A Deep Reinforcement Learning Based Hyper-heuristic for Combinatorial Optimisation with Uncertainties

Article *in* European Journal of Operational Research · October 2021

DOI: 10.1016/j.ejor.2021.10.032

Ruibin Bai
University of Nottingham Ningbo China
93 PUBLICATIONS **1,416** CITATIONS
SEE PROFILE

Rong Qu
University of Nottingham
211 PUBLICATIONS **6,883** CITATIONS
SEE PROFILE

Jiahuan Jin
University of Nottingham Ningbo China
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

A Deep Reinforcement Learning Based Hyper-heuristic for Combinatorial Optimisation with Uncertainties

Yuchang Zhang[a], Ruibin Bai[a,*], Rong Qu[b], Chaofan Tu[a], Jiahuan Jin[a]

[a] *School of Computer Science, University of Nottingham Ningbo China, Ningbo, 315100, China*
[b] *School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK*

Abstract

In the past decade, considerable advances have been made in the field of computational intelligence and operations research. However, the majority of these optimisation approaches have been developed for deterministically formulated problems, the parameters of which are often assumed perfectly predictable prior to problem-solving. In practice, this strong assumption unfortunately contradicts the reality of many real-world problems which are subject to different levels of uncertainties. The solutions derived from these deterministic approaches can rapidly deteriorate during execution due to the over-optimisation without explicit consideration of the uncertainties. To address this research gap, a deep reinforcement learning based hyper-heuristic framework is proposed in this paper. The proposed approach enhances the existing hyper-heuristics with a powerful data-driven heuristic selection module in the form of deep reinforcement learning on parameter-controlled low-level heuristics, to substantially improve their handling of uncertainties while optimising across various problems. The performance and practicality of the proposed hyper-heuristic approach have been assessed on two combinatorial optimisation problems: a real-world container terminal truck routing problem with uncertain service times and the well-known online 2D strip packing problem. The experimental results demonstrate its superior performance compared to existing solution methods for these problems. Finally, the increased interpretability of the proposed deep reinforcement learning hyper-heuristic

---

[*] Corresponding author

Email addresses: `Yuchang.Zhang@nottingham.edu.cn` (Yuchang Zhang), `Ruibin.Bai@nottingham.edu.cn` (Ruibin Bai), `Rong.Qu@nottingham.ac.uk` (Rong Qu), `Chaofan.Tu@nottingham.edu.cn` (Chaofan Tu), `Jiahuan.Jin@nottingham.edu.cn` (Jiahuan Jin)

has been exhibited in comparison with the conventional deep reinforcement learning methods.

Keywords: Transportation, 2D Packing, Hyper-heuristics, Deep Reinforcement Learning, Container Truck Routing

1. Introduction

Research on combinatorial optimisation problems is of vital importance because of their broad applications in various real-world scenarios, including transportation, logistics, production, resource allocation, timetabling, digital services, finance and numerous other domains. Despite the recent advances, existing studies have largely focused on algorithmic development on the deterministic variant of the problems, in which the parameters to define the problems are assumed to be known in advance. It is not always possible to acquire accurate information of all the problem characteristics in most real-life scenarios. Instead, the real values of problem parameters are often sequentially revealed over time during decision making. In such situations, solutions generated as a `priori` often encounter various issues such as inferior service quality, increased costs, and infeasibile solutions, all of which would lead to substantial losses. For example, in Uncertain Capacitated Arc Routing Problem studied in Mei et al. (2010), offline methods showed to generate infeasible solutions when the actual demand of a task exceeds the remaining capacity of the vehicle, and consequently, the vehicle cannot fully serve the task as expected by the offline methods.

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.

One of the possible methods for this purpose is the hyper-heuristics, originally proposed as a high-level search paradigm that aims to achieve an increased generality in performance across different problem instances and problem domains. Different from the metaheuristic approaches that operate directly in the space of solutions, hyper-heuristics search in the heuristic space, the landscape of which is considered less problem-dependent than that of the solution search space (Burke et al., 2013; Pillay & Qu, 2018a). The idea of hyper-heuristic originated from Fisher & Thompson (1963) in the 1960s before the term was formally introduced by Cowling et al. (2000). Hyper-heuristics have been studied to solve various combinatorial optimisation problems, such as timetabling (Soria-Alcaraz et al., 2014), production scheduling (Rahimian et al., 2017) and vehicle routing (Ahmed et al., 2019), and potentially can be a promising candidate framework to address online optimisation problems.

This work is motivated by the demand of advanced algorithms for solving challenging online combinatorial optimisation problems, 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. The DDQN offers good performance and training stability, and it has been used to solve problems in different fields, such as edge computing (Chen et al., 2018) and recommendation systems (Zheng et al., 2018).

