how many following tasks the GA will dispatch (as shown in Figure 5.4). In other words, GA with variable visions evolves a partial

solution, denoted as $Psol_{GA}$, which indicates how to dispatch the next set of a number of *vision* tasks. Consistent with the previous two studies, in this online problem formulation, the act of dispatching a task is the selection of a suitable work queue (obtaining the index of the appropriate work queue, and subsequently assigning the first task from the work queue to the given idle truck). Therefore, a partial solution given by the GA consists of a sequence of indexes corresponding to the work queues. For instance, when a problem involves a total of 4 work queues (their indexes are 0, 1, 2, 3), each gene of a GA individual has 4 alleles correspondingly, labelled 0, 1, 2, 3. When a GA with a vision equals to 10 is selected, a possible partial solution $Psol_{GA}$ (a GA individual) that could be given by the GA is $[0, 3, 2, 2, 1, 3, 3, 3, 1, 0]$.

The crossover is simply replacing one of the genes in one parent by the corresponding gene of the other. And the mutation is assigning a new allele value for one of the genes.

To obtain the fitness of individuals in the GA, the partial solution before the decision point at which GA is selected, represented as $Psol_{bdp}$, is needed too. And a $CPsol_{bdp,GA}$ is a concatenation partial solution which concatenates a $Psol_{bdp}$ and a $Psol_{GA}$ (an individual).

Similar to the objectives in the previous two studies, the objective of this problem is also to minimise the average total crane waiting time. Therefore, the fitness of an individual is defined as the total crane waiting time corresponding to the $CPsol_{bdp,GA}$. It can be obtained through two different approaches. One is to directly acquire the total crane waiting time by sequentially executing each task in the $CPsol_{bdp,GA}$ within the port environment simulator. The other one is using a Surrogate Network that directly maps $CPsol_{bdp,GA}$ to the total crane waiting time (See details in Section 5.3.2).

Using this trick, when a GA is selected in a mild congestion scenario,

the GA can perform well. Meanwhile, when a GA is selected in a medium or heavy congestion scenario, it will not cause significant degradation in solution quality.

Specifically, when the DRL agent perceives that it is currently at the early stage of a mild congestion scenario, it tends to select a GA, especially a GA with a relatively big vision. In the late stages of a mild congestion scenario, GAs with relatively small visions perform better. This is because, in the late stages of a mild congestion scenario, a scenario switch may occur, which will exacerbate the degree of service time variation of ITs. In this case, the partial solution obtained by the GA will be poor. Also, although a GA is not a good choice in medium and heavy congestion scenarios, it may still be selected with a small probability. If this happens, a GA with a very small vision can reduce the negative impact on the solution to some extent.

In addition, compared with the direct use of GA, limiting the vision to a certain range can effectively reduce the GAs' computation time, which is also crucial in solving the online problem. Since the solution time of the GA shows an exponential rise as the *vision* increases, choosing a smaller *vision* will greatly reduce the solution time. Together with a high-performance computer, this can, to some extent, meet the real-time needs of online problems.

5.3.2 Surrogate Network for GA

Surrogate-assisted evolutionary computation uses efficient computational models, often known as surrogates, to approximate the fitness function in evolutionary algorithms (EAs). The fitness evaluations of an individual in EAs can be achieved by an experiment, a computational simulation, or an explicit fitness function. However, in real-world problems,

the evaluations using an experiment or a computational simulation may be time-consuming and expensive, and sometimes there is no explicit fitness function at all. Surrogate models aim to reduce time and computational resource consumption for fitness evaluations in EAs. And they are widely applied in real-world problems, such as antenna structure optimisation (Deb et al., 2014), hybrid vehicle design and control (Cheng et al., 2017) and aerodynamic design optimisation (Lian et al., 2010).

The computationally most efficient way of estimating fitness is the use of machine learning models, among which artificial neural networks (ANN) have been widely applied in modern surrogate modelling. Their accurate generalisation and parallel computation capabilities in complex engineering design problems are helpful in the rapid investigation of design space and searching for optimalities (Van Nguyen et al., 2015).

As this study addresses an online problem, it requires that the algorithm can respond quickly. However, in this problem, accurate fitness evaluations of individuals are achieved through simulation (sequentially executing each task in the $CPsol_{bdp,GA}$ within the port environment simulator.), which is very time-consuming. Although the performance of surrogates with different types of neural networks may vary somewhat, this is not the focus of this study. Therefore, we chose fully connected moderately deep ANNs as surrogate networks (SNs).

The structure of the SNs is illustrated in Figure 5.5. Given that the length of a partial solution $CPsol_{bdp,GA}$ varies, and the input dimension of a neural network is fixed, it is necessary to append a series of fixed padding values after a $CPsol_{bdp,GA}$ to make the input layer's dimension equal to a fixed value, namely, the length of a complete solution (the number of tasks contained in the problem instances). The output layer of SN directly outputs a total crane waiting time, which aligns with the fitness obtained

through simulation. Therefore, the fitness of GA individuals can be rapidly evaluated by SN.

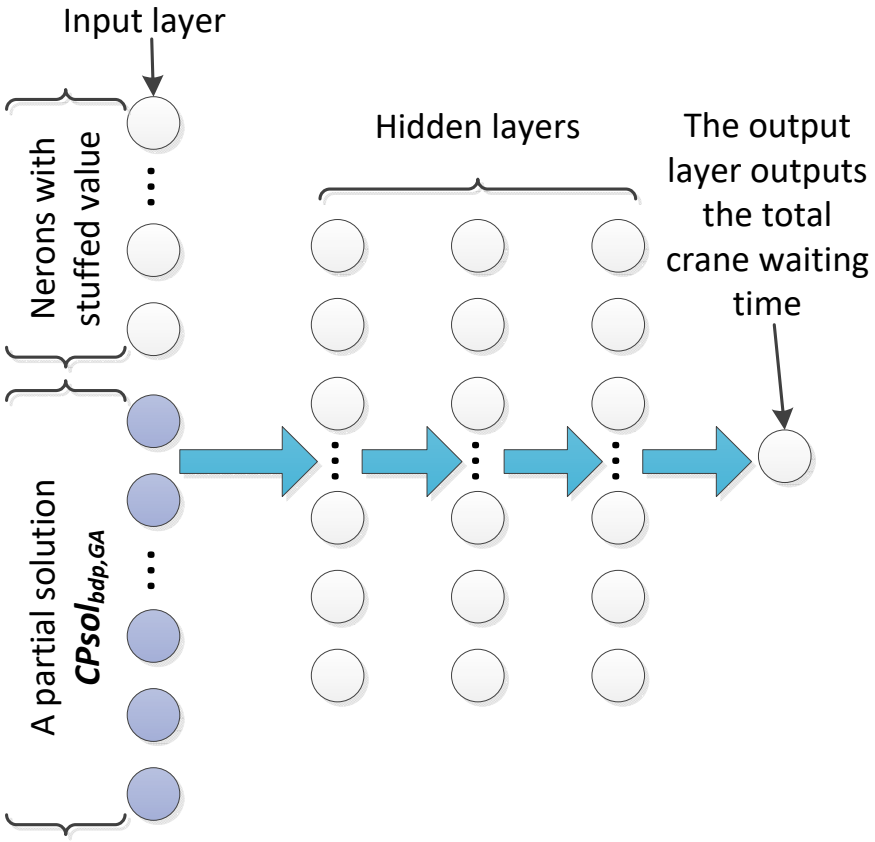

Figure 5.5: *The structure of the surrogate networks*

It is worth noting that online training is used for SN. This is because, during the training DRL, as the parameters of the DRL model undergo changes, the agent will explore different regions of the solution space, thereby acquiring entirely distinct data. Similarly, the SN also needs to be continuously updated to accommodate fitness evaluations as the training of DRL progresses.

The training data for SN is collected concurrently with the acquisition of fitness of an individual using simulations (as discussed in Section 5.3.1). Specifically, each training datum is an (x, y) pair, where x denotes a partial solution $CPsol_{bdp,GA}$ and y represents the actual total crane waiting time obtained through sequentially executing each task in the partial solution

$CPsol_{bdp,GA}$ within the port environment simulator.

Figure 5.6 shows the flowchart of the GA. In each generation, 30% (we did not fine-tune it) of the individuals' fitness evaluations were done by simulation. The remaining 70% of individuals' fitness is then evaluated by the updated SN.

After the GA converges, when we select a final individual, still, 30% of the individuals' (set A) fitness evaluations were done by simulation. The remaining 70% of individuals' (set B) fitness is evaluated by the SN. Then, we first select the top 5 individuals from set B, and further evaluate their fitness by simulation. If their fitness obtained by simulation is higher than the individuals' fitness in set A, we finally choose the individual with the highest fitness among the top 5. Conversely, if their fitness is lower than the individuals' fitness in set A, we choose the individual with the highest fitness in set A.

Thanks to today's advanced GPU technology and relatively small incremental training data, the training of SN can be completed in a very short time.

In this study, we adopted *non-elitist tournament selection* as the competitive selection method to increase diversity. The evolutionary process is repeated until the maximum number of generations is reached. All the GA-related hyperparameters are given in Table 5.2.

5.3.3 Experience Collection

Due to the introduction of GAs in this framework, the experience collection of the DRL Agent is also different from traditional DRL algorithms. In a traditional DRL framework, the experiences are generated by the direct interactions of the DRL Agent with the environment. For example, in a certain state s, the DRL Agent performs action a, obtains reward r, and

