



Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:

Tu, C; Bai, R; Aickelin, U; Zhang, Y; Du, H

Title:

A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems

Date:

2023-11-15

Citation:

Tu, C., Bai, R., Aickelin, U., Zhang, Y. & Du, H. (2023). A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems. Expert Systems with Applications, 230, <https://doi.org/10.1016/j.eswa.2023.120568>.

Persistent Link:

<https://hdl.handle.net/11343/336007>

A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems

Chaofan Tu^{a,1}, Ruibin Bai^{a,*}, Uwe Aickelin^{b,3}, Yuchang Zhang^{a,4}, Heshan Du^{a,5}

^a School of Computer Science, University of Nottingham Ningbo China, Ningbo, China & Nottingham Ningbo China Beacons of Excellence Research and Innovation Institute, China

^b School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

ARTICLE INFO

Keywords:

Hyper-heuristic
Deep reinforcement learning
Feature fusion
Knapsack problem
Strip packing problem

ABSTRACT

In recent years, deep reinforcement learning has shown great potential in solving computer games with sequential decision-making scenarios. Hyper-heuristic is a generic search framework, capable of intelligently selecting or generating algorithms to solve a class of optimisation problems with stochastic or dynamic settings. This paper proposes a new general framework for solving online packing problems using deep reinforcement learning hyper-heuristics. Although analytical approaches can address most offline packing problems successfully, their online versions have proved much more challenging and the performance of the existing methods is often not satisfactory. In this paper, we extend a recent deep reinforcement learning hyper-heuristic framework by fusing the visual information of real-time packing with distributional information of random parameters of the problem. Computational experiments show that our method outperforms the state of the art online methods with reductions in optimality gap between 2%–19% for knapsack problem and 0.7% for the online strip packing problem. In addition, a new visual analysis presentation is also devised to better interpret the learned packing strategies, which can reveal more information than the widely used landscape analysis. As online packing problems are widely available in production environments, the proposed approach can serve as an important reference to solve other similar combinatorial optimisation problems for which visual layout inputs would aid learning.

1. Introduction

Combinatorial optimisation has a wide range of real-life applications such as planning, scheduling, resource allocation, and decision making (Blum, Pinacho, López-Ibáñez, & Lozano, 2016). Many of these problems belong to NP-hard problems, among which packing problems are some of the first and most widely studied problems. They serve as good references for studying other NP-hard problems, in addition to their apparent real-life applications in logistics, transportation, space planning, and material cutting. For example, finding the least-waste material cutting, the very large-scale integration (VLSI) layout design, newspaper

layout planning can be formulated as packing problems. When the complete information of the problem is available prior to problem solving, packing problems are considered offline and can be solved using exact or approximate methods such as those in Burke, Hyde, Kendall, and Woodward (2010) and Domović, Rolich, and Golub (2019).

However, in many real-world packing problems, the problem inputs (parameters) are sometimes revealed one after another over time. For example, the berth allocation problem (Lim, 1998) in a container terminal can be abstracted as an online strip packing problem, where the arrival and handling times (dwell times) of each vessel defines the rectangle to pack next but they are not fully determined prior to problem

* Corresponding author.

E-mail addresses: chaofan.tu@nottingham.edu.cn (C. Tu), ruibin.bai@nottingham.edu.cn (R. Bai), uwe.aickelin@unimelb.edu.au (U. Aickelin), yuchang.zhang@nottingham.edu.cn (Y. Zhang), heshan.du@nottingham.edu.cn (H. Du).

¹ ORCID: 0000-0002-9730-3563.

² ORCID: 0000-0003-1722-568X.

³ ORCID: 0000-0002-2679-2275.

⁴ ORCID: 0000-0002-4152-6784.

⁵ ORCID: 0000-0002-6300-3503.

solving. Also, personalised recommendation systems (Zhao, Zhang, Friedman, & Tan, 2015) can be formulated as the online knapsack problem due to uncertainties in customer's willingness-to-pay and dynamic prices (discounts). Other examples of online knapsack problems include budget constrained bidding in auctions (Zaidi, Bhatti, & Ullah, 2018; Zhou, Chakrabarty, & Lukose, 2008), dynamic pricing in vehicle ferries (Bayliss, Currie & Bennell et al. 2019), portfolio investment (Vaezi, Sadjadi, & Makui, 2019), and electric vehicle charging (Sun et al., 2020). A number of related efforts have been made to tackle the online packing problems. A recent survey of packing problems introduced both offline and online algorithms (Oliveira, Neuenfeldt Júnior, Silva, & Carravilla, 2016). These existing approaches, however, are usually bespoke for specific problem domains thus lack the generality across multiple problems. More importantly, the current methods for online packing problems still lacks the required performance in extremely competitive businesses and hence better algorithms are needed.

With the significant advancement in deep learning in recent years, it is now a hot topic to use deep reinforcement learning to solve combinatorial optimisation problems, especially the online optimisation problems which are proved still challenging for classic optimisation methods. Most online combinatorial optimisation problems can be reviewed as a sequential decision-making problem. Mnih et al. (2015) showed that by combining deep learning and reinforcement learning, the resulting method has great potential to solve computer games. Another eye-catching success of Deep Reinforcement Learning (DRL) is the AlphaZero (Silver et al., 2017), an artificial intelligence application that was able to defeat the human world champions of Go. Inspired by those achievements, there have been more and more attempts to solve combinatorial optimisation problems using DRL (Chen & Tian, 2019; Yi, Qu, & Jiao, 2023; Kool et al., 2018). However, most of studies focus on offline optimisation problems and there are relatively fewer studies to tackle the combinatorial optimisation problems in online settings, which are more challenging due to uncertainties. It is often of great challenge to determine state features and reward schemes for different problems. Most of the existing methods favour end-to-end type of solution style and tend to encode problem raw data as inputs. However, for packing problems, manual heuristics would not just focus on the objective(s), but also use visual packing layout as instinctive features which are missing from the existing studies.