Our proposed method extends the previous research in the followings: 1) different from the DRL methods in Mnih et al. (2013), Mnih et al. (2015) and Van Hasselt et al. (2016), which are designed for solving gaming problems without obvious mathematical formulations, our DRL-HH is applied to classical combinatorial optimisation problems which have rich literature, especially for their offline version of the problems. 2) compared with previous hyper-heuristics, the simple Q-learning mechanism is replaced with a much powerful DDQN, which provides much better ability to handle high dimensionality data; 3) the proposed framework is now applicable for online combinatorial optimisation problems while the previous Q-learning based hyper-heuristics are designed for offline optimisation.

Compared to online genetic programming (GP) hyper-heuristic methods (e.g. MacLachlan et al. (2019) and Chen et al. (2020)), our proposed method could offer better performance 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). A GP based method will need a set of pre-defined features as well as a set of customised operators. Compared to supervised learning methods heavily used in data predictions and pattern recognition, our 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 paper is organised as follows: Section 2 reviews hyper-heuristics and related problems. Section 3 describes the proposed DRL hyper-heuristic framework. In Sections 4 and 5, the proposed framework is evaluated by solving two considerably different combinatorial optimisation problems with uncertainties, followed with discussions of the experimental design and results analysis. Finally, Section 6 concludes the paper.

2. Literature Review

Boosted by the increased computing power and more sophisticated optimisation algorithms that are now capable of exploiting more advanced problem structures, possibilities exist to tackle optimisation problems of considerably larger sizes and to obtain solutions of substantially higher quality in terms of stated objectives. Among these methods, hyper-heuristics have been explored with the primary goal of raising the generality of the performance of optimisation methods across different problem domains and instances. In broad terms, hyper-heuristics can be defined as "heuristics to select or generate heuristics" (Burke et al., 2010). The selection hyper-heuristic, being the focus of this paper, can be further divided into two types: construction-based and perturbation-based. Selection perturbative hyper-heuristics start from a complete solution and then iteratively select a low-level heuristic among a set of perturbative low-level heuristics (often neighbourhood operators) that can efficiently search the solution space and rapidly improve the incumbent solution. Selection constructive hyper-heuristics learn to select among a set of constructive low-level heuristics at each point of solution construction, incrementally building a complete solution to a given optimisation problem (Pillay & Qu, 2018b).

Since the initial introduction in 2000, hyper-heuristics have received progressively increasing research attention, especially in the past few years. The number of yearly publications on hyper-heuristics is close to 100 in the Web of Science database. Among these publications, the majority of existing literature on selection hyper-heuristics is perturbative hyper-heuristics, which operate on sets of perturbative low-level heuristics searching upon complete solutions for optimisation problems (Bai et al., 2012; Drake et al., 2019). Studies have been conducted exploring different pairwise combinations of selection and move acceptance (Burke et al., 2013). A Modified Choice Function was chosen by Choong et al. (2019) to solve TSP by choosing between low-level heuristics within a swarm-based evolutionary algorithm. In Zamli et al. (2016), a tabu search hyper-heuristic was utilised for combinatorial interaction testing, choosing among four low-level metaheuristics for t-way test suite generation, and obtaining good results for problems with up to 6-way interactions. Traditional reinforcement learning was adopted in selection perturbative hyper-heuristics. Kheiri & Keedwell (2017) proposed a sequence-based selection hyper-heuristic that maintains scores representing the probability of choosing a low-level heuristic. The scores are updated by employing a reinforcement learning strategy during the search. The selection perturbative hyper-heuristics can perform well for problems with perfect and complete information, but could potentially suffer from performance degradation when dealing with uncertain factors or dynamic events which impact on the performance of decisions obtained offline.

As depicted in Figure 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.