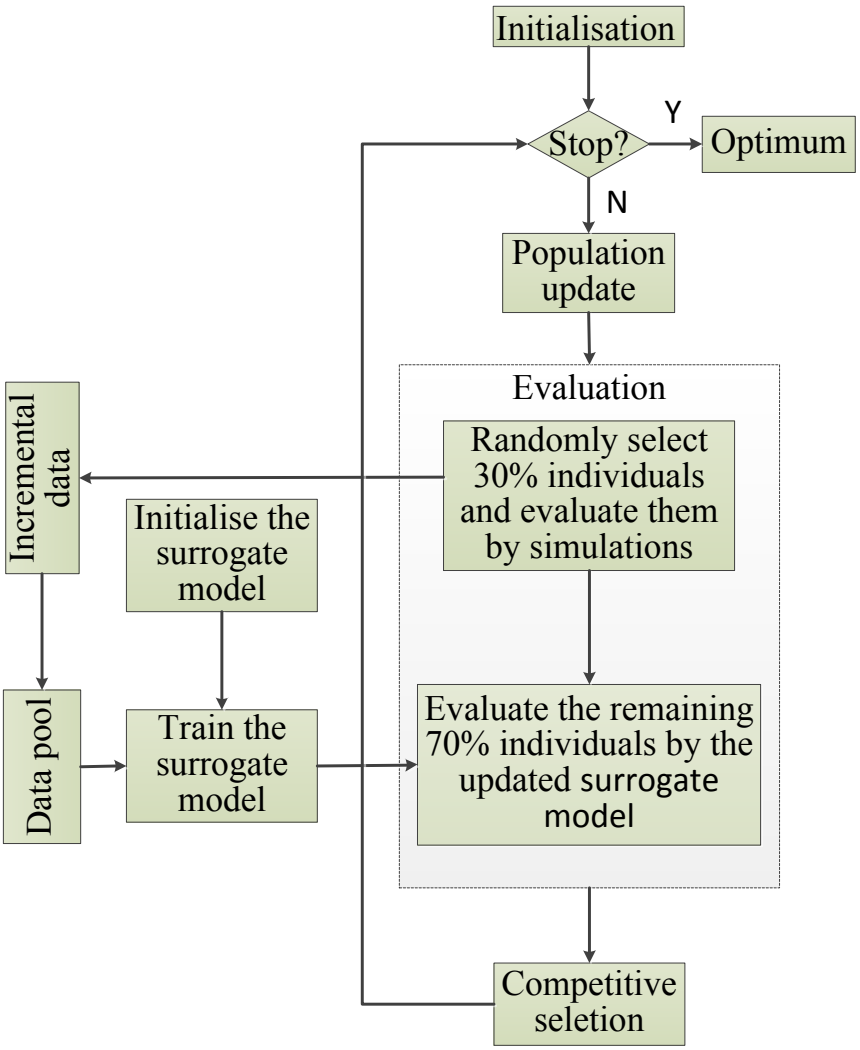

Figure 5.6: *The flowchart of the GA*

finally moves to the next state s'. Then it obtains a piece of experience (s, a, r, s'). While in this framework, if the selected action is a GA with a *vision* equal to 10, we can obtain ten pieces of additional experiences (as shown in Figure 5.4).

These additional experiences speed up the training process to some extent. This is due to the fact that the decisions associated with these experiences are guided by GAs, and these experiences thus are relatively high-quality (can lead the DRL Agent to get a relatively higher cumulative reward). In the traditional DRL framework, in the early stages of training,

the DRL Agent almost chooses actions blindly due to little knowledge of the environment, which makes the overall training efficiency relatively low. However, in the proposed DRL-GA-HH framework, with the help of these additional high-quality experiences, the training speed of the DRL Agent is significantly improved.

5.4 Computational Experiments and Analysis

In this section, we present the experiments carried out to test the performance and robustness of the proposed method. Firstly, we describe the hardware and software environment and the dataset applied for the experiments. This is followed by the settings of the numerous hyperparameters in the experiments, which include the hyperparameters related to DRL and those related to GAs involved in the framework. Then, we describe the purposes and methods of the three sets of experiments conducted. Finally, the experimental results are shown and analysed.

5.4.1 Experimental Settings

5.4.1.1 The Hardware and Software Settings

Unless otherwise indicated, all of the experiments were conducted on a high-performance server with a 3.7GHz*12 Intel Core Intel Core i7-8700K CPU, 48 Gigabytes of RAM and a GeForce GTX 1080 Ti graphics card. For consistency with the previous code, we used Python 2.7.12 and Tensorflow 1.0.0 under a 64-Bit Ubuntu 18.04 operating system. All processes related to neural network training in the experiments are done on the GeForce GTX 1080 Ti graphics card.

5.4.1.2 Dataset

Similar to the method used to generate the data in Zhang et al. (2022b), we first generated two datasets of different sizes: a small basic multi-scenarios dataset (Small_Basic_M) and a big basic multi-scenarios dataset (Big_Basic_M), each containing 70 instances. Half of the instances were used for training, and the other half for testing. In each problem instance of Small_Basic_M and Big_Basic_M, 72 and 216 tasks were included, respectively. Each task is described by a tuple as mentioned in Table 2.1, including a source node, destination node, source node service time and destination node service time. All these tasks are derived from a real-world port. And in order to fully represent the characteristics of different scenarios in the real world, and the fluctuation of the node service time of tasks when scenarios change, we manually generated the data of the node service time in different scenarios. Unlike the dataset used in Chapter 4 and 5, in a multi-scenarios environment, the distributions of the node service time of tasks change dramatically over time (with the switching of scenarios).

Figure 5.7 and Figure 5.8 show the difference between them. Figure 5.7 depicts the variation of node service times over time in the dataset used in experiments in Chapter 4 and 5. It can be seen that the node service time was drawn from several different Gaussian distributions, but since they were randomly drawn, the overall distribution remains normal as it is randomly selected. Meanwhile, their mean values did not change throughout the task assignment. However, in the dataset used in this problem (Figure 5.8), the distribution of the node service time shows an obvious change over time. There are very few fluctuations for the node service time in time intervals during the mild congestion scenarios but huge fluctuations during heavy congestion scenarios. And the node service time variation in medium congestion scenarios falls somewhere in between. At

the same time, their mean values have shown great ups and downs over time (μ rises from 120 seconds to 220 seconds and eventually falls gradually again to 120 seconds in the end).

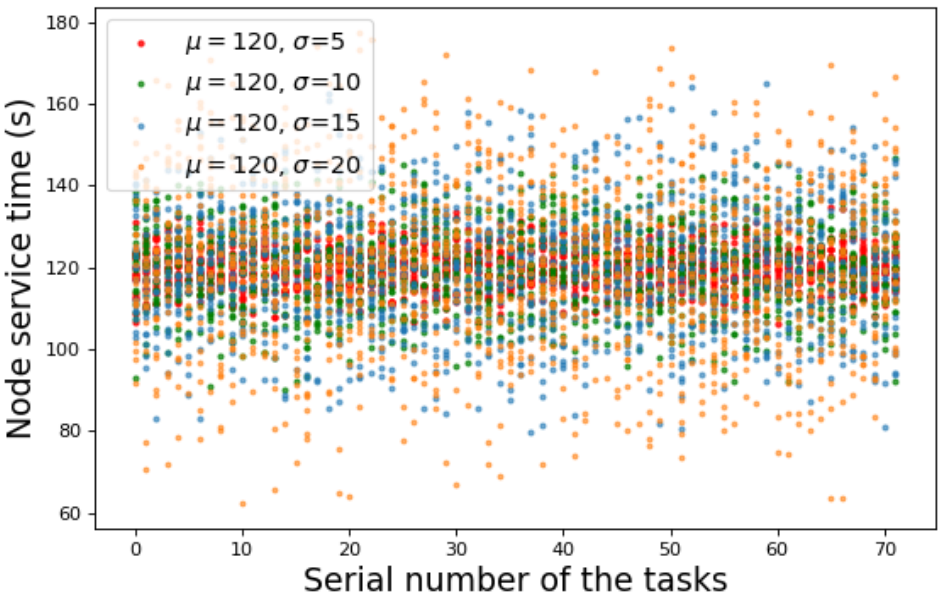

Figure 5.7: *The node service time distribution over time in DRL-HH*

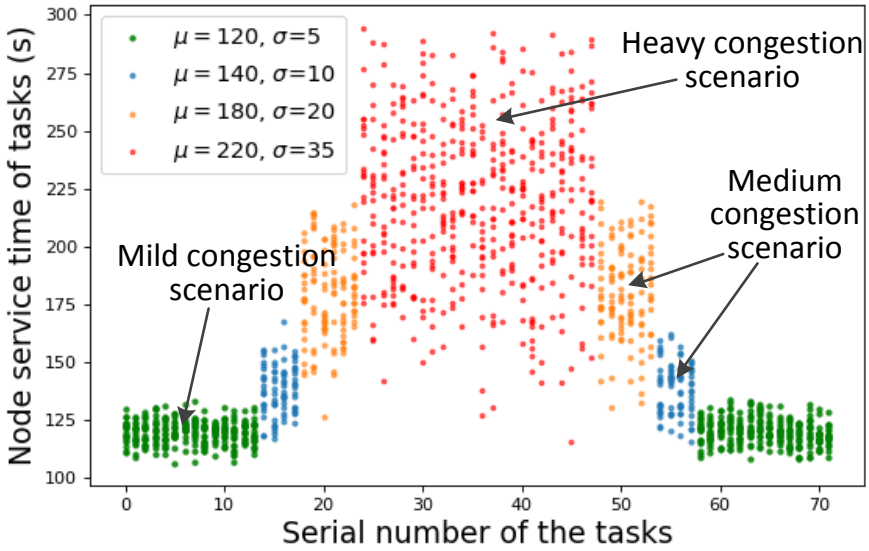

Figure 5.8: *The node service time distribution over time in DRL-GA*

The tasks in instances in Small_Basic_M are distributed over four ship

crane nodes, while those in Big_Basic_M involve six ship crane nodes. Based on the need to evaluate the generality of the proposed method, we also generated additional datasets including Small48T_M, Small96T_M, Big144T_M and Big288T_M, which were used for scalability experiments 1. The Small3WQ_M, Small2WQ_M Big5WQ_M, Big4WQ_M, Big3WQ_M and Big2WQ_M were used for scalability experiments 2. More details about the task datasets are presented in Table 5.1. The details about scalability experiments 1 and 2 were introduced in Section 5.4.2.4.

Table 5.1: Details of datasets used in various experiments.

Datasets	no. of tasks per instance	no. of ship cranes
Small96T_M	96	4
Small_Basic_M	72	4
Small48T_M	48	4
Small3WQ_M	54	3
Small2WQ_M	36	2
Big288T_M	288	6
Big_Basic_M	216	6
Big144T_M	144	6
Big5WQ_M	180	5
Big4WQ_M	144	4
Big3WQ_M	108	3
Big2WQ_M	72	2

5.4.1.3 Hyperparameters

The hyperparameters of DRL are decided based on some tuning experience and some trial runs.

For all the hyperparameters related to GA, we made choices by following recommendations from Mulloorakam & Mathew Nidhiry (2019) and Lavinas et al. (2018), except that the population size is set to 60 to ensure fast computation. This is because the evaluation of fitness of GA individuals is very time-consuming and the total training time of DRL-GA-HH is

far longer than that of DRL-HH (as shown in Table 5.8). Therefore, we chose a relatively small value for the population size to reduce the evolving time of GA.

The concrete settings are described in Table 5.2.

Table 5.2: List of hyperparameters and their values.