For online packing problems, considering the fact that visual features are widely used in real-life problem solving but are missing in some existing methods using deep reinforcement learning (Zhang, Bai, Qu, Tu, & Jin, 2021; Hubbs et al., 2020; Kong et al., 2019), we propose a novel DRL hyper-heuristic with feature fusion, which is a mechanism that involves visual information to better characterise uncertain environments. Due to the generalisation abilities of the chosen hyper-heuristic framework, the proposed approach are expected to perform well for online combinatorial problems that benefit from visual information during problem solving. We applied the proposed approach to two classic packing problems and the benefits of feature fusion with visual information are clearly established through empirical experiments. Additionally, we further analysed the behaviour patterns of learned strategies, which can help interpret (at least partially) the decision-making process of our proposed method.

The remainder of this paper is organised as follows. Section 2 reviews related work and Section 3 presents some relevant foundations. It is followed by the proposed methodology in Section 4 which mainly presents the high-level design principles concerning the deep reinforcement learning hyper-heuristics and the fusion of two different classes of information characterising the problem states to be addressed. Section 5 provides the implementation details for two specific problems, including problem-specific heuristics and the process of feature extraction. Section 6 is the related numerical experiments and analysis. Following the discussion in Section 7, Section 8 concludes with possible further extensions.

2. Literature review

It is well known that many combinatorial optimisation problems are NP-hard, and their optimal solutions cannot be found in polynomial time. Exact algorithms often take prohibitive amount of time to find optimal solutions (Cook, 2006). Cleverly searching the solution space through branch-and-bound methods can prune the search tree and reduce time complexity to certain extent, while polynomial-time approximation schemes are the other helpful methods to solve the problem to near optimality (Vazirani, 2013). Nevertheless, many of these methods are still not practical for real-life applications where decisions are often required to be made in seconds if not in real-time (Bai et al., 2023).

Packing problem is one of such combinatorial problems with rich literature. More and more sophisticated approaches are still being proposed over time. One can refer to two surveys (Christensen, Khan, Pokutta, & Tetali, 2017; Oliveira et al., 2016) for recent advancements. Among the packing problems, perhaps the simplest one is the 0-1 knapsack problem, which is NP-complete (Cormen, Leiserson, Rivest, & Stein, 2009). Although its offline version is generally not considered difficult to solve, the online knapsack problem is proved much more challenging. The well-known greedy heuristic "Bang-per-buck" was first introduced in Dantzig (1957), followed with a number of other algorithms, ranging from dynamic programming to metaheuristic algorithms (Abdel-Basset, El-Shahat, & Sangaiah, 2019; Feng, Yang, Wu, Lu, & Zhao, 2018). Various resource allocation paradigms can be modeled sharing characteristics of the knapsack problems, including combinatorial auctions (Zaidi et al., 2018), virtual machine placement (Wang et al., 2018), portfolio investment (Vaezi et al., 2019), dynamic pricing (Bayliss, Currie, Bennell, & Martinez-Sykora, 2019) and electric vehicle charging (Sun et al., 2020). A more difficult version of packing problem is the 2-D strip packing problem which was first studied by Baker, Coffman, and Rivest (1980). Hopper & Turton compared solution quality and solution time of different meta-heuristics for 2D strip packing (Hopper & Turton, 2001). A tailor-designed algorithm, called best-fit, was proposed in 2004 which has widely been considered as the most effective heuristic to solve the strip packing problem (Burke, Kendall, & Whitwell, 2004). The best-fit heuristic has good performance on a wide range of benchmarks. Some follow-up works are conducted, including improvements to reduce the time complexity as well as some theoretical proofs (Imahori & Yagiura, 2010; Verstichel, De Causmaecker, & Berghe, 2013; Wei, Hu, Leung, & Zhang, 2017).

The above hand-crafted algorithms to solve combinatorial optimisation problems involve expensive efforts by different domain experts in specific domains, and they need to be revised when problem settings change. In spite of metaheuristics' capability to solve cross-domain problems, they still search on problem spaces and need expertise in both algorithm design and problem domain-specific knowledge, if pursuing high performance (Gogna & Tayal, 2013). However, hyper-heuristics can provide a more general search paradigm by automatically selecting or generating algorithms for different problems, and have drawn much attention (Burke et al., 2013). Back to the late 1990s, the term hyper-heuristic was coined to describe a combination of artificial intelligence methods for automatic theorem proving (Denzinger, Fuchs, & Fuchs, 1997). In combinatorial optimisation community, now it is presented as an automated methodology using a high-level heuristic to generate and/or select low-level heuristics (Chen, Bai, Qu, & Dong, 2022; Pillay & Qu, 2018).