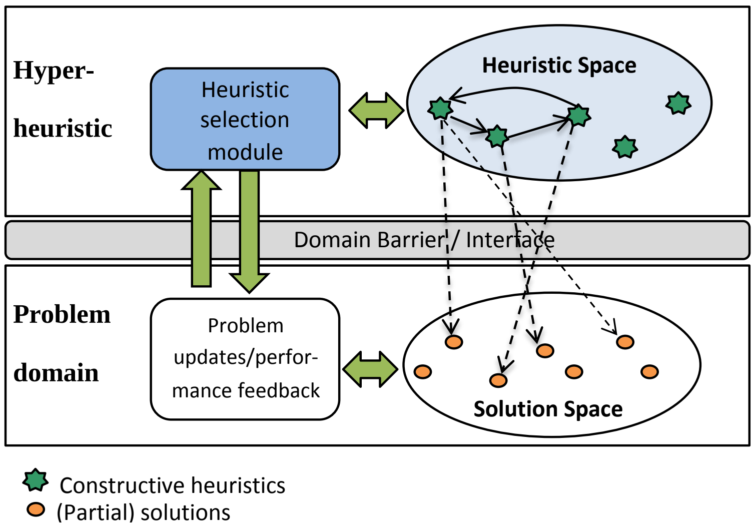

Figure 1: A selection constructive hyper-heuristic framework

Evolutionary algorithms have been the most adopted methods to explore sequences of low-level heuristics for solution construction. With low-level graph colouring heuristics, Burke et al. (2007) proposed a constructive selection hyper-heuristic to solve educational timetabling problems. Soghier & Qu (2013) presented a hybrid approach for exam timetabling utilising classic graph colouring heuristics to choose and assign an exam before using bin packing heuristics to allocate a time slot and room. To generate probability distributions of sequences of low-level heuristics at different stages of a search, Qu et al. (2015) applied a Univariate Marginal Distribution Algorithm (UMDA). Gomez & Terashima-Marín (2018) used an evolutionary algorithm to evolve rules to select low-level heuristics for solving multi-objective two-dimensional bin packing problems.

In these selection constructive hyper-heuristics, a decision is made step by step at each decision point, offering the potential to solve online combinatorial optimisation problems. However, the aforementioned methods can only perform effectively when certain vital information of problem characteristics is known beforehand. For example, in 2-D bin packing problems, the items (rectangles) need to be known in advance. After being sorted, the rectangles are fed to the algorithm. In timetabling problems, all the events that are not yet scheduled similarly 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 timeslots 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.

To handle uncertainties in real-world problems in a very flexible way, Chen et al. (2016) established a manually crafted dynamic heuristic based on the human experience. The algorithm obtained solutions that are superior to those used in practice. This type of manual heuristics can be used as baselines for performance evaluations but are often far from optimality. MacLachlan et al. (2019) proposed a Genetic Programming Hyper-Heuristic (GPHH) to develop routing policy for the Uncertain Capacitated Arc Routing Problem, while Chen et al. (2020) proposed a data-driven genetic programming heuristic that evolved different decision-making rules (heuristics) in solving real-world truck routing problems in a container port. Superior results were reported in comparison with those from Chen et al. (2016).

The GP based hyper-heuristics used by MacLachlan et al. (2019) and Chen et al. (2020) showed to improve the manual heuristic in Chen et al. (2016), as the resulting heuristic (in the form of a GP decision tree) is evolved with a large number of instances, thus of better average performance than human-designed ones. However, the evolution process of GP can be extremely time-consuming on hundreds of training instances, and the resulting tree can be too large to be used in practice with a large number of features and terminal operators.

In the present research, a Deep Reinforcement Learning (DRL) based selection constructive hyper-heuristic is proposed to solve difficult online combinatorial optimisation problems with uncertain variables revealed over time. Compared against the existing hyper-heuristics, the proposed hyper-heuristic method can take advantage of large scale historical data of the random variables to train the heuristic selection module so that robustness can be built into the constructed solution. **Furthermore, we note that the solutions obtained by the proposed method is of improved interpretability, compared with those obtained by traditional deep reinforcement learning. This is due to: Firstly, the underlying actions in the proposed framework are human-understandable heuristics (rather than direct variable fixing actions in conventional deep reinforcement learning); Secondly, through the spectrum analysis of the state-action pairs, we can identify decision patterns in which the agent prefers to choose certain low-level heuristics (actions) at specific decision points (states).**

3. The Proposed DRL Hyper-heuristic Framework

In the new hyper-heuristic framework for online combinatorial optimisation problems, a deep reinforcement learning is introduced into an existing selection constructive hyper-heuristic framework. Specifically, a double deep Q-network (DDQN) (Van Hasselt et al., 2016) was utilised to train the present heuristic selection module exhibited in Figure 1. Details are described in the following sub-sections.

DRL combines reinforcement learning (RL) and deep learning. Since it was first proposed by Mnih et al. (2013), DRL has attracted intensive attention, with the highlight of AlphaGo (Silver et al., 2016) which beat the world Go champion Lee Sedol in 2016. The deep neural networks in DRL are capable of perceiving and extracting advanced features from data automatically; while RL can iteratively improve the decision-making thereof by 'trial and error' interactions with the problem model. 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.