Hyperparameter		Value	Description
DRL -related	minibatch size	300	number of pieces of training data (s, a, r, s') fed to the neural network
	initial exploration rate	0.97	in ε-greedy policy, the initial value of ε
	final exploration rate	0.0	in ε-greedy policy, the final value of ε
	replay buffer size	30000	the training data fed to the neural network are sampled from this number of pieces of most recent data
	target network update frequency	300	how often the parameters in the target network are updated
	learning rate	0.0001	the learning rate used in the stochastic gradient decent (SGD)
	optimiser	RMSprop	the optimiser that SGD applied
	reward decay	0.8	the discount factor γ
GA -related	population size	60	
	max generation	100	
	crossover probability	0.9	
	mutation probability	0.08	
	selection method	stochastic tournament, size 6	
	No. of layers of the surrogate networks (SNs)	5	
	No. of neurons in each layers in SNs	S:(72, 200, 300, 200, 1); B:(216, 200, 300, 200, 1)	S is for the no. of neurons of SN associated with Small_Basic_M and B for Big_Basic_M

5.4.2 Experimental Results and Analysis

In this section, we present the sets of experiments that were carried out. First, we describe the purposes and methods of these experiments, and then the experimental results are analysed.

5.4.2.1 Experiments about GA with a Fixed Vision

Since we used GA as an action in our DRL-GA-HH framework, intuitively, we need to compare the proposed method with GA. However, using GA directly is time-consuming (based on previous explanations, see Section 5.3.1 for details). Unlike some other problems, this problem needs to run a solution in the simulator with complex dynamics to obtain its fitness. Also, since it is an online problem, the node service time can change drastically from one scenario to another. It is unlikely to yield good results by directly using GA. Therefore, we made a little adjustment to the GA to suit the online environment, which is the first set of experiments.

In this set of experiments we set the *vision* of a GA to a relatively small fixed value, which we called GA with a fixed vision (GAFV). We have done experiments on the test sets in Small_Basic_M and Big_Basic_M, respectively. Specifically, the fixed values of visions are set to {5, 10, 15, 20} for Small_Basic_M and {10, 20, 30, 40} for Big_Basic_M. Each time a GA is selected, a truck dispatch plan for the next *vision* tasks is obtained. And the GA is not called again until this plan has been completed. As the objective of this problem is to minimise the average total crane waiting time, it has been chosen as the performance evaluation indicator. The results of the experiment are presented in Table 5.3. As can be seen from the table, in the selected vision set, the best performance is obtained when the vision is 10 and 30 for Small_Basic_M and Big_Basic_M, respectively (the values shown in bold in Table 5.3).

Table 5.3: The Performance of GAFV. FV stands for the fixed vision.

Datasets	Average total crane waiting time (s)			
Small_Basic_M	FV = 5	FV = 10	FV = 15	FV = 20
	2471.60	**2435.92**	2463.38	2494.03
Big_Basic_M	FV = 10	FV = 20	FV = 30	FV = 40
	7093.22	7102.12	**7045.62**	7060.88

5.4.2.2 Training of DRL-GA-HH

The experiments conducted consist of two phases: training and testing. We take half of Small_Basic_M and Big_Basic_M as the training set during training. We repeated both training and testing experiments for DRL-GA-HH five times because it is very time-consuming (as shown in Table 5.8). The termination criteria are the same as those introduced in Section 4.3.4. In each episode, all these instances are executed once to obtain training data. Once each instance is finished, we update the parameters of the neural network. The training of the DRL agent converged after approximately 3000 and 4500 episodes for Small_Basic_M and Big_Basic_M, respectively.

5.4.2.3 Testing of DRL-GA-HH

During the testing phase, the other half of Small_Basic_M and Big_Basic_M was chosen as the test set. Similar to Zhang et al. (2022b), we firstly conducted three sets of experiments: performance evaluation experiments, scalability experiment 1 and scalability experiment 2. In these experiments, DRL-GA-HH was compared with the following three existing methods: a manually crafted heuristic (MCH) proposed by Chen et al. (2016), a data-driven genetic programming method (Chen et al., 2020) and DRL-HH proposed in Zhang et al. (2022b). For consistency with Zhang et al. (2022b), the MCH is still used as the baseline. The results of the performance evaluation experiments are shown in Table 5.4.

By comparing the results obtained by GAFV (Table 5.3) and those

obtained by other existing online methods shown in Table 5.4, we have some observations. Firstly, even the best results of GAFV (2435.92 for Small_Basic_M and 7045.62 for Big_Basic_M) were worse than the result of MCH. It is the worst of the existing methods. This shows that using GA directly is not a good solution method for online combinatorial optimisation problems with uncertainty, even if the *vision* of GA was shortened to a relatively smaller value. As long as the *vision* is fixed, the solutions obtained are of low quality.

Secondly, as can be seen from Table 5.4, the proposed DRL-GA-HH yields the best results on both Small_Basic_M and Big_Basic_M datasets compared to other existing methods. Compared to DRL-HH (proposed in Chapter 4), the proposed method shows a greater advantage on the Big_Basic_M dataset (2.08%). On Small_Basic_M, the proposed method only slightly outperforms DRL-HH (1.00%).

Table 5.4: The Performance of DRL-GA-HH in comparison with the manually crafted heuristic (MCH), a data-driven GP method and the DRL-HH method. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets		Small_Basic_M	Big_Basic_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	2381.82/14.39	6980.91/55.22
	Data-driven GP	2340.33/14.54	6894.35/37.72
	DRL-HH	2207.64/13.73	6450.36/47.22
	DRL-GA-HH	2185.38/27.18	6316.22/85.03
Imp% of DRL-GA-HH over MCH		8.25%	9.52%

5.4.2.4 Scalability Experiments

Next, we performed Scalability Experiment 1 and Scalability Experiment 2 to test the generalisation ability of the proposed method. Based on the characteristics of the problem and the realistic requirements of the

port, the generalisation ability was evaluated in two aspects: how well it performed when the number of tasks in the problem instances changed and how well it performed when the number of work queues involved in the instances changed. These two situations are the kind of business that real-world ports need to face on a regular basis. For vessels of the same size moored in the same location, they are almost served by the same ship crane, but they may offer a completely different number of tasks to be completed (some vessels need to load or unload all the containers on board, some only half the ship, and some even less). And when the sizes of the vessels arriving in port change, or when the location of the mooring changes, the number of ship cranes involved changes, i.e. the number of work queues changes.

In Scalability Experiment 1, we fixed the number of work queues involved in the tasks in each instance and changed the number of tasks. As shown in Table 5.1, this group of experiments was conducted on Small48T_M, Small96T_M, Big144T_M and Big288T_M datasets. Taking Big144T_M and Big288T_M as examples, instances in them have the same number of work queues as instances in Big_Basic_M have, but the number of tasks contained in an instance (144 and 288) is smaller and larger than the number of tasks of instances in Big_Basic_M (216), respectively. The results of the experiments are shown in Table 5.5.

In Scalability Experiment 2, we reduced the number of queues involved in each instance. As shown in Table 5.1, the number of work queues is reduced to 3 and 2 (Small3WQ_M and Small2WQ_M) for small datasets. For the big dataset, the number of work queues is reduced to 5, 4, 3 and 2 (Big5WQ_M, Big4WQ_M, Big3WQ_M and Big2WQ_M). According to the actual situation in ports as discussed in (Zhang et al., 2022b), we have not increased the number of queues. First, the expected results will be poor if the training is not conducted on datasets with newly added ship

Table 5.5: Results of scalability experiment 1. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets		Small48T_M	Small96T_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	1293.60/8.80	2959.58/15.88
	Data-driven GP	1287.36/8.07	2914.62/15.33
	DRL-HH	*1264.98/10.58*	2817.57/31.88
	DRL-GA-HH	*1261.20/15.08*	2790.76/26.16
Imp% of DRL-GA-HH over MCH		2.50%	5.70%
Datasets		Big144T_M	Big288T_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	3811.08/23.58	9283.46/67.55
	Data-driven GP	3772.63/29.81	9130.77/80.78
	DRL-HH	3666.26/42.51	8898.68/61.35
	DRL-GA-HH	3625.93/23.39	8803.47/65.41
Imp% of DRL-GA-HH over MCH		4.86%	5.17%

cranes. Second, the number of ship cranes is related to the length of a port's coastline. In a real-world port, the ship cranes are relatively stable in number. If we can train a large enough model that involves all the ship cranes in the port during the training process, then the trained model can still provide high-quality solutions when the number of ship cranes is reduced. The experimental results are shown in Table 5.6.

Also, we conducted Wilcoxon Rank Sum tests to compare the results obtained from DRL-GA-HH and the other three methods in all the experiments including the performance evaluation experiments and the two groups of scalability experiments. All of the obtained p-values are less than 0.03, except for three p-values obtained from the results between DRL-HH and DRL-GA-HH (results about Small48T_M, Small2WQ_M and Big2WQ_M), which are greater than 0.05 (the corresponding results are displayed in italics in Table 5.5 and Table 5.6). In general, the obtained p-values indicate

statistically significant differences between the results obtained from DRL-GA-HH and the other three methods. For the three comparisons between DRL-GA-HH and DRL-HH with p-values less than 0.03, the results indicate that there is no significant improvement in the performance of DRL-GA-HH compared to DRL-HH on datasets Small48T_M, Small2WQ_M and Big2WQ_M. This could be attributed to the simplicity of these datasets (including a low number of tasks or involving a limited number of work queues), which hinders the full utilisation of the advantages offered by DRL-GA-HH.

Table 5.6: Results of scalability experiment 2. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets		Small3WQ_M	Small2WQ_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	1772.07/9.70	1164.35/7.33
	Data-driven GP	1761.58/10.74	1160.03/5.89
	DRL-HH	1714.48/11.83	*1129.30/6.93*
	DRL-GA-HH	1701.90/13.99	*1124.18/13.03*
Imp% of DRL-GA-HH over MCH		3.96%	3.45%

Datasets		Big5WQ_M	Big4WQ_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	5209.85/36.34	4356.79/32.43
	Data-driven GP	5174.03/41.77	4334.01/37.32
	DRL-HH	4983.74/53.93	4196.02/27.20
	DRL-GA-HH	4930.08/50.49	4151.59/37.87
Imp% of DRL-GA-HH over MCH		5.37%	4.71%

Datasets		Big3WQ_M	Big2WQ_M
Average total crane waiting time (s) / StdDev	MCH (baseline)	2850.31/16.40	2010.50/16.47
	Data-driven GP	2837.55/18.15	2002.47/16.61
	DRL-HH	2753.68/23.00	*1950.39/19.88*
	DRL-GA-HH	2732.59/21.61	*1949.38/22.69*
Imp% of DRL-GA-HH over MCH		4.13%	3.04%

5.4.2.5 DRL-GA-HH VS DRL-HH on Previous Datasets

DRL-GA-HH was designed to address the proposed multi-scenario problem, and its performance in the multi-scenario problem has been demonstrated in the aforementioned experiments. It is a new concern whether DRL-GA-HH works specifically for this multi-scenario problem or is more general. Therefore, we have conducted experiments to test the performance of DRL-GA-HH on all the datasets in Table 3.2 used in the previous problem. The results are shown in Table 5.7. The results of DRL-HH were also copied over to provide a direct representation of the performance of both methods. The results of both methods are very close, and the p-value of the Wilcoxon Rank Sum tests is greater than 0.05, which also indicates that there is no statistically significant difference between the two methods on these previous datasets. On Small48T and Small2WQ, the results (shown in italics in Table 5.7) of DRL-GA-HH are worse than those of DRL-HH (P-value is less than 0.01). The reason might be the same as that in the previous scalability experiments, namely, the simplicity of these datasets.