A genetic programming hyper-heuristic was designed to solve the strip packing problem (Burke, Hyde, Kendall, & Woodward, 2010). These automatically generated heuristics showed good performance compared with human created algorithms. A simulated annealing hyper-heuristic was proposed to solve bin packing problem (Bai, Blazewicz, Burke, Kendall, & McCollum, 2012). It employed stochastic strategy along with a short-term memory, and achieved competitive results when compared with tailor-made algorithms. On the multidimensional

knapsack problem, Drake, Özcan, and Burke (2016) adopted a hyper-heuristic that is integrated with a mechanism of maintaining potential solutions at different conceptual levels to help generate higher quality solutions. In Domović et al. (2019), the authors used the genetic programming hyper-heuristic with a designed placement order to solve a real-life application in the textile industry, which can be regarded as a kind of strip packing problem. Related tests showed that it outperformed all other methods. For the 0-1 knapsack problem, Burke, Hyde, Kendall, and Woodward (2012) implemented a genetic programming hyper-heuristic approach, with results indicating that it can outperform its human-designed counterparts, especially for unseen instances. The work by Gómez-Herrera, Ramirez-Valenzuela, Ortiz-Bayliss, Amaya, and Terashima-Marín (2017) presented a hyper-heuristic method based on quartile-based information. On a heterogeneous set of 0/1 knapsack problem instances, it is more efficient than other four packing heuristics. By using the ant colony optimization, Duhart, Camarena, Ortiz-Bayliss, Amaya, and Terashima-Marín (2018) proposed a novel hyper-heuristic approach that performed equally, or even better, than spoke-designed packing. Two recent studies of hyper-heuristics in the knapsack domain have been conducted separately using a fuzzy system and an evolutionary-powered framework. In the former study, experiments demonstrated that they produced better average results than other hyper-heuristics, however there is still a gap to optimality (Olivas, Amaya, Ortiz-Bayliss, Conant-Pablos, & Terashima-Marín, 2020). The latter method, which does not rely on any problem characterization, performed best in most situations, but inferior to a simple packing heuristic in small cases (Sánchez-Díaz et al., 2021).

In addition to supervised and unsupervised learning, reinforcement learning is the third fundamental machine learning method and is attracting increasing attention. Williams (1992) is the first research to combine neural network and reinforcement learning to find optimal policies. However, it is only in the last decade that deep learning aided reinforcement learning (DRL) has started to offer promising solutions for some complex dynamic processes, thanks to the increased power of computing hardware. Volodymyr, Kavukcuoglu, Silver, Graves, and Antonoglou (2013) proposed the Deep Q-learning Network (DQN) which showed human-level performance in playing the Atari 2600 games. In Mnih et al. (2015) the DQN algorithm was improved by a more stable network update strategy for learning control policies more efficiently, which is the well-known nature DQN. Due to the impressive end-to-end decision-making ability of DRL, employing it in solving combinatorial optimisation problems is starting to become an important shift. A pointer network was firstly combined with reinforcement learning to solve the Traveling Salesman Problem (Bello, Pham, Le, Norouzi, & Bengio, 2016). They used the policy gradient algorithm (Mnih et al., 2016) to train model, which showed good performance for instances with up to 100 cities. Khalil, Dai, Zhang, Dilkina, and Song (2017) combined reinforcement learning and a deep graph embedding called structure2vec (Dai, Dai, & Song, 2016) for combinatorial optimisation problems with graph structures, including the Minimum Vertex Cover, Maximum Cut and Traveling Salesman problems. In Hu, Zhang, Yan, Wang, and Xu (2017), the authors proposed a new 3-D packing problem arising in delivery industry and solved it by using a DRL approach, experiments shown that it achieved about 5% improvement than traditional algorithms.

Whereas information of the problem is assumed to be known in advance in the offline settings, online algorithms can only access partial information at the start of the problem solving. Generally offline algorithms are not applicable to online situations, even though some of constructive heuristics can be adapted for online environments (Oliveira et al., 2016). In contrast to offline packing problems, input items appear one by one in online packing problems, which leads to great challenges. One of the main interests of online algorithms is related to the competitive ratio, which is a performance guarantee of an online algorithm. It is also used to analyse and compare different online algorithms at the theoretic level. Competitive analysis of online algorithms is a

fundamental area. Iwama and Zhang (2010) proved that the competitive ratio for online knapsack with removal does not exist. A randomised 2-competitive algorithm for a kind of knapsack problem was presented in Han, Kawase, and Makino (2015). For online strip packing, according to a recent survey (Christensen et al., 2017), a class of super harmonic algorithms of bin packing problems can be applied to online strip packing problem, with maintaining the same asymptotic approximation ratio.

However, in real production environments, performances on pathological cases may be not representative for most real cases and competitive analysis can only provide partial insights into solution quality. This is especially true when considering the long-term efficiency and actual production benefits. When it comes to the average-case performance, a few studies used the DRL approaches recently. For the online knapsack problem, Kong, Liaw et al. demonstrated a deep reinforcement approach that could learn similar behavioural patterns as online primal-dual algorithms on special dataset (Kong et al., 2019), but left work for universal distributions. In Hubbs et al. (2020), a deep reinforcement learning algorithm of PPO was applied to the online knapsack problem too, acquiring similar performances compared with human-designed heuristics. However, there was still gap to the optimal solution from solvers.

Since the selection hyper-heuristic does not perform time-consuming searching and generate real-time solutions in problem solving process, it is considered as one of the most effective methods for solving real-time dynamic or online problems (Drake, Kheiri, Özcan, & Burke, 2020). A few studies have incorporated reinforcement learning to further enhance the selection hyper-heuristic. Chong, Wong et al. implemented a Q-learning based hyper-heuristic and this integration shown competitive performances than traditional hyper-heuristics (Choong, Wong, & Lim, 2018). In a recent study to address an online truck dispatching problem primarily, a DRL method, the Double-DQN (Van Hasselt, Guez, & Silver, 2016) was combined with hyper-heuristic for better selecting dispatching strategies (Zhang et al., 2021).