3.1. DRL based Hyper-heuristics

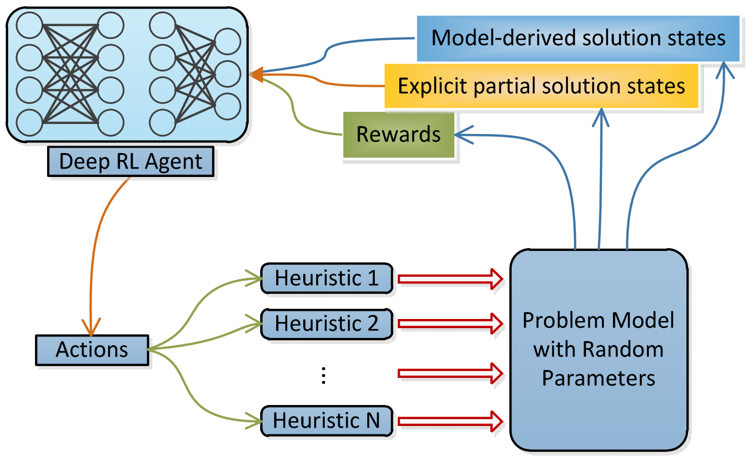
Model-derived solution states
Explicit partial solution states
Rewards
Deep RL Agent
Actions
Heuristic 1
Heuristic 2
⋮
Heuristic N
Problem Model with Random Parameters

Figure 2: The DRL hyper-heuristic framework

The proposed DRL based constructive hyper-heuristic framework is depicted in Figure 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 (see Section 3.3 for details) and 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 1, where γ is the discounted factor, t is the time step, R is the reward and π is the policy.

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

Notably, in the DRL framework, the training data of the states, actions and rewards is generated during the interactions between the agent and the problem model with random parameters. 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 constructive hyper-heuristic method capitalises on both the problem mathematical model (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.

3.2. Offline Training of Double Deep Q Network (DDQN)

Deep Q-network (DQN) is a widely used robust DRL method, which combines a deep neural network function approximator with the classical Q-learning algorithm to learn a state-action value function. By acting greedily, a policy can be iteratively acquired (Mnih et al., 2013). Numerous methods to enhance the performance of the original DQN have been proposed in prior literature. In the present framework, the double deep Q network method (DDQN) (Van Hasselt et al., 2016) with the experience replay strategy (Lin, 1993) is adopted because of their performance consistency and fast convergence. 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 hyper-heuristics, rather than exploring the best DRL method in the context of hyper-heuristics. Particular focus is centred on the hybridisation of data-driven and model-driven schemes as well as the interpretability of the proposed DRL hyper-heuristic framework.

DQN is a deep neural network that outputs a vector of actions' (i.e. low-level heuristics) preference values $Q(s, \bullet; \theta)$ given state s, where θ is the set of parameters of the network that can be trained to help select the most appropriate heuristics. Mnih et al. (2015) utilised a target network and experience replay strategy that substantially improved the performance of the basic DQN. The target network (with parameters θ_t^-) and the online network (with parameters θ_t) share the same structure, but the parameters of the target network are only updated every τ steps from the online network. Here, τ is a parameter to define how frequently the target network parameters (θ_t^-) are updated. The target used by DQN can be described by Equation 2.

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-) \tag{2}$$

Owing to the max operator in the standard DQN in Equation 2, the DQN agent is more likely to select overestimated values, leading to over-optimistic value estimates. Double-DQN (DDQN) is hence introduced to reduce over-estimations by decomposing the max operation in the target into action selection and action evaluation. Then, the target used by DDQN is changed to Equation 3:

$$Y_t^{DDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-) \tag{3}$$

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. 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 1 in Appendix A.

3.3. Solution state

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). 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.

3.4. Actions

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 two sections, we demonstrate how the proposed hyper-heuristic method can be used to solve two different combinatorial optimisation problems with challenging uncertainties and evaluate its performance in comparison with existing methods.

4. Application to Online Container Terminal Truck Routing Problem

4.1. Online Container Truck Routing Problem Description

The real-world problem considered in this paper is a container truck routing 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 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 a typical layout in Figure 3). 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 (ship cranes) and the yard area (yard cranes). The yard area consists of a number of yard blocks, each of which with a unique yard block ID (e.g. A1-A6 in Figure 3), and is equipped with a single yard crane. A fleet of homogeneous trucks, based at the depot initially, transport containers between ship cranes and yard blocks. In this problem, the depot, each ship crane and each yard block is represented as a node.