Considering that the training time for DRL-GA-HH is significantly longer than that of DRL-HH, using DRL-HH to solve the problem is a more reasonable choice in non-multi-scenario problems. DRL-GA-HH only performs well in multi-scenario problems.

5.4.2.6 Computational Time Comparison

The time taken to train a machine learning algorithm and the time taken when it is applied in practice are also important aspects in evaluating the algorithm. Therefore, we compared the training time of DRL-GA-HH and DRL-HH and the execution time for the trained models of them (as shown in Table 5.8). Since DRL-GA-HH introduces GA as a component and the evaluation of GA individuals is very time-consuming, the training

Table 5.7: The Performance of DRL-GA-HH in comparison with the DRL-HH method. In x/y, x is the average waiting time, and y is the standard deviation of x.

Datasets		Small_Basic	Big_Basic
Average total crane waiting time (s) / StdDev	DRL-HH	1674.40/14.58	5868.30/66.68
	DRL-GA-HH	1678.35/18.26	5873.46/62.45
Datasets		Small48T	Small96T
Average total crane waiting time (s) / StdDev	DRL-HH	*971.44/10.49*	2130.67/19.79
	DRL-GA-HH	*1015.27/13.62*	2143.33/22.32
Datasets		Big144T	Big288T
Average total crane waiting time (s) / StdDev	DRL-HH	3450.30/32.67	8175.28/54.37
	DRL-GA-HH	3448.25/36.26	8183.77/82.03
Datasets		Small3WQ	Small2WQ
Average total crane waiting time (s) / StdDev	DRL-HH	1313.10/10.84	*869.92/9.48*
	DRL-GA-HH	1317.42/8.18	*890.32/9.38*
Datasets		Big5WQ	Big4WQ
Average total crane waiting time (s) / StdDev	DRL-HH	4646.37/40.24	3774.21/39.71
	DRL-GA-HH	4639.52/41.17	3768.70/42.61
Datasets		Big3WQ	Big2WQ
Average total crane waiting time (s) / StdDev	DRL-HH	2533.54/22.77	1692.46/18.38
	DRL-GA-HH	2542.67/29.18	1701.33/20.23

time of DRL-GA-HH is much longer than that of DRL-HH.

Table 5.8: Comparison of the average training time and average execution time of DRL-HH and DRL-GA-HH.

	Datasets	Small_Basic_M	Big_Basic_M
Average training time (hour) / StdDev	DRL-HH	1.45/0.12	6.62/0.75
	DRL-GA -HH	75.92/8.43	309.88/37.23
Average execution time for an instance (second) / StdDev	DRL-HH	0.169/0.013	0.588/0.062
	DRL-GA -HH	4.65/1.12	12.31/2.86

Although the strategy of shortening the GAs' *vision* reduced the time consumed by the GAs to some extent, the results were still not satisfactory. Therefore, we further added a surrogate network (SN) as introduced in Section 5.3.2. The new framework is called DRL-GA-HH with a surrogate network (DRL-GASN-HH). As expected, the inclusion of the SN has resulted in a significant drop in training time. However, we still need to consider whether the addition of SN causes a degradation in the performance of the algorithm. We therefore also evaluated the performance of DRL-GASN-HH. Table 5.9 presents a comparison of the training time and performance for DRL-GA-HH and DRL-GASN-HH. As can be seen from this table, for the Small_Basic_M dataset, the training time decreases dramatically (82.89%) with the addition of SN, but the average total crane waiting time increases only slightly (0.38%). The average total crane waiting time of DRL-GASN-HH is still less than that of DRL-HH.

5.4.2.7 Analysis of GA Selected Points and Corresponding Visions

Finally, we conducted statistics and analysis of the GA selected decision points and the corresponding *visions*. First, the trained models were used to dispatch the tasks in both the Small_Basic_M and Big_Basic_M

Table 5.9: Comparison of average training time and average total crane waiting time of DRL-GA-HH and DRL-GASN-HH.

	Datasets	Small_Basic_M	Big_Basic_M
Average training time (hour) / StdDev	DRL-GA-HH	75.92/8.43	309.88/37.23
	DRL-GASN -HH	12.99/1.33	50.41/4.92
Imp% of DRL-GASN-HH over DRL-GA-HH		82.89%	83.73%
Average total crane waiting time (s) / StdDev	DRL-GA-HH	2185.38/27.18	6316.22/85.03
	DRL-GASN -HH	2193.72/28.16	6394.23/113.07
Imp% of DRL-GASN-HH over DRL-GA-HH		-0.38%	-1.24%

datasets. They were then grouped into classes based on the points where GA was selected and the corresponding visions. We calculated the proportion of different categories, and the results are presented in Figure 5.9 (for Small_Basic_M) and Figure 5.10 (for Big_Basic_M). The y-axis indicates the proportion of different combinations of GA selected points and visions. And their sum is one. We have found two observations in the figures. Firstly, overall, during the task assignment, the probability of a GA being selected in heavy congestion scenarios (red area in the figures) is relatively lower compared to mild congestion scenarios (green area in the figures). Secondly, most of the time, the corresponding visions when the GA is selected are also shorter in heavy congestion scenarios than in mild congestion scenarios.

The experimental results show that a well-trained model is able to learn the characteristics of the group of GAs with different lengths of visions. It has learned whether to choose or not choose GAs at some decision points. And when choosing a GA, it knows which GA performs better in a specific state. Specifically, in mild congestion scenarios where the service time of the task fluctuates less, GAs are more likely to be selected, and the corre-

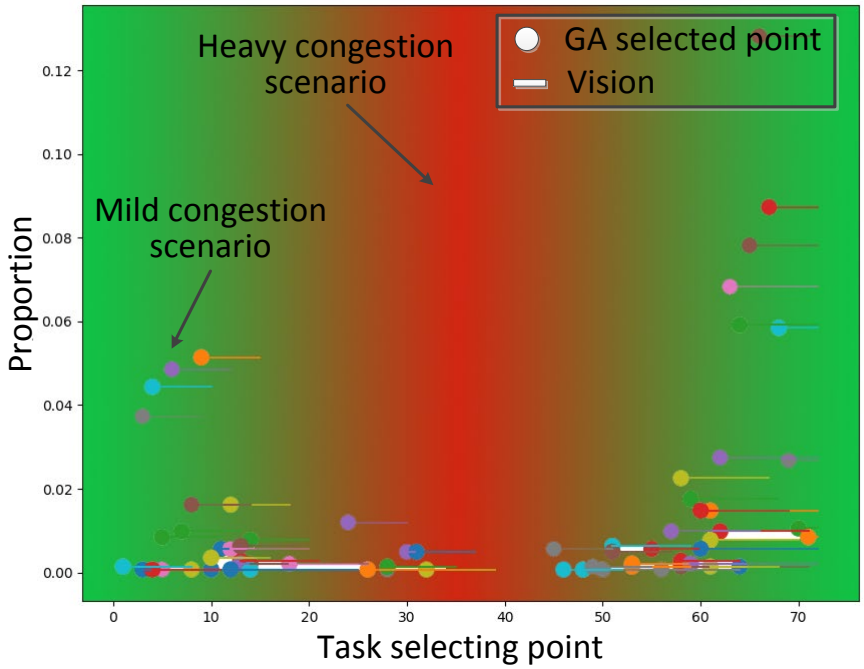

Figure 5.9: *The distribution of the GA selected points and the corresponding visions through the whole dispatching process for Small_Basic_M.*

sponding visions of GAs are larger. The opposite is true in heavy congestion scenarios.

5.5 Summary

In this chapter, we considered a more challenging real-world online internal container truck routing problem. The uncertainty and randomness of external trucks' arrival times and number at the port lead to different levels of congestion at the truck terminal. This has caused a significant impact on the scheduling of internal container trucks. In the problem, we defined different scenarios by the different distributions of service times.

We propose a DRL-based hyper-heuristic framework in which a group of GAs with variable visions serve as actions. GAs can obtain high-quality solutions in mild congestion scenarios since the service time of the trucks is relatively stable in these scenarios (scenarios that are closer to deterministic

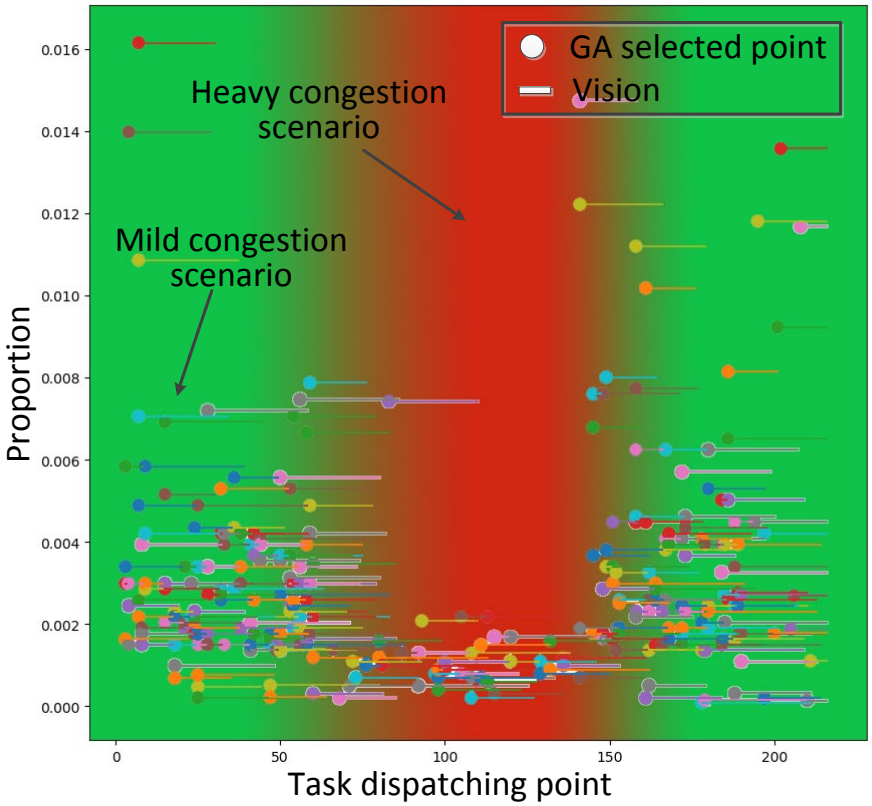

Figure 5.10: *The distribution of the GA selected points and the corresponding visions through the whole dispatching process for Big_Basic_M.*

problems), and DRL is good at dealing with dynamic sequential decision problems; namely, it can perform well when the service time of the trucks varies drastically (scenarios with more dynamics and uncertainties). Due to the DRL Agent's ability to perceive the environment, the hyper-heuristic framework is able to identify different scenarios well and select appropriate actions. Therefore, this framework takes advantage of both DRL which is good at tackling problems with uncertainties and GA which is good at solving deterministic COPs to make appropriate decisions for mixed scenario switches. This is also the reason why this method outperforms other approaches in multi-scenario problems.