Overall, in contrast to offline algorithms, online algorithms on average-cases have been less studied, mainly due to the challenge of uncertainty. Most of these online algorithms suffer from weak optimality and lack the generality to solve similar problems. Unlike many other combinatorial optimisation problems like TSP and VRP, visual layouts play very important role in real-life packing by packing operators. The use of visual information can be very beneficial to boost the performance of the existing DRL methods for packing problems. Therefore, to copy with these limitations, in this paper, we consider a deep reinforcement learning hyper-heuristic with feature fusion for online packing problems.

3. Preliminary

In this section, we provide some relevant foundations in advance, including general modeling, hyper-heuristic, and reinforcement learning.

3.1. General modelling

The goal of combinatorial optimisation problems is to find the optimal solution from a space of discrete feasible solutions subject to different constraints (Pardalos, Du, & Graham, 2013). More formally, the problem can be described in the following way: let I be an input instance of a combinatorial optimisation problem and denote sol be the solution that satisfies the associated problem constraint C . Hence, sol is called a feasible solution and the fact that C holds for sol is denoted as $C(sol)$. $\mathcal{F}$ is the mapping from the instance I to a feasible solution sol of the problem; $\mathcal{F}$ is the objective function of mapping the solution sol to the real-valued objective function. Given a deterministic input instance I , the corresponding set of solutions $\{\mathcal{F}(I)\}$ is likely to be very large, but

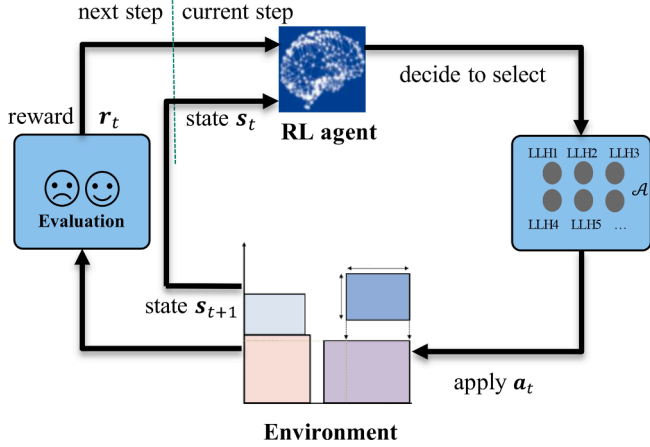


Fig. 1. RL agent to act as the high-level heuristic.

there must exist an optimal solution sol^* among them as Eq. (1).

$$sol^* = \operatorname{argmax}_{sol \in \{\mathcal{S}(I)\}} (\mathcal{F}(sol)) \quad (1)$$

According to whether the data in the input instance I is known in advance, the problems can be divided into offline or online. In offline problems, the entire information of the problem is known prior the problem solving and can be fully utilized. For example, for packing problems, a dispatcher has full knowledge of information of each item, and thus it can do some pre-processing operation like sorting all items in advance. However, many problems are needed to solve in an online manner, in which items in instances are only revealed one by one as time goes by. This online version of combinatorial optimization problems appears in many real-life pipeline tasks and real-time dispatching operations.

With the framework of Request-Answer Game introduced by Ben-David, Borodin, Karp, Tardos, and Wigderson (1994), we further illustrate the definition of online combinatorial optimisation problems. This definition can be generalised to cover many problems with sequence structure of input instances. In Request-Answer Game there are two adversarial players, one initiating a real-time request and the other responding in real-time. Specifically, consider a discrete time step horizon $H = [1, \dots, N]$, at each step $t \in H$, one player throws a random input request R_t (unassociated with the previous sequence), while the other player must output a response variable A_t , which is a feasible extension of the current solution. During the online process, the requests at the future time steps $\{R_i | i > t \text{ and } i \in H\}$ are unknown before time t . This restriction of immediate and irrevocable decision-making is the main characteristic of online problems. Finally, all the requests arranged in time sequence, $\langle R_1, R_2, \dots, R_t, \dots, R_N \rangle$, constitute one input instance of the problem; and the corresponding sequence of answer variables, $\langle A_1, A_2, \dots, A_t, \dots, A_N \rangle$, constitutes a feasible solution. Following above, the objective of an online maximisation problem can be formalised as Eq. (2).

$$\begin{aligned} & \operatorname{MAX}(\mathcal{F}(\langle A_1, A_2, \dots, A_t, \dots, A_N \rangle)) \\ & \text{such that} \\ & \langle A_1, A_2, \dots, A_t, \dots, A_N \rangle \in \{\mathcal{S}(\langle R_1, R_2, \dots, R_t, \dots, R_N \rangle)\} \text{ and} \\ & \mathcal{E}(\langle A_1, A_2, \dots, A_t, \dots, A_N \rangle) \text{ holds.} \end{aligned} \quad (2)$$

Because complete information is not available, solutions found by online algorithms cannot guarantee overall optimality. Online algorithms are traditionally measured by using the competitive ratio, which compares the objective function values of solutions by the online algorithm and the ideal offline solutions. For example, if an online algorithm is c -competitive, it means that for any instance I , the objective function value $\mathcal{F}(sol)$ is always c times bigger or smaller than $\mathcal{F}(OPT(I))$, where $OPT(I)$ denotes the ideal offline solution. Obviously, competitive

analysis implicitly assumes an adversary who provides the pathological instances which cannot stand for long-term efficiency and actual production benefits in practice. Therefore, it is noted again that the attention of this piece of work is related to the average performance over many online instances.