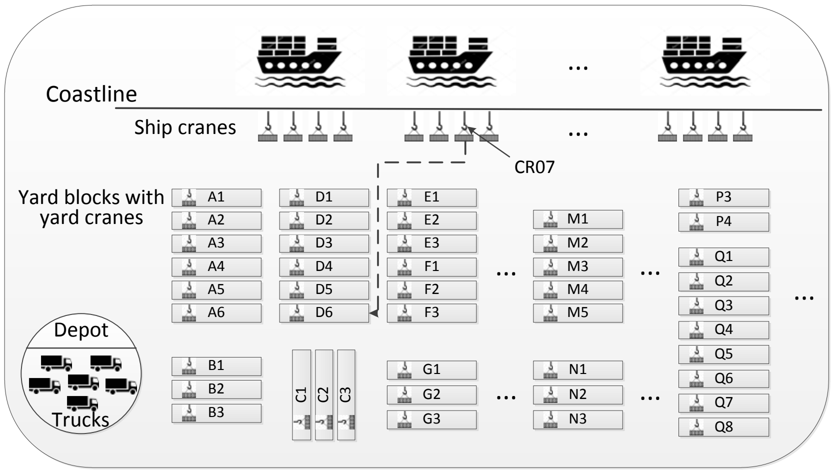

Figure 3: Typical layout of a container terminal

As the most valuable resources in a container terminal, ship cranes 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 ship crane waiting time between two consecutive operations, which is mostly caused by late truck arrivals.

Different from other container truck routing problem studies, the problem considered in this study is modelled 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 ship and yard cranes with deterministic problem formulation.

Each time when a vessel arrives, high-level decisions are made in terms of the assignments of the berth, the ship cranes and the yard blocks for this vessel. For practicality, these decisions are made separately from the truck routing problem concerned in this paper. Additionally, for each assigned ship crane, 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. Again, practical rules require that each ship crane is responsible for one type of operations only (i.e. either loading or unloading but not both). 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). A task is serviced by one truck exactly. The dashed line in Figure 3 represents a transportation task from a ship crane CR07 to yard block D6.

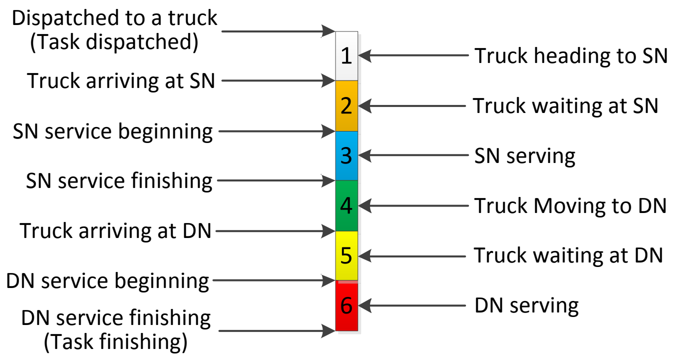

Figure 4: The flow of handling a task.

The detailed events of handling a task are shown in Figure 4. The timings of these events for all tasks define a full truck dispatching solution. The ship cranes strictly follow the sequences defined by the work queues while operations at each yard crane adopt the First-Come-First-Serve (FCFS) policy.

4.2. 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), ship cranes N^{ship}, and yard cranes N^{yard}. Set A denotes the arcs between the nodes. Let WQ_l denote the work queue list associated with ship crane $l \in N^{ship}$ 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 ship crane (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 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.

4.2.1. 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 ship and yard cranes (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.

4.2.2. Objective function

The objective of the problem is to minimise the aggregated ship crane 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 (4).

$$\min \sum_{l \in N^{ship}} \sum_{h=1}^{n_l - 1} [T(q_l^{h+1})^{ship} - T(q_l^h)^{ship} - t(q_l^h)^{ship}] \tag{4}$$

4.2.3. Constraints

The following constraints need to be satisfied to ensure the feasibility of the solution. Constraint (5) ensure that all tasks are serviced exactly in the order specified by the work queues. Constraint (6) ensures the feasible operation start times for any two consecutive tasks assigned to the same truck. Constraints (7)-(8) 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. Its mathematical representation is not included due to its expression complexities and page limitation.

$$T(q_l^{h+1})^{ship} \geq T(q_l^h)^{ship} + t(q_l^h)^{ship}, \quad \forall q_l^h \in WQ_l, l \in N^{ship} \tag{5}$$

$$T(j)^{SN} \geq (T(i)^{DN} + t(i)^{DN} + td(i,j))x_(i,j)^k \quad \forall i,j \in Q, k \in K \tag{6}$$

$$\sum_{j \in Q} \sum_{k \in K} x(i,j)^k = 1 \quad \forall i \in Q \tag{7}$$

$$\sum_{i \in Q} \sum_{k \in K} x(i,j)^k = 1 \quad \forall j \in Q \tag{8}$$

Note that in this study, 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 combinatorial problem 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 way, the problem can then be solved within a reinforcement learning framework. Nevertheless, the DRL agent is exposed to the nature of the uncertainties through training instances; while its performance is evaluated on a set of independently generated instances.