In order to adapt to the online environment, we introduced a trick: GA with variable visions. It has two advantages. Firstly, limiting the vision to a relatively small range can effectively reduce the GAs' computation time.

Secondly, the variable visions can also balance the performance of GAs in different scenarios. That is, in mild congestion scenarios, GAs with big visions are preferred, while in heavy congestion scenarios, a good choice is to avoid GA or to choose GAs with very small visions.

To further reduce the computation time of the GA, we also applied surrogate networks to evaluate the fitness of individuals in the GA. Experimental results show that this method significantly reduces the training time of this framework (about 83%) and does not cause significant performance degradation (only about 1%).

Finally, we also found an additional source of training data, namely the additional experience gained by the GAs. At the beginning of training, unlike the experience gained from the relatively blind exploration of DRL, the additional experience gained by GAs is of higher quality. The experimental results show that at the early stage of training, adding this experience to the experience pool can effectively improve the training speed.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis studies the use of DRL-based hyper-heuristics to solve online combinatorial optimisation problems. In an online combinatorial optimisation problem, a problem instance (or part of the key information related to it) is revealed incrementally over time, and decisions must be made at each decision point. Traditional offline algorithms need to know the entire problem instance (or its key information) at the beginning and calculate and determine all future decisions once and for all before the execution of the problem. Due to the lack of some key information for the future, using offline algorithms to solve online problems is impossible to obtain high-quality solutions.

DRL is an advanced artificial intelligence technique proposed by researchers in the machine learning community in recent years. It combines the traditional reinforcement learning framework with the latest deep learning technologies. Its deep neural network has powerful perception capability, and the trial-and-error learning mechanism allows it to quickly improve its capabilities by interacting with the online environment. Besides, it also has the feature of fast response. Therefore, it is widely used to solve various

online sequential decision problems.

Based on these advantages of DRL, we chose to solve an online container truck routing problem with a traditional DRL algorithm. Experimental results show that DRL can provide significantly better performance compared to a manually crafted heuristic (Chen et al., 2016). Also, since DRL can be trained to learn some patterns that exist in the environment and training data, it can obtain good results without the help of problem-related expertise. The manually crafted heuristic, on the other hand, is developed iteratively over a long period by continuously adding and deleting relevant expertise. Although it requires some computational resources (e.g., GPU) for training, DRL has obvious advantages over developing a heuristic with lots of expertise.

The direct use of traditional DRL to solve the online container truck routing problem also has some problems. First, the interpretability of the obtained solutions is relatively poor, which hinders its application in industry. Second, due to the lack of necessary constraints, DRL may select very bad actions, which greatly increases the training time to converge. We then adopted hyper-heuristics to solve the above problems.

Hyper-heuristics are search algorithms that operate on the space of heuristics with the goal of solving a wide range of optimisation problems. We have combined DRL with hyper-heuristics to design a DRL-based hyper-heuristic framework. Based on the manually crafted heuristic, we generated a series of interpretable low-level heuristics. Since these low-level heuristics also take into account some relevant constraints, the DRL Agent no longer selects some extremely bad actions, which indirectly saves training time and improves training speed. The experimental results show that DRL-HH saves about 75% of the training time when obtaining similar results as the direct DRL approach.

In the last sub-study, we solved a more challenging multi-scenario online container truck routing problem, in which the service time of trucks at the nodes shows different distributions in different problem-solving stages. We propose a framework that combines DRL-HH with GA. It can take advantage of DRL's ability to recognise specific environmental states. In some scenarios, it may select GA, and in others, it may select the actions in the previous action set for DRL-HH. The framework performs better overall by taking advantage of both DRL and GA in different scenarios.

6.1.1 Effectiveness and Potential Application to Other Different Problems

The experimental results also indicate that the proposed methods exhibit good generality, performing well on unseen problem instances of the same type as those used for training, as well as maintaining good performance on problem instances with relatively significant variations. This suggests that the proposed methods are capable of adapting to different problem instances to some extent within the same problem domain.

As general frameworks, both DRL-HH and DRL-GA-HH can be applied to solve various COPs. However, it must be acknowledged that there are certain challenges associated with applying the proposed methods to different COPs. This is because in these DRL methods, the problem features included in the state are problem-specific, and the majority of actions in the action set, such as atom actions and low-level heuristics, are also problem-specific.

We may potentially avoid the cumbersome process of feature selection or manual design of problem-specific features in the states by utilising panoramic images that depict the problem environment as the state, similar to using the game screen-shots directly as the state in DRL for playing

Atari games. For example, in the online container truck dispatching problems addressed in this thesis, the panoramic images of the port (as shown in Figure 1.1) at each decision point can be used as the state. Similarly, in the strip-packing problem, the images of the strips being packed can be employed as the state.

This approach may potentially address the cross-problem generality issue of the state design for the proposed methods to some extent. However, the problem of the problem-specific nature of the action set remains a challenge, which could be considered as a potential direction for future research. Once this problem is overcome, it could greatly enhance the generality of the proposed methods.

6.1.2 Concept Drift about the Proposed Methods

Concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. If the concept drift occurs, the induced pattern of past data may not be relevant to new data, resulting in poor predictions and decision outcomes. Concept drift is recognised as a crucial factor in the decreased effectiveness of data-driven information systems such as data-driven early warning systems and data-driven decision support systems. Solving these challenges endows prediction and decision-making with the adaptability in an uncertain environment.

In the online container truck dispatching problems addressed in this thesis, the impact of concept drift is not considered because in the experiments conducted, the distribution of the random variable "crane service time" varied within known ranges. For example, in the first problem, the distribution of crane service time was a fixed one. And in the second multi-scenario problem, the distribution of crane service time varied over time,

but it was only switching between a few fixed distributions.

However, we should not overlook the importance of concept drift in these problems. While the distribution of data involved in real-world ports is relatively stable most of the time, the unpredictability of international environments and market trade can lead to unforeseen changes in ports. If such situations occur, the performance of existing trained models may significantly deteriorate. Therefore, concept drift should receive sufficient attention.

In this thesis, we utilised DRL as an offline training method, where the model is first trained and then used for decision-making. However, in reality, online training is also applicable to DRL, allowing DRL to make decisions while being trained. When the training data is relatively stable, over time, the TD-ERROR in DRL gradually decreases, and the training process converges. But when the training data changes, the TD-ERROR in DRL increases. Over time, with the progress of online training, the TD-ERROR gradually decreases again, and the training process converges once more. Therefore, theoretically, online training DRL inherently possesses the ability to adapt to data changes. It can sensitively detect changes in data distribution through the variation of TD-ERROR and adapt to these changes through online training. Given the significance of concept drift in data-driven decision support systems, investigating whether online training DRL can effectively address the issue of concept drift in real-world COPs can be considered as a potential future research direction.

6.1.3 Interpretability about the Proposed Methods

The proposed methods improve the interpretability of the solutions from two perspectives. Firstly, the human-designed low-level heuristics themselves are interpretable. Secondly, the spectral analysis of states shows

relationships (patterns) between actions and states in the DRL framework. Although these patterns are statistical/fuzzy patterns, rather than discrete, clear-cut rules that can be directly used for operational level decision making, they are still useful for gaining insights and understanding the solutions and the problems.

It also should be noted that while there has been some improvement in interpretability, these methods are still not fully white-box methods, they are more like grey-box approaches, which are a step forward compared to the traditional DRL methods using primitive actions. These methods do not allow the decision-making process of the RL to become transparent and the trained models to be "interpretable".

In general, in this thesis, we have explored the first use of DRL-based hyper-heuristics to address complex online port truck dispatching problems. From both the principle analysis and the experimental results, it can be proved that DRL is well-suited to solve this type of problem. During the research, we have identified problems in solving online container truck dispatching problems when using DRL directly. Then, we have addressed those problems to some extent by combining DRL with hyper-heuristics. We have also explored the interaction of DRL-HH with traditional meta-heuristic (GA) on a more difficult problem with greater uncertainty. We have found some effective tricks that can help GA be better applied in an online environment, such as GA with variable visions and GA with surrogate networks. These tricks can also be applied to other meta-heuristics when used to interact with DRL in online environments.

6.2 Future Work

In this thesis, we attempt to use DRL-based hyper-heuristics to solve online combinatorial optimisation problems. In particular, we have made some progress with the online truck routing problems in ports. In the future, the following work deserves to be explored and improved:

Firstly, for the problem model, we can develop a better problem model for the port environment, in which more details are considered. The current problem model is formulated in (Chen et al., 2016). The model was advanced at that time and had fully considered many key factors involved in the problem, such as the queuing of trucks at quay cranes and yard cranes. However, there is still space for improvement. For example, the position of quay cranes and yard cranes is fixed in that model, while in a real port, the cranes move back and forth within a certain range. It also does not take into account the interaction between yards. For instance, when the queue of one yard is too big, it may cause congestion in other yards next to it.

Secondly, for the methodology, a single-agent DRL algorithm is applied in this thesis, but the multi-agent DRL frameworks are currently the most active ones in academia. They can be used to solve larger-scale problems. In the online truck routing problem, each quay crane or yard crane can be formulated as an agent, and the overall objective can be optimised through competition and cooperation among them.

Thirdly, it is better to consider multiple interrelated problems together from the perspective of the problem. Several operational problems need to be solved in the port: berth allocation, quay crane or yard crane scheduling, yard truck dispatching and scheduling, storage location and space planning problems. The algorithms to solve these problems may interact with each other. Therefore, it is necessary to combine them into joint problems.

However, previous machine learning algorithms can only solve a single task, which hinders the use of such algorithms to solve joint problems. However, the new model Gato, proposed by DeepMind recently, makes this idea more likely to be implemented. Gato is a single transformer that can play Atari games, caption images, chat with people, and control a real robotic arm. It is trained once and uses the same weights to achieve all those tasks. Therefore, by introducing Gato to the operational research society, the joint problems may be well solved.

Bibliography

Ababneh, J. I., & Bataineh, M. H. (2008). Linear phase fir filter design using particle swarm optimization and genetic algorithms. *Digital Signal Processing*, *18*, 657–668.