For solving general online combinatorial optimisation problems, this work investigates how to design algorithms that maximize the expectations of objective function values of obtained solutions. Specifically, denote the parameter Φ to represent the distribution of instances, and use R_t^Φ as each request variable, thus we can finally define the objective as Eq. (3).

$$\operatorname{MAX}(\mathbb{E}_{\Phi} [\mathcal{F}(\mathcal{S}(\langle R_1^\Phi, R_2^\Phi, \dots, R_t^\Phi, \dots, R_n^\Phi \rangle))]) \quad (3)$$

3.2. Hyper-heuristic

It is known that hyper-heuristics typically have a two-tier structure that comprises a control domain and a problem domain (Burke et al., 2013). The problem domain consists of different low-level heuristics (LLH) and evaluation function, whereas the control domain includes control strategy as well as high-level heuristic, specifically how to use those LLH to construct feasible solutions or improve the quality of solutions. By establishing domain shielding from problem environments, hyper-heuristics promote universal intelligent computing to find good solutions faster, while reducing the difficulty of algorithm design across different problems. Hyper-heuristics can be divided into two categories based on search paradigms: constructive and perturbative (Burke et al., 2013). The perturbative approach searches for better quality solutions from a complete solution, while the constructive approach starts with an empty solution and builds it until it is complete. Constructive hyper-heuristics are clearly better suited to solving online problems than perturbative ones. Based on the nature of the search space, hyper-heuristics can be classified into selection based and generation based (Burke, Hyde, Kendall, Ochoa, et al., 2010). In the selection based method, an existing heuristic is selected from a predefined LLH set. By contrast, the generation method selects algorithm components to generate new heuristics to solve problem instances. A hyper-heuristic can be considered as a learning algorithm when it can use feedback from problem domains to update the high-level control strategy in control domains. Online learning provides adaptive updating when solving a single instance, whereas offline learning gathers knowledge from a set of training instances, so as to provide better performance against unknown online instances. Hyper-heuristic is also closely related to algorithm-portfolio-based framework (Ryser-Welch & Miller, 2014) by interleaving different low-level heuristics in order for better performance than applying a single one alone (Oliveto, 2021).

3.3. Reinforcement learning

In the interdisciplinary field of machine learning and optimal control, reinforcement learning is also known as “approximate dynamic programming”. Most problems with stochastic and discrete-time sequential decision making can be modelled as Markov process, including some computer games, board games like Go and the Request-Answer Game framework mentioned above. It is said that a stochastic process is Markov if its conditional probability distribution of future states depends only on the present value; that is, if the present is given, the future is independent of the past. It is the memoryless property of stochastic processes that is called the Markov property. The most famous Markov process is the Markov chain. The Markov decision process (MDP) is an extension of Markov chains; the difference is the addition of actions (allowing for choice) and rewards (giving motivation) (Sutton & Barto, 2018). There are two ways to solve MDPs: model-based and model-free. The difference refers strictly to whether the predictions of environment response are used. The main upside to having or learning an environment model is that it allows to plan ahead. This is because it

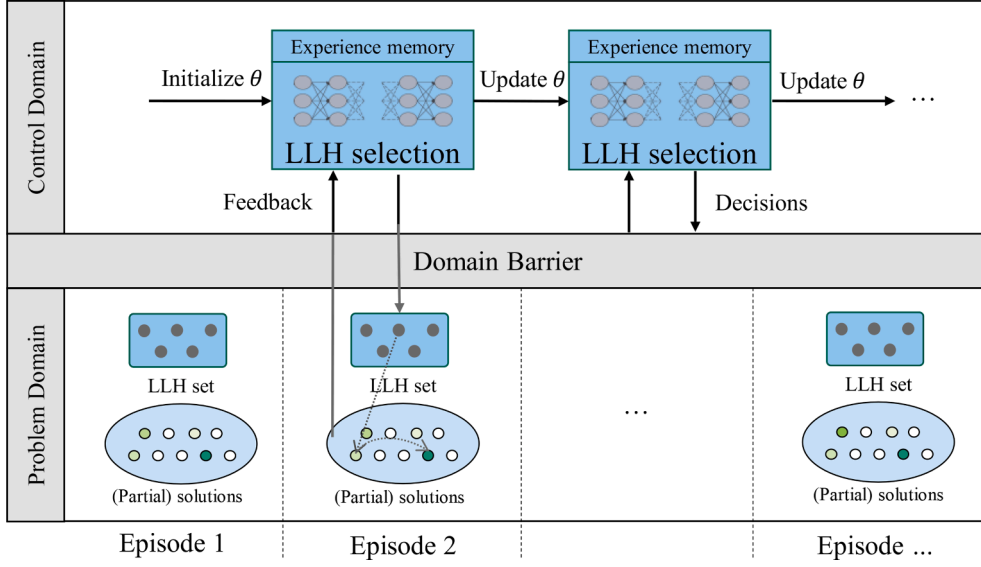


Fig. 2. Offline training of deep reinforcement learning hyper-heuristic.

can predict what would happen for a range of next steps, and explicitly decide between its options. The archetypical model-based approach is Dynamic Programming (DP), which uses predictions or distributions of next states and rewards to determine optimal actions. However, for online cases, a ground-truth model of the environment is usually not available. Consequently, we adopt a model-free reinforcement learning approach where environment models are not needed. The model-free method relies only on experiences from interactions with problem environments to learn the optimal target strategy.