4.3. Implementation details

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.

4.3.1. State design

Unlike in most reinforcement learning problems, such as playing Atari games, where screen images can be directly fed as a state to the neural network, a real-world problem like the container truck routing in a port terminal requires professional advice on selecting features to encode a state. The following features have been seen as vital by our collaborators in the port when dispatching a task, being subsequently selected to be part of the state.

- The remaining number of tasks each ship crane needs to finish (i.e. the length of the work queue),
- The distance between the current position of the truck to be dispatched and the source nodes of the first tasks of every work queue,
- The predicted number of trucks to serve every ship crane, including trucks already dispatched and to be dispatched in the near future (i.e. the supply),
- The predicted number of tasks to be finished at every ship crane 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. For example, a moving-average method is applied to predict the service time of a task in a ship crane. Algorithm 2 in Appendix A denotes how the problem mathematical model calculates the predicted number of tasks to be finished at every ship crane 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 the four items, represented as:
$[[rn_1, rn_2, \cdots, rn_m], [[d_1, d_2, \cdots, d_m], [wn_1, wn_2, \cdots, wn_m], [pn_1, pn_2, \cdots, pn_m]]$.

4.3.2. 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. $A_{agent} = \{agent\ action\ i \mid i \in (0, 1, 2, \cdots, 9)\}$. 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 ship cranes. 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. See Appendix B for more details of the low-level heuristics. This leads to a total of 10 low-level heuristics.

4.3.3. Rewards

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 ship crane 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. $T(q_l^{h+1})^{ship} - T(q_l^h)^{ship} - t(q_l^h)^{ship}$. 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 ship crane idle time between tasks.

Note that, when computing the reward of a specific task assignment, the corresponding ship crane 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 ship crane. 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.

4.4. 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 combinatorial optimisation problem are scarce due to its huge solution space and a relatively low response time required. The manually crafted heuristic (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).

4.4.1. Datasets

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 contain 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 to sufficiently simulate the complexity of the real-life data (see Appendix C.1 for more details). Their real values are revealed dynamically over time.

4.4.2. Experiment settings

We adopted a four-layer DRL similar to the one used in Chen & Tian (2019), where a DRL was used to solve some deterministic combinatorial optimisation problems. Due to space limitations, the details of our DRL settings are given in Appendix C.2.

4.4.3. 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. The average total crane waiting time of the present DRL-HH for both datasets during the training process is shown in two figures in Appendix D.1.

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. Meanwhile, rank tests were conducted to fully compare the performance of the proposed method and the other two benchmark methods, results shown in Table 1. It was shown that combining multiple heuristics is beneficial than applying any of them alone (Burke et al., 2007; Pillay & Qu, 2021). The comparison of our proposed DRL-HH against each individual heuristic is presented in Appendix D.2, and results confirm this finding in the literature.

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

Table 1: 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 average rank from the rank test.

Datasets	Average total crane waiting time (s) / Average rank			Imp% of DRL-HH over manual heuristic
	Manual heuristic (baseline)	Data-driven GP	DRL-HH	
Small_Basic	1808.02/2.85	1786.32/2.04	1674.40/1.11	7.39%
Big_Basic	6392.29/2.87	6333.48/2.03	5868.30/1.10	8.20%

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).

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 ship cranes) 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 Appendix C.1 for details). The experimental results are presented in Table 2.

Table 2: Results of Scalability Experiment 1. In x/y, x is the average waiting time, and y is the average rank from the rank test.

Datasets	Average total crane waiting time (s) / Average rank			Imp% of DRL-HH over manual heuristic
	Manual heuristic (baseline)	Data-driven GP	DRL-HH	
Small48T	1017.06/2.85	1037.40/2.06	971.44/1.09	4.52%
Small96T	2207.33/2.91	2192.76/1.95	2130.67/1.14	3.51%
Big144T	3601.26/2.83	3580.37/2.12	3450.30/1.05	4.20%
Big288T	8489.78/2.90	8423.56/2.02	8175.28/1.08	3.71%

In Scalability Experiment 2, DRL-HH was tested on 'small3WQ', 'small2 WQ', 'big5WQ', where the number of work queues involved is decreased (See Appendix C.1 for details), results shown in Table 3. Note that here we did not test our trained algorithm on instances with increased ship cranes due to two reasons: First, we foresee poor results because the DRL-HH agent knows nothing about these newly added ship cranes during the training. Second, for a real-world port, the maximum number of ship cranes stays unchanged. If we train our DRL-HH with the maximum ship cranes, the model would still perform well for instances with fewer ship cranes because it has seen all the information.

Table 3: Results of Scalability Experiment 2. In x/y, x is the average waiting time, and y is the average rank from the rank test.