Akar, E., Topcuoglu, H. R., & Ermis, M. (2014). Hyper-heuristics for online uav path planning under imperfect information. In *European Conference on the Applications of Evolutionary Computation* (pp. 741–752). Springer.

Alanazi, F., & Lehre, P. K. (2014). Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *2014 IEEE congress on evolutionary computation (CEC)* (pp. 2515–2523). IEEE.

Alfi, A., & Modares, H. (2011). System identification and control using adaptive particle swarm optimization. *Applied Mathematical Modelling*, *35*, 1210–1221.

Allen, J. G., Coates, G., & Trevelyan, J. (2013). A hyper-heuristic approach to aircraft structural design optimization. *Structural and Multidisciplinary Optimization*, *48*, 807–819.

Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., & Parkes, A. J. (2016a). Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Information Sciences*, *373*, 476–498.

Asta, S., & Özcan, E. (2014). An apprenticeship learning hyper-heuristic for vehicle routing in hyflex. In *2014 IEEE symposium on evolving and autonomous learning systems (EALS)* (pp. 65–72). IEEE.

Asta, S., Özcan, E., & Curtois, T. (2016b). A tensor based hyper-heuristic for nurse rostering. *Knowledge-based systems*, *98*, 185–199.

Asta, S., Özcan, E., & Parkes, A. J. (2016c). Champ: Creating heuristics via many parameters for online bin packing. *Expert Systems with Applications*, *63*, 208–221.

Azhikodan, A. R., Bhat, A. G., & Jadhav, M. V. (2019). Stock trading bot using deep reinforcement learning. In *Innovations in Computer Science and Engineering* (pp. 41–49). Springer.

Bader-El-Den, M., & Poli, R. (2007). Generating sat local-search heuristics using a gp hyper-heuristic framework. In *International Conference on Artificial Evolution (Evolution Artificielle)* (pp. 37–49). Springer.

Bae, J. W., & Kim, K. H. (2000). A polled dispatching strategy for automated guided vehicles in port container terminals. *Management Science and Financial Engineering*, *6*, 47–67.

Bai, R., Burke, E. K., Gendreau, M., Kendall, G., & McCollum, B. (2007). Memory length in hyper-heuristics: An empirical study. In *2007 IEEE Symposium on Computational Intelligence in Scheduling* (pp. 173–178). IEEE.

Bai, R., Chen, X., Chen, Z.-L., Cui, T., Gong, S., He, W., Jiang, X., Jin, H., Jin, J., Kendall, G. et al. (2023). Analytics and machine learning in vehicle routing research. *International Journal of Production Research*, *61*, 4–30.

Bai, R., Van Woensel, T., Kendall, G., & Burke, E. K. (2013). A new model and a hyper-heuristic approach for two-dimensional shelf space allocation. *4or*, *11*, 31–55.

Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A., Garcia, P. M., Jain, A., Luo, R., Maggiar, A., Narayanaswamy, B., & Ye, C. (2019). Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*, .

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, (pp. 679–684).

Bellman, R. (1966). Dynamic programming. *Science*, *153*, 34–37.

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

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

Bertsekas, D. P. (2007). Neuro-dynamic programming: An overview and recent results. *Operations research proceedings 2006*, (pp. 71–72).

Bian, Z., Li, N., Li, X.-j., & Jin, Z.-h. (2014). Operations scheduling for rail mounted gantry cranes in a container terminal yard. *Journal of Shanghai Jiaotong University (Science)*, *19*, 337–345.

Bidgoli, A. M., & Haghighi, H. (2020). Augmenting ant colony optimization with adaptive random testing to cover prime paths. *Journal of Systems and Software*, *161*, 110495.

Bierwirth, C., & Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, *244*, 675–689.

Bose, J., Reiners, T., Steenken, D., & Voß, S. (2000). Vehicle dispatching at seaport container terminals using evolutionary algorithms. In *Proceedings of the 33rd annual Hawaii international conference on system sciences* (pp. 10–pp). IEEE.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, *64*, 1695–1724.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010a). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics* (pp. 449–468). Springer.

Burke, E. K., Hyde, M., Kendall, G., & Woodward, J. (2010b). A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, *14*, 942–958.

Burke, E. K., Hyde, M., Kendall, G., & Woodward, J. R. (2007a). The scalability of evolved on line bin packing heuristics. In *2007 IEEE Congress on Evolutionary Computation* (pp. 2530–2537). IEEE.

Burke, E. K., Hyde, M. R., & Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature-PPSN IX* (pp. 860–869). Springer.

Burke, E. K., Hyde, M. R., & Kendall, G. (2011). Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, *16*, 406–417.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., & Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence* (pp. 177–201). Springer.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2019). A classification of hyper-heuristic approaches: revisited. In *Handbook of metaheuristics* (pp. 453–477). Springer.

Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2007b). Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (pp. 1559–1565).

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007c). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, *176*, 177–192.

Burke, E. K., Qu, R., & Soghier, A. (2014). Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research*, *218*, 129–145.

Cao, B., Glover, F., & Rego, C. (2015). A tabu search algorithm for cohesive clustering problems. *Journal of Heuristics*, *21*, 457–477.

Cappart, Q., Moisan, T., Rousseau, L.-M., Prémont-Schwarz, I., & Cire, A. A. (2021). Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 3677–3687). volume 35.

Carlo, H. J., Vis, I. F., & Roodbergen, K. J. (2015). Seaside operations in container terminals: literature overview, trends, and research directions. *Flexible Services and Manufacturing Journal*, *27*, 224–262.

Chakrapani, Y., & Rajan, K. S. (2009). Genetic algorithm applied to fractal image compression. *ARPN Journal of Engineering and applied Sciences*, *4*, 53–58.

Chen, J., Bai, R., Dong, H., Qu, R., & Kendall, G. (2016). A dynamic truck dispatching problem in marine container terminal. In *2016 IEEE Symposiums on Computational Intelligence in Scheduling and Network Design (CISND2017), 6-9 December, 2016 Athens, Greece*.

CHEN, L.-l., HUANG, Y.-f., & LE, M.-l. (2016). Yard crane scheduling under multi-type task combined operation. *Journal of Transportation Systems Engineering and Information Technology*, *16*, 205.

Chen, X., Bai, R., Qu, R., Dong, H., & Chen, J. (2020). A data-driven genetic programming heuristic for real-world dynamic seaport container terminal truck dispatching. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8).

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, *32*, 6281–6292.

Chen, Y., Cowling, P., Polack, F., Remde, S., & Mourdjis, P. (2017). Dynamic optimisation of preventative and corrective maintenance schedules for a large scale urban drainage system. *European journal of operational research*, *257*, 494–510.

Chen, Z., Liu, H., Tian, Y., Wang, R., Xiong, P., & Wu, G. (2020). A particle swarm optimization algorithm based on time-space weight for helicopter maritime search and rescue decision-making. *IEEE Access*, *8*, 81526–81541.

Cheng, R., Rodemann, T., Fischer, M., Olhofer, M., & Jin, Y. (2017). Evolutionary many-objective optimization of hybrid electric vehicle control:

From general optimization to preference articulation. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *1*, 97–111.

Cheng, Y.-L., Sen, H.-C., Natarajan, K., Teo, C.-P., & Tan, K.-C. (2005). Dispatching automated guided vehicles in a container terminal. In *Supply chain optimization* (pp. 355–389). Springer.

Chung, Y.-G., Randhawa, S. U., & McDowell, E. D. (1988). A simulation analysis for a transtainer-based container handling facility. *Computers & Industrial Engineering*, *14*, 113–125.

Covic, F. (2018). A literature review on container handling in yard blocks. In *International Conference on Computational Logistics* (pp. 139–167). Springer.

Cowling, P., Kendall, G., & Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling* (pp. 176–190). Springer.

Crawford, B., Soto, R., Monfroy, E., Palma, W., Castro, C., & Paredes, F. (2013). Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization. *Expert Systems with Applications*, *40*, 1690–1695.

Da Fonseca, G. H. G., Santos, H. G., Toffolo, T. Â. M., Brito, S. S., & Souza, M. J. F. (2016). Goal solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research*, *239*, 77–97.

Dai, H., Dai, B., & Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning* (pp. 2702–2711). PMLR.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management science*, *6*, 80–91.

Das, G., Pattnaik, P. K., & Padhy, S. K. (2014). Artificial neural network trained by particle swarm optimization for non-linear channel equalization. *Expert Systems with Applications*, *41*, 3491–3496.

Deb, A., Roy, J. S., & Gupta, B. (2014). Performance comparison of differential evolution, particle swarm optimization and genetic algorithm in the design of circularly polarized microstrip antennas. *IEEE transactions on antennas and propagation*, *62*, 3920–3928.

Denzinger, J., & Fuchs, M. (1996). High performance atp systems by combining several ai methods, .

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., & Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research* (pp. 170–181). Springer.

Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*, .

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, *285*, 405–428.

Drigo, M. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, *26*, 1–13.

Drori, I., Kharkar, A., Sickinger, W. R., Kates, B., Ma, Q., Ge, S., Dolev, E., Dietrich, B., Williamson, D. P., & Udell, M. (2020). Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 19–24). IEEE.

Eden, A. H., Steinhart, E., Pearce, D., & Moor, J. H. (2012). Singularity hypotheses: an overview. *Singularity hypotheses*, (pp. 1–12).

Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, *3*, 124–141.

Fan, H., Xiong, H., & Goh, M. (2021). Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints. *Computers & Operations Research*, *134*, 105401.

Fang, H.-L., Ross, P., & Corne, D. (1993). *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. Citeseer.

Fisher, R. D., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial scheduling* (pp. 225–251).

Frank, M. J., & Claus, E. D. (2006). Anatomy of a decision: striato-orbitofrontal interactions in reinforcement learning, decision making, and reversal. *Psychological review*, *113*, 300.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, *16*, 31–61.

Gharehgozli, A. H., Yu, Y., de Koster, R., & Udding, J. T. (2014). An exact method for scheduling a yard crane. *European Journal of Operational Research*, *235*, 431–447.

Girgis, M. R., Mahmoud, T. M., Abdullatif, B. A., & Rabie, A. M. (2014). Solving the wireless mesh network design problem using genetic algorithm and simulated annealing optimization methods. *International Journal of Computer Applications*, *96*.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, *13*, 533–549.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

Godinho Filho, M., Barco, C. F., & Tavares Neto, R. F. (2014). Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (fms): a literature survey, classification and analysis. *Flexible Services and Manufacturing Journal*, *26*, 408–431.

Gomez, J. C., & Terashima-Marín, H. (2018). Evolutionary hyper-heuristics for tackling bi-objective 2d bin packing problems. *Genetic Programming and Evolvable Machines*, *19*, 151–181.