Fig. 1 illustrates several important elements in the interaction where the reinforcement learning agent chooses optional actions. At time step t , the agent's current observation of the problem environment is the state s_t . Those optional actions belong to action space $\mathcal{A}$ on the right side are some specific LLHs for packing problem. For example, when a rectangle arrives in the pipeline, the agent's decision determines which LLH is selected to be executed in the problem environment, such as placing rectangle to the right or to the left of the container. After the execution of a_t , the environment state moves from s_t to the next time step's state s_{t+1} . Meanwhile, a reward r_t will be calculated based on the selected a_t , and the two states in this transition. It is a form of feedback used to measure how well the agent has made the decision. The design of the reward function is crucial and determines whether the agent's decisions can be effectively evaluated. Moreover, rewards may be delayed depending on previous decisions that were made. Reinforcement learning aims at increasing cumulative rewards as a long-term goal. An experience in such an interaction can be recorded as a trajectory $\langle s_t, a_t, s_{t+1}, r_t \rangle$.

4. Method

Any online combinatorial optimisation problem that fits the Request-Answer Game framework can be solved using a generic online algorithm to construct solutions, as shown in Algorithm 1. We can see that its core is a *dispatcher* sub-algorithm in response to generating an answer A_t irrevocably at each time step t . This decision-making is based on the latest request variable R_t along with previous requests and the effect of the current *sol* in problem environment. Clearly, it is shown that the performance of online algorithms only depends on the *dispatcher*, whose behaviour is dominated by dispatching rules it contains, whether these rules are manually designed or automatically generated.

Algorithm 1 General Algorithm for Request-Answer Game Problem

Input: an ordered sequence with n Requests $\langle R_1, R_2, \dots, R_N \rangle$.
sol $\leftarrow \langle \rangle$ (Initialise solution with an empty sequence).
 $t = 1$
while $t \leq N$:
 $A_t \leftarrow \text{dispatcher}(\langle R_1, R_2, \dots, R_t \rangle, \text{sol})$
 update *sol* by appending A_t , thus the current partial solution is $\langle A_1, A_2, \dots, A_{t-1}, A_t \rangle$
 $t = t + 1$
End
Output: *sol*.

There are many excellent manually designed algorithms for combinatorial optimisation problems (Colomi et al., 1996), however, those by domain experts tend to be expensive, time-consuming, and problem-dependent. In contrast, the hyper-heuristic approach provides solutions for NP-hard problems that are more general, with only limited problem-specific information needed. In the last two decades, it has been developed to solve different NP-hard problems, including scheduling, cutting and packaging, vehicle routing and other problems, which can be seen in two recent surveys (Burke et al., 2013; Drake et al., 2020). To address the challenge of uncertainty in online settings, our approach combines the end-to-end sequential decision-making capability of deep reinforcement learning with selection hyper-heuristic and employ a feature fusion method to extract key features of online environments. Details of our approach are presented in this section.

4.1. DRL Hyper-heuristic

In this piece of work, we intend to solve online classic packing problems, thus a selection constructive hyper-heuristic is adopted in conjunction with existing algorithms from the combinatorial optimisation community. It utilises a deep reinforcement learning based module as the high-level heuristic to characterize uncertain online environments. The training of the proposed hyper-heuristic along with its structure is illustrated in Fig. 2.

In the training process, tasks (i.e. items to be packed) in an instance are presented sequentially until they are all completed, and such a simulation is called an episode. During each time step in episodes, the LLH selection module must decide a LLH from the pre-defined LLH set to use for constructing solutions in real time. LLHs differ according to the problems they are designed to solve. However, high-level heuristics are more generally applicable to different problems due to domain shield-

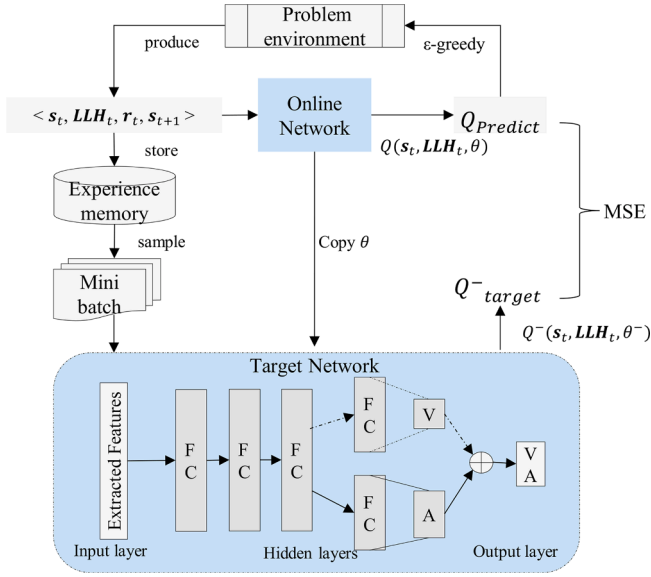


Fig. 3. Dueling Double Deep Q-learning Networks.

ing. As shown in Fig. 2, the high-level heuristic adopted is a neural network initialised by weights θ . During the training, those decisions made by the neural network are stored in an experience memory for updating the weights θ . More specifically, our approach utilizes a deep reinforcement learning trained model as the high-level heuristic, which collects decisions, states and rewards from problem environments to improve itself performance. We next describe how to train the neural network to obtain good decision-making ability.