Datasets	Average total crane waiting time (s) / Average rank			Imp% of DRL-HH over manual heuristic
	Manual heuristic (baseline)	Data-driven GP	DRL-HH	
Small3WQ	1360.45/2.93	1354.74/1.99	1313.10/1.08	3.42%
Small2WQ	895.43/2.87	889.79/1.99	869.92/1.14	2.93%
Big5WQ	4850.87/2.89	4805.76/2.04	4646.37/1.07	4.22%
Big4WQ	3920.40/2.79	3898.45/2.05	3774.21/1.16	3.76%
Big3WQ	2620.34/2.81	2612.49/2.04	2533.54/1.15	3.32%
Big2WQ	1745.55/2.89	1735.95/1.97	1692.46/1.14	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 significantly in the problem structure. Hence, obtaining a 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 manual heuristic (except for the task set with 48 tasks in Table 2).

Notably, in Table 2 and Table 3, 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.

DRL VS. DRL-HH. Finally, the direct use of DRL to choose work queues 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 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: Average training time and number of training episodes for two methods to achieve the same results

Task sets	Methods	Average Training Time (minute) & StdDev	Average num of episodes & StdDev
Small_basic	DRL-HH	87(6)	1933(61)
	Simple DRL	324(11)	7740(209)
Big_Basic	DRL-HH	397(17)	1992(75)
	Simple DRL	1640(45)	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).

4.5. 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.

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 5. It can be observed that out of the total 11035 states, actions 0, 1, 3 and 6 are among the most frequently used heuristics.

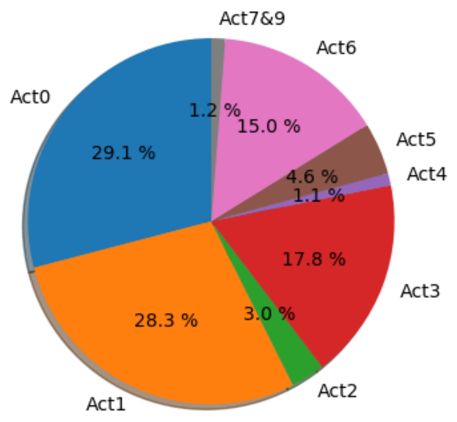

Figure 5: The distribution of the actions corresponding to the 11035 states.

Spectrum analysis was conducted on each group of these classified states. Taking action 0 in Figure 6 as an example, where the x-axis represents the state elements discussed in Section 4.3.1. The urgent degree (see Appendix B for details) 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 values 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. In Figure 6 and Figure D.3 in Appendix D.3, 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 D.3 and Figure D.4 in Appendix D.3 indicate that compared with action 4, action 3 was more likely to be chosen when the urgent degree was high.

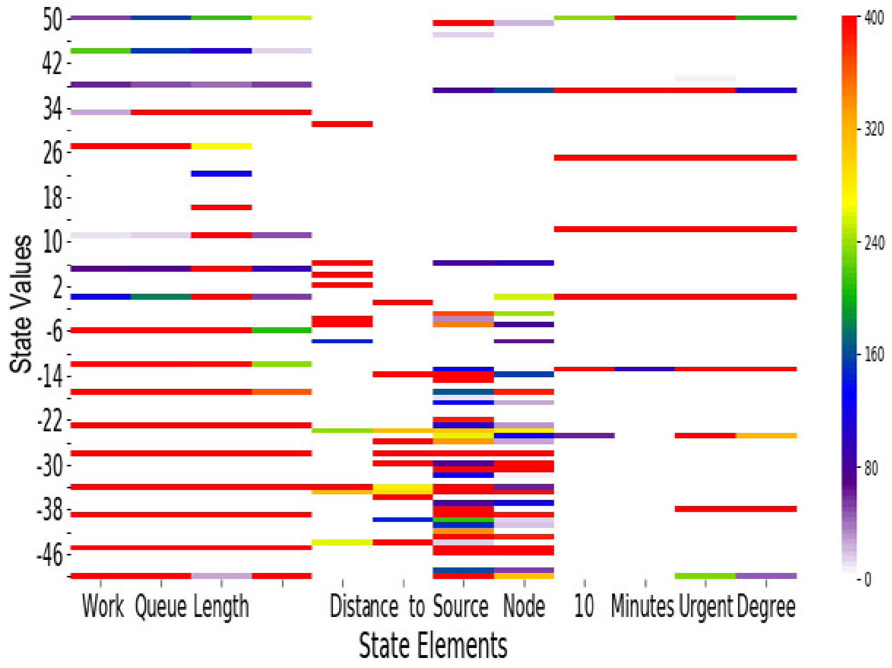

Figure 6: The spectrogram of states for action 0.