Grunow, M., Günther, H.-O., & Lehmann, M. (2007). Strategies for dispatching agvs at automated seaport container terminals. In *Container terminals and cargo systems* (pp. 155–178). Springer.

Gupta, D., & Maravelias, C. T. (2016). On deterministic online scheduling: Major considerations, paradoxes and remedies. *Computers & Chemical Engineering*, *94*, 312–330.

Gutierrez-Rodríguez, A. E., Ortiz-Bayliss, J. C., Rosales-Pérez, A., Amaya-Contreras, I. M., Conant-Pablos, S. E., Terashima-Marín, H., & Coello, C. A. C. (2017). Applying automatic heuristic-filtering to improve hyper-heuristic performance. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2638–2644). IEEE.

Hao, X., & Liu, J. (2017). A multiagent evolutionary algorithm with direct and indirect combined representation for constraint satisfaction problems. *Soft Computing*, *21*, 781–793.

Hart, E., Ross, P., & Nelson, J. (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, *6*, 61–80.

Hart, E., & Sim, K. (2016). A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary computation*, *24*, 609–635.

Hartmann, S. (2004). A general framework for scheduling equipment and manpower at container terminals. *OR Spectrum*, *26*, 51–74.

Herroelen, W., De Reyck, B., & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, *25*, 279–302.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press. Second edition, 1992.

Homayouni, S. M., & Tang, S. H. (2013). Multi objective optimization of coordinated scheduling of cranes and vehicles at container terminals. *Mathematical Problems in Engineering*, *2013*.

Hu, X., Guo, J., & Zhang, Y. (2019). Optimal strategies for the yard truck scheduling in container terminal with the consideration of container clusters. *Computers & Industrial Engineering*, *137*, 106083.

Inthachot, M., Boonjing, V., Intakosum, S. et al. (2016). Artificial neural network and genetic algorithm hybrid intelligence for predicting thai stock price index trend. *Computational intelligence and neuroscience*, *2016*.

Jackson, W. G., Özcan, E., & John, R. I. (2018). Move acceptance in local search metaheuristics for cross-domain search. *Expert Systems with Applications*, *109*, 131–151.

Jacobsen-Grocott, J., Mei, Y., Chen, G., & Zhang, M. (2017). Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1948–1955). IEEE.

Jaillet, P., & Wagner, M. R. (2008a). Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Operations research*, *56*, 745–757.

Jaillet, P., & Wagner, M. R. (2008b). Online vehicle routing problems: A survey. *The Vehicle Routing Problem: Latest Advances and New Challenges*, (pp. 221–237).

Kampouridis, M., Alsheddy, A., & Tsang, E. (2013). On the investigation of hyper-heuristics on a financial forecasting problem. *Annals of Mathematics and Artificial Intelligence*, *68*, 225–246.

Kaveshgar, N., & Huynh, N. (2015). Integrated quay crane and yard truck scheduling for unloading inbound containers. *International Journal of Production Economics*, *159*, 168–177.

Ke, J.-C. (2003). The analysis of a general input queue with n policy and exponential vacations. *Queueing systems*, *45*, 135–160.

Keller, R. E., & Poli, R. (2007a). Cost-benefit investigation of a genetic-programming hyperheuristic. In *International conference on artificial evolution (evolution artificielle)* (pp. 13–24). Springer.

Keller, R. E., & Poli, R. (2007b). Linear genetic programming of parsimonious metaheuristics. In *2007 IEEE Congress on Evolutionary Computation* (pp. 4508–4515). IEEE.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (pp. 1942–1948). IEEE volume 4.

Keogh, E. J., & Mueen, A. (2017). Curse of dimensionality. *Encyclopedia of machine learning and data mining*, *2017*, 314–315.

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

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., & Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*. volume 30.

Khan, S. A., & Engelbrecht, A. P. (2012). A fuzzy particle swarm optimization algorithm for computer communication network topology design. *Applied Intelligence*, *36*, 161–177.

Kheiri, A., & Keedwell, E. (2017). A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. *Evolutionary computation*, *25*, 473–501.

Kieffer, E., Danoy, G., Brust, M. R., Bouvry, P., & Nagih, A. (2019). Tackling large-scale and combinatorial bi-level problems with a genetic programming hyper-heuristic. *IEEE Transactions on Evolutionary Computation*, *24*, 44–56.

Kim, K. Y., & Kim, K. H. (2003). Heuristic algorithms for routing yard-side equipment for minimizing loading times in container terminals. *Naval Research Logistics (NRL)*, *50*, 498–514.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*, 671–680.

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

Korošec, P., & Eftimov, T. (2020). Insights into exploration and exploitation power of optimization algorithm using dsctool. *Mathematics*, *8*, 1474.

Koulinas, G., & Anagnostopoulos, K. (2013). A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities. *Automation in Construction*, *31*, 169–175.

Koulinas, G., Kotsikas, L., & Anagnostopoulos, K. (2014). A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. *Information Sciences*, *277*, 680–693.

Lavinas, Y., Aranha, C., Sakurai, T., & Ladeira, M. (2018). Experimental analysis of the tournament size on genetic algorithms. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 3647–3653). IEEE.

Lee, D.-H., Cao, J., & Shi, Q. (2009). Synchronization of yard truck scheduling and storage allocation in container terminals. *Engineering Optimization*, *41*, 659–672.

Lehre, P. K., & Özcan, E. (2013). A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *Proceedings of the twelfth workshop on Foundations of genetic algorithms XII* (pp. 97–104).

Li, D., Li, M., Meng, X., & Tian, Y. (2014). A hyperheuristic approach for intercell scheduling with single processing machines and batch processing machines. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*, 315–325.

Li, D., Zhan, R., Zheng, D., Li, M., & Kaku, I. (2015). A hybrid evolutionary hyper-heuristic approach for intercell scheduling considering transportation capacity. *IEEE Transactions on Automation Science and Engineering*, *13*, 1072–1089.

Li, J., & Kendall, G. (2015). A hyperheuristic methodology to generate adaptive strategies for games. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*, 1–10.

Lian, Y., Oyama, A., & Liou, M.-S. (2010). Progress in design optimization using evolutionary algorithms for aerodynamic problems. *Progress in Aerospace Sciences*, *46*, 199–223.

Liang, C.-J., Chen, M., Gen, M., & Jo, J. (2014). A multi-objective genetic algorithm for yard crane scheduling problem with multiple work lines. *Journal of Intelligent Manufacturing*, *25*, 1013–1024.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, .

Lim, A., Rodrigues, B., & Xu, Z. (2007). A m-parallel crane scheduling problem with a non-crossing constraint. *Naval Research Logistics (NRL)*, *54*, 115–127.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks*. Technical Report Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lodi, A., & Zarpellon, G. (2017). On learning and branching: a survey. *Top*, *25*, 207–236.

López-Camacho, E., Terashima-Marin, H., Ross, P., & Ochoa, G. (2014). A unified hyper-heuristic framework for solving bin packing problems. *Expert Systems with Applications*, *41*, 6876–6889.

Lu, H., Zhang, X., & Yang, S. (2019). A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*.

Luo, J., Vanhoucke, M., Coelho, J., & Guo, W. (2022). An efficient genetic programming approach to design priority rules for resource-constrained project scheduling problem. *Expert Systems with Applications*, *198*, 116753.

Luo, J., Wu, Y., & Mendes, A. B. (2016). Modelling of integrated vehicle scheduling and container storage problems in unloading process at an automated container terminal. *Computers & Industrial Engineering*, *94*, 32–44.

MacLachlan, J., Mei, Y., Branke, J., & Zhang, M. (2019). Genetic programming hyper-heuristics with vehicle collaboration for uncertain capacitated arc routing problems. *Evolutionary Computation*, (pp. 1–31).

MacLachlan, J., Mei, Y., Zhang, F., & Zhang, M. (2022). Genetic programming for vehicle subset selection in ambulance dispatching. In *2022 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8). IEEE.

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

Mei, Y., Tang, K., & Yao, X. (2010). Capacitated arc routing problem in uncertain environments. In *IEEE Congress on Evolutionary Computation* (pp. 1–8). IEEE.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*, 1087–1092.

Minsky, M. L. (1954). *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. Princeton University.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937). PMLR.

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

Mulloorakam, A. T., & Mathew Nidhiry, N. (2019). Combined objective optimization for vehicle routing using genetic algorithm. *Materials Today: Proceedings*, *11*, 891–902. International Multi-Conference on Computing, Communication, Electrical Nanotechnology (I2CN-2K'18).

Narasimhan, A., & Palekar, U. S. (2002). Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation science*, *36*, 63–78.

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

Ng, W., & Mak, K. (2005). An effective heuristic for scheduling a yard crane to handle jobs with different ready times. *Engineering optimization*, *37*, 867–877.

Nguyen, V. D., & Kim, K. H. (2009). A dispatching method for automated lifting vehicles in automated port container terminals. *Computers & Industrial Engineering*, *56*, 1002–1020.

Niu, B., Xie, T., Tan, L., Bi, Y., & Wang, Z. (2016). Swarm intelligence algorithms for yard truck scheduling and storage allocation problems. *Neurocomputing*, *188*, 284–293.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, *13*, 387–410.

Ortiz-Bayliss, J. C., Terashima-Marín, H., & Conant-Pablos, S. E. (2013). Learning vector quantization for variable ordering in constraint satisfaction problems. *Pattern Recognition Letters*, *34*, 423–432.

Ortiz-Bayliss, J. C., Terashima-Marín, H., & Conant-Pablos, S. E. (2016). Combine and conquer: an evolutionary hyper-heuristic approach for solving constraint satisfaction problems. *Artificial Intelligence Review*, *46*, 327–349.

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, *12*, 417–431.

Özcan, E., & Parkes, A. J. (2011). Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (pp. 2011–2018).

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Park, J., Mei, Y., Nguyen, S., Chen, G., & Zhang, M. (2018). An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing*, *63*, 72–86.

Peng, B., Wang, J., & Zhang, Z. (2019). A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *International Symposium on Intelligence Computation and Applications* (pp. 636–650). Springer.

Pillay, N., & Qu, R. (2018). Selection constructive hyper-heuristics. In *Hyper-heuristics: theory and applications* (pp. 7–16). doi:10.1007/978-3-319-96514-7_2.

Pillay, N., & Qu, R. (2021). *Rigorous Performance Analysis of Hyper-Heuristics*. Springer Natural Computing Series.

Pour, S. M., Drake, J. H., & Burke, E. K. (2018). A choice function hyper-heuristic framework for the allocation of maintenance tasks in danish railways. *Computers & Operations Research*, *93*, 15–26.

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, *14*, 130–154.

Qu, R., & Burke, E. K. (2009). Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, *60*, 1273–1285.