4.2. Dueling double deep Q-learning network

In this study, a powerful variant of DQN (Mnih et al., 2015) is used as an the reinforcement learning algorithm for the high-level heuristics, in order to find behaviour policies that works best for online problems. A policy is defined as a mapping between states and actions. It is often expressed using π . A good policy focuses on decision-makings with the greatest long-term returns rather than on immediate rewards. The long-term return can be calculated by accumulating reward signals in the long run. The long-term return at time $t \in [1, n]$ is a random variable, denoted as U_t . Its common definition is as Eq. (4).

$$U_t \stackrel{\text{def}}{=} r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + \dots + \gamma^{n-t} \cdot r_n \quad (4)$$

In this definition, the long-term return is equal to the weighted sum of all immediate rewards from the current time step to the end of an episode. Here $\gamma \in [0, 1]$ is called the discount rate and is a hyper-parameter. If it is close to 0, the RL agent is short-sighted and only cares about recent immediate rewards. U_t is a random variable, thus its value cannot be known until the episodes end. We can eliminate randomness by taking the expectation of U_t , which is used to evaluate how good a policy is. It has two kinds of variants: state-value function and state-action function. As Eq. (5), the former is defined $V_\pi(s_t)$ as the expected long-term return under policy π only based on state s_t , and Eq. (6) defines the expected return by taking action a_t in state s_t under policy π , as the state-action function $Q_\pi(s_t, a_t)$, it is also called the Q-value function.

$$V_\pi(s_t) \stackrel{\text{def}}{=} \mathbb{E}_\pi[U_t | s_t] \quad (5)$$

$$Q_\pi(s_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}_\pi[U_t | s_t, a_t] \quad (6)$$

In order to further evaluate expected long-term return without considering any certain policy π , the optimal Q-value function is defined

by Eq. (7) below.

$$Q_*(s_t, a_t) = \max_{\pi} Q_\pi(s_t, a_t) \quad (7)$$

Naturally, if $Q_*(s_t, a_t)$ can be acquired the RL agent can make best decisions following the Eq. (8) because always taking the best actions leads to the greatest long-term return. In reinforcement learning, to learn the optimal Q-value function and then use it to guide agents' behaviours is called value-based learning.

$$a_t = \operatorname{argmax}_{a \in \mathcal{A}} Q_*(s_t, a) \quad (8)$$

Modelling problems as MDP require to break long-term return into immediate rewards, through designing proper reward functions. This is crucial to problem solving as it allows the use of the Bellman's principle of optimality proposed by Bellman (1957), because applying this property allows to transform the original problem into sub-problems of at the next stage. Therefore, the optimal decision in the initial state can be found iteratively from the optimal decision in the final state step by step. The Optimal Bellman Equation in line with the optimal Q-value function is presented in Eq. (9).

$$Q_*(s_t, a_t) = \mathbb{E}_{s_{t+1}} \left[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q_*(s_{t+1}, a) | s_t, a_t \right] \quad (9)$$

The right term of the equation is at the current time step, while the left term is at the next time step. Applying the Monte Carlo approximation for the right term, we obtain a form of temporal difference in Eq. (10), which is the core of the Q-learning algorithm.

$$Q_*(s_t, a_t) \approx r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q_*(s_{t+1}, a) \quad (10)$$

Early versions of Q-learning algorithms used the look-up table method as the representation of the optimal Q-value function, such as the Williams (1992). Briefly, the look-up table method makes use of a table which stores the value of Q-value function for each state-action pair. It is simple but not suitable for unseen states that appeared in some complex or dynamic environments. In this paper, we use a specially designed neural network with parameter θ to be the approximator of the optimal Q-value function, as in the following Eq. (11).

$$Q_{\text{Predict}}(s_t, LLH_t; \theta) \approx r_t + \gamma \cdot \max_{LLH \in \mathcal{A}} Q(s_{t+1}, LLH; \theta) \quad (11)$$

The left part denoted as Q_{Predict} , is the prediction made by the neural network about the long-term return after performing the low-level heuristic LLH_t in state s_t , which does not have any factual component. The right term denotes Q_{target} , which contains a true observed reward r_t and a prediction related to state s_{t+1} . The left part should be encouraged to get closer to the right part that partially relies on factual observation. Thus, we use the mean-squared error (MSE) as the loss function described in Eq. (12).

$$\text{Loss}(\theta) = \text{MSE}(Q_{\text{Predict}}, Q_{\text{target}}) \quad (12)$$

Fig. 3 illustrates the Double Deep Q-learning Network adopted in this research, which has two neural networks with the same structure. In the top of this figure, it is shown that the online network itself is involved in the process of interacting with the environment, the target network is not involved in this interaction and is only used to calculate the target value.

To alleviate this drawback that the original Q-learning is vulnerable to overestimation of Q-values, consistent with the design in Mnih et al. (2015), we considers replicating a target network with the same structure as the online network the beginning of training, denoted as $Q^-(s_t, LLH_t, \theta^-)$ which is only used for calculating Q_{target} . The parameters of the online network are copied into the target network in every certain training step to synchronize. This separation can serve to moderate the self-propagation of overestimation.

The overestimation caused by the 'max' operator in Eq. (11) is

prevalent and leads to unstable decisions and poor local optimal solutions. The mechanism of double Q-learning can effectively overcome the overestimation in DQN based algorithms by separating the action selection and value estimation (Van Hasselt et al., 2016). It computes the right part of Eq. (11) within two steps: the first step is to find the action with the max Q-value through the online network, and the second step is to use the target network to calculate the Q-value with the action found in the first step. The new target value denoted as Q^-_{target} can be described as Eq. (13). It can be proved that Q^-_{target} is smaller than Q_{target} target obtained under the same conditions.

$$Q^-_{target} = r_t + \gamma Q^- \left(s_{t+1}, \underset{LLH \in \mathcal{A}}{\operatorname{argmax}} Q(s_{t+1}, LLH, \theta), \theta^- \right) \quad (13)$$