The results reveal that certain patterns exist between the states and the corresponding actions. 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.

5. Application of DRL-HH to online 2-D strip packing problem

The proposed DRL-HH was further evaluated on a classic online 2-D strip packing problem. Two online variants of the `best-fit` algorithm (Burke et al., 2004) were used as baselines. The first is `best-fit` with a fixed strategy, which only selects the "leftmost" placement throughout the whole packing process. The second is `best-fit` with a stochastic strategy, which selects different placements at different decision points. At the same time, we also included in the comparison the grouping super harmonic algorithm (GSHA) proposed in Han et al. (2016), which is commonly considered as a state of the art online method. More details of this problem and the experiments are reported in Appendix E.

Table 5 displays the average results (i.e. heights) achieved by the proposed DRL-HH in comparison with the other three methods. An observation can be made that the present DRL-HH method outperformed the best-fit algorithm on average, and ranked the highest (1.22). The recently proposed GSHA method, to our surprise, did not perform well. This is probably due to its lack of internal learning mechanism to the given problem data set. The results demonstrate the generality of the proposed method across different types of online optimisation problems.

Table 5: The average heights achieved by four approaches and their scores in the rank test for the online 2-D strip packing problem.

Datasets	Average heights of packings / Rank score			
	Best-fit with fixed strategy Burke et al. (2004)	Best-fit with Stochastic strategy Burke et al. (2004)	DRL-HH	GSHA Han et al. (2016)
Training	271.8	272.2	269.3	374.3
Test	271.7/2.07	271.9/2.29	269.1/1.22	372.8/3.78

6. Conclusions and Future Work

Real-world combinatorial optimisation problems are frequently featured with uncertainties. This poses a major challenge to traditional optimisation algorithms. In this paper, we propose a deep reinforcement learning (DRL) based hyper-heuristic framework. For the first time, DRL was introduced into a constructive hyper-heuristics framework to address the challenging online combinatorial optimisation problems. Experimental results highlight several advantages with this new framework. Firstly, it shows better performance compared with the existing state of the art methods on both a real-world truck routing problem and a 2D strip packing problem with uncertainties. Secondly, it shows a good scalability when the problem sizes change. Thirdly, compared with traditional DRL methods, it holds better convergence. Finally, the proposed approach showed to improve the interpretability of the solutions, thus is more acceptable in real-life applications.

In our future work, first, we can explore a better neural network structure with variable input dimensions to adapt to the changing number of ship cranes or other relevant elements. Second, it would be interesting to investigate whether the ‘patterns’ or ‘knowledge’ obtained from the state spectral analysis can be fed to the learning agent at the early training process to improve the training efficiency.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (grant number 72071116) and the Ningbo Science and Technology Bureau (grant numbers 2019B10026, 2017D10034).

References

Ahmed, L., Mumford, C., & Kheiri, A. (2019). Solving urban transit route design problem using selection hyper-heuristics. *European Journal of Operational Research*, *274*, 545–559.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH*, *10*, 43–66. doi:{10.1007/s10288-011-0182-8}.

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. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics* (pp. 449–468). Springer.

Burke, E. K., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, *52*, 655–671.

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

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, 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, X., Zhang, H., Wu, C., Mao, S., Ji, Y., & Bennis, M. (2018). Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, *6*, 4005–4018.

Choong, S. S., Wong, L.-P., & Lim, C. P. (2019). An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. *Swarm and evolutionary computation*, *44*, 622–635.

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.

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2019). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, .

Fisher, R. D., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial scheduling* (pp. 225–251).

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.

Han, X., Iwama, K., Ye, D., & Zhang, G. (2016). Approximate strip packing: Revisited. *Information and Computation*, *249*, 110–120.

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.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks*. Technical Report Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

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).

Mei, Y., Tang, K., & Yao, X. (2010). Capacitated arc routing problem in uncertain environments. In *IEEE Congress on Evolutionary Computation* (pp. 1–8). IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, .

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.

Pillay, N., & Qu, R. (2018a). *Hyper-Heuristics: Theory and Applications*. Springer.

Pillay, N., & Qu, R. (2018b). Selection constructive hyper-heuristics. (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.

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.

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.

Soghier, A., & Qu, R. (2013). Adaptive selection of heuristics for assigning time slots and rooms in exam timetables. *Applied Intelligence*, *39*, 438–450.

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.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Zamli, K. Z., Alkazemi, B. Y., & Kendall, G. (2016). A tabu search hyper-heuristic strategy for t-way test suite generation. *Applied Soft Computing*, *44*, 57–74.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., & Li, Z. (2018). Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference* (pp. 167–176).