Qu, R., Pham, N., Bai, R., & Kendall, G. (2015). Hybridising heuristics within an estimation distribution algorithm for examination timetabling. *Applied Intelligence*, *42*, 679–693.

Rahimian, E., Akartunalı, K., & Levine, J. (2017). A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems. *European Journal of Operational Research*, *258*, 411–423.

Ramírez-Nafarrate, A., González-Ramírez, R. G., Smith, N. R., Guerra-Olivares, R., & Voß, S. (2017). Impact on yard efficiency of a truck appointment system for a port terminal. *Annals of Operations Research*, *258*, 195–216.

Rios, B. H. O., Xavier, E. C., Miyazawa, F. K., Amorim, P., Curcio, E., & Santos, M. J. (2021). Recent dynamic vehicle routing problems: A survey. *Computers & Industrial Engineering*, *160*, 107604.

Ritzinger, U., Puchinger, J., & Hartl, R. F. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, *54*, 215–231.

Rosales-Pérez, A., Gutiérrez-Rodríguez, A. E., Ortiz-Bayliss, J. C., Terashima-Marín, H., & Coello, C. A. C. (2017). Evolutionary multi-label hyper-heuristic design. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2622–2629). Ieee.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2013). Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, *17*, 840–861.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2014a). Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, *19*, 309–325.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2014b). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE transactions on cybernetics*, *45*, 217–228.

Sabar, N. R., Turky, A., Song, A., & Sattar, A. (2017). Optimising deep belief networks by hyper-heuristic approach. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2738–2745). Ieee.

Sait, S. M., & Arafeh, A. M. (2014). Cell assignment in hybrid cmos/nanodevices architecture using tabu search. *Applied intelligence*, *40*, 1–12.

Salhi, A., & Vázquez Rodríguez, J. A. (2014). Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. *Memetic Computing*, *6*, 77–84.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, .

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

Schwientek, A. K., Lange, A.-K., & Jahn, C. (2017). Literature classification on dispatching of container terminal vehicles. In *Digitalization in Maritime and Sustainable Logistics: City Logistics, Port Logistics and Sustainable Supply Chain Management in the Digital Age. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 24* (pp. 3–36). Berlin: epubli GmbH.

Senington, R., & Duke, D. (2013). De composing metaheuristic operations. In *Implementation and Application of Functional Languages: 24th International Symposium, IFL 2012, Oxford, UK, August 30-September 1, 2012, Revised Selected Papers 24* (pp. 224–239). Springer.

Servranckx, T., & Vanhoucke, M. (2019). A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs. *European Journal of Operational Research*, *273*, 841–860.

Seymour, B., O'Doherty, J. P., Dayan, P., Koltzenburg, M., Jones, A. K., Dolan, R. J., Friston, K. J., & Frackowiak, R. S. (2004). Temporal difference models describe higher-order learning in humans. *Nature*, *429*, 664–667.

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

Sim, K., Hart, E., & Paechter, B. (2015). A lifelong learning hyper-heuristic method for bin packing. *Evolutionary computation*, *23*, 37–67.

Song, H.-B., & Lin, J. (2021). A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. *Swarm and Evolutionary Computation*, *60*, 100807.

Soria-Alcaraz, J. A., Ochoa, G., Sotelo-Figeroa, M. A., & Burke, E. K. (2017). A methodology for determining an effective subset of heuristics in selection hyper-heuristics. *European Journal of Operational Research*, *260*, 972–983.

Soria-Alcaraz, J. A., Ochoa, G., Swan, J., Carpio, M., Puga, H., & Burke, E. K. (2014). Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research*, *238*, 77–86.

Sosa-Ascencio, A., Ochoa, G., Terashima-Marin, H., & Conant-Pablos, S. E. (2016). Grammar-based generation of variable-selection heuristics for constraint satisfaction problems. *Genetic Programming and Evolvable Machines*, *17*, 119–144.

Swan, J., Adriaensen, S., Brownlee, A. E., Hammond, K., Johnson, C. G., Kheiri, A., Krawiec, F., Merelo, J. J., Minku, L. L., Özcan, E. et al. (2022). Metaheuristics "in the large". *European Journal of Operational Research*, *297*, 393–406.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation* volume 74. John Wiley & Sons.

Tan, B., Ma, H., & Mei, Y. (2018). A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds. In *Australasian Joint Conference on Artificial Intelligence* (pp. 146–152). Springer.

Tan, B., Ma, H., Mei, Y., & Zhang, M. (2020). A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds. *IEEE Transactions on Cloud Computing*, .

Taner, M. E., Kulak, O., & Koyuncuoğlu, M. U. (2014). Layout analysis affecting strategic decisions in artificial container terminals. *Computers & Industrial Engineering*, *75*, 1–12.

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, *54*, 453–473.

Thomas, J., & Chaudhari, N. S. (2014). Design of efficient packing system using genetic algorithm based on hyper heuristic approach. *Advances in Engineering Software*, *73*, 45–52.

Thorndike, E. L. (1927). The law of effect. *The American journal of psychology*, *39*, 212–222.

Thrun, S., & Littman, M. L. (2000). Reinforcement learning: an introduction. *AI Magazine*, *21*, 103–103.

Tsai, C.-W., Chang, W.-L., Hu, K.-C., & Chiang, M.-C. (2017). An improved hyper-heuristic clustering algorithm for wireless sensor networks. *Mobile Networks and Applications*, *22*, 943–958.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Van Nguyen, N., Lee, J., Tyan, M., & Kim, S. (2015). Repetitively enhanced neural networks method for complex engineering design optimisation problems. *The Aeronautical Journal*, *119*, 1253–1270.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Vincent, F. Y., Redi, A. P., Hidayat, Y. A., & Wibowo, O. J. (2017). A simulated annealing heuristic for the hybrid vehicle routing problem. *Applied Soft Computing*, *53*, 119–132.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *Advances in neural information processing systems*, *28*.

Vis, I. F., & Carlo, H. J. (2010). Sequencing two cooperating automated stacking cranes in a container terminal. *Transportation Science*, *44*, 169–182.

Wang, L., Pan, Z., & Wang, J. (2021). A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation*, *1*, 257–270.

Wang, S., Mei, Y., Park, J., & Zhang, M. (2019). Evolving ensembles of routing policies using genetic programming for uncertain capacitated arc routing problem. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1628–1635). IEEE.

Wauters, T., Vancroonenburg, W., & Vanden Berghe, G. (2012). A guide-and-observe hyper-heuristic approach to the eternity ii puzzle. *Journal of Mathematical Modelling and Algorithms*, *11*, 217–233.

Wu, X., Consoli, P., Minku, L., Ochoa, G., & Yao, X. (2016). An evolutionary hyper-heuristic for the software project scheduling problem. In *International Conference on Parallel Problem Solving from Nature* (pp. 37–47). Springer.

Xing, L.-N., Chen, Y.-W., Wang, P., Zhao, Q.-S., & Xiong, J. (2010). A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, *10*, 888–896.

Yang, C., Peng, S., Jiang, B., Wang, L., & Li, R. (2014). Hyper-heuristic genetic algorithm for solving frequency assignment problem in td-scdma. In *Proceedings of the companion publication of the 2014 annual conference on genetic and evolutionary computation* (pp. 1231–1238).

Yska, D., Mei, Y., & Zhang, M. (2018). Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In *Genetic Programming: 21st European Conference, EuroGP 2018, Parma, Italy, April 4-6, 2018, Proceedings 21* (pp. 306–321). Springer.

Yu, H., Ni, J., & Zhao, J. (2013). Acosampling: An ant colony optimization-based undersampling method for classifying imbalanced dna microarray data. *Neurocomputing*, *101*, 309–318.

Zehendner, E., & Feillet, D. (2014). Benefits of a truck appointment system on the service quality of inland transport modes at a multimodal container terminal. *European Journal of Operational Research*, *235*, 461–469.

Zehendner, E., Feillet, D., & Jaillet, P. (2017). An algorithm with performance guarantee for the online container relocation problem. *European Journal of Operational Research*, *259*, 48–62.

Zeng, Q., Yang, Z., & Hu, X. (2011). A method integrating simulation and reinforcement learning for operation scheduling in container terminals. *Transport*, *26*, 383–393.

Zhang, F., Chen, Y., & Chen, Y. (2018). Evolving constructive heuristics for agile earth observing satellite scheduling problem with genetic programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–7). IEEE.

Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2021a). Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling. *IEEE Transactions on Cybernetics*, *52*, 10515–10528.

Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2022a). Instance rotation based surrogate in genetic programming with brood recombination for dynamic job shop scheduling. *IEEE Transactions on Evolutionary Computation*, .

Zhang, F., Mei, Y., Nguyen, S., & Zhang, M. (2021b). Correlation coefficient-based recombinative guidance for genetic programming hyper-heuristics in dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, *25*, 552–566.

Zhang, R., & Wu, C. (2011). A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardinessobjective. *Computers & Operations Research*, *38*, 854–867.

Zhang, Y., Bai, R., Qu, R., Tu, C., & Jin, J. (2022b). A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, *300*, 418–427.

Zhang, Y., Mi, W., Chang, D., Yan, W., & Shi, L. (2007). An optimization model for intra-bay relocation of outbound container on container yards. In *2007 IEEE International Conference on Automation and Logistics* (pp. 776–781). IEEE.

Zhao, J., Zhou, Y.-W., & Wahab, M. (2016). Joint optimization models for shelf display and inventory control considering the impact of spatial relationship on demand. *European Journal of Operational Research*, *255*, 797–808.

Zhu, L., Lin, J., Li, Y.-Y., & Wang, Z.-J. (2021). A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Knowledge-Based Systems*, *225*, 107099. URL: https://www.sciencedirect.com/science/article/pii/S0950705121003622. doi:https://doi.org/10.1016/j.knosys.2021.107099.

Appendix A

List of Abbreviations

ACA	Ant Colony Algorithm
AIS	Artificial Immune System
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
COP	Combinatorial Optimisation Problems
CSP	Constraint Satisfaction Problem
CST	Crane Service Time
DL	Deep Learning
DQN	Deep Q-Network
DDQN	Double DQN
D3QN	Dueling Double DQN
DRL	Deep Reinforcement Learning
ET	External Container Truck
IT	Internal Container Truck
FC	Fully Connection
FCFS	First Come First Served
GA	Genetic Algorithm
HH	Hyper-heuristics
ILS	Iterated Local Search
LLH	low-level heuristic
ML	Machine Learning
OR	Operation Research
PCA	Principal Component Analysis
PSO	Particle Swarm Optimisation
QC	Quay Crane
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SA	Simulated Annealing
SVM	Support Vector Machine
TSP	Traveling Salesman Problems
VNS	variable neighbourhood search
YC	Yard Crane
YT	Yard Truck