Finally, based on the Eq. (12) and Eq. (13), we can use the stochastic gradient-based algorithm such as the well-known Adam optimizer, to minimize the loss function, as the online network could be updated with a learning rate η , in line with Eq. (14).

$$\theta \leftarrow \theta - \eta \cdot (Q_{Predict} - Q^-_{target}) \cdot \nabla_{\theta} Q(s_t, LLH_t; \theta) \quad (14)$$

In the left part of Fig. 3, it can be seen that the trajectories generated by interacting with environments are stored in an experience memory. Once the memory buffer is full, the oldest trajectory is discarded for new ones. Such a mechanism of storing and sampling experience is known as the experience replay. It can improve training performance and efficiency compared to directly training with sequentially correlated trajectory sequences. In the process of RL agents interacting with their environment to gather experience, they naturally face a trade-off between exploration and exploitation. Using random behaviour facilitates exploration of various states in problem environments, while utilizing learned good policies allows to track experiences with better returns. In this paper we use a decaying ϵ -greedy exploration strategy. Detailly, based on a decaying positive parameter ϵ , at each time step the agent randomly selects a LLH with probability ϵ , otherwise it selects a LLH with the highest Q-value with probability $1 - \epsilon$. The decaying of parameter ϵ is shown in Eq. (15).

$$\epsilon = \epsilon_0 + \min \left(\frac{i}{f_{\epsilon} \times N}, 1 \right) \times (\epsilon_N - \epsilon_0) \quad (15)$$

Here N represents the total number of training steps planned and i is the current training step. ϵ_0 and ϵ_N are the start and stop values respectively, $1 > \epsilon_0 > \epsilon_N > 0$, and f_{ϵ} represents the ratio of the number of steps that stop decaying to the total number of training steps. In other words, if the training process exceeds this fraction f_{ϵ} , ϵ will remain at ϵ_N subsequently until the end of the whole training progress. According to the decaying ϵ -greedy strategy, it is possible to explore more at the beginning and gradually increase the exploitation of learned policies.

In the neural network architecture presented at the bottom of Fig. 3, the dueling architecture proposed in Wang et al. (2016) is applied. The main principle behind the dueling architecture is to decompose the Q-values into state values and action advantage values. This is due to the fact that a particular state's value is different from the value of the actions in that state. For example, if all actions from a particular state result in an episode ending with bad performance, the state has no value. Especially regarding the uncertain online problem environments, it will be possible to achieve better long-term returns if it is possible to distinguish between good and bad states in Q-value.

As illustrated in the bottom of Fig. 3, a common part of several dense layers works for processing the extracted features from the input layer, followed by two separate dense layers that express two parts of the optimal Q-value separately. The top output, labelled by V, which is a scalar, presents an estimate of the current state value. The bottom one marked as A, is the estimates of advantages of all actions compared to their average, which is a vector with the same dimension as the number of actions. Since the actions are LLHs in problem domains, the dimensionality of the vector is the number of LLHs as well. At last, these two

outputs are aggregated to together express the optimal Q-value in the form of Eq. (16).

$$Q(s, LLH, \theta) = V(s, \omega, \alpha) + A(s, LLH, \omega, \beta) - \frac{\sum_{LLH} D(s, LLH, \omega, \beta)}{|LLH|} \quad (16)$$

Here, three parameters ω, α, β are used to represent the total neural network more specifically. ω denotes the parameters of the common part, while α and β are the parameters of the two independent dense layers mentioned above. The first term refers to the state value while the second term is the advantage function which is related to both the state and LLH. The last term is the average of all advantage function values, which provide identifiability for the first two independent items. In contrast to the average level, the last two terms together relate to the advantage of the LLH chosen. Clearly, the average of all the outputs of the dueling neural network described above can be regarded as the state value. Our training algorithm and the pseudo-code is given in Algorithm 2.

4.3. Feature fusion

Algorithm 2 The training algorithm of deep reinforcement learning based hyper-heuristic.

Input: A set of predefined low-level heuristics $\{LLH\}$, an interactive environment ENV , discount factor γ , learning rate η , min-batch size M , total training step T .
Initialize *online network* Q with weights θ and *target network* Q^- with weights $\theta^- = \theta$.
Initialize *experience memory* D .
Initialize ϵ -greedy policy parameter ϵ .
set training step $i = 0$.
for $i < T$ **do**:
 obtain current observation of ENV as state s .
 select LLH by ϵ -greedy policy with *online network* Q in line with equation (8).
 perform selected LLH in ENV , acquire immediate reward r_t and new observation s' .
 add tuple $\langle s, LLH, r_t, s' \rangle$ to *experience memory* D .
 update $s \leftarrow s', i \leftarrow i + 1$.
 update ϵ according to equation (15).
 if enough tuples in D **do**:
 randomly sample M tuples from D as a mini-batch.
 for each tuple in the mini-batch, calculate Q^-_{target} with γ using equation (13).
 then update θ by performing gradient descent based on equation (14) with η .
 end if
 in every fixed time steps, update $\theta^- \leftarrow \theta$.
end for
Output: The *online network* Q with the latest θ .

It is well acknowledged that achieving good performance in deep learning is not only related to algorithm design, but also about good feature design. It is believed that the addressed online problems may also benefit from the utilization of multiple types of features and hence we need to find a way to exploit different types of features from real-time packing. Here a feature fusion method is introduced for representing uncertain environment states. Although reinforcement learning can generate and collect data through self-exploration, it is still crucial to select and extract appropriate features from interactive problem environments. Deep learning emphasises its end-to-end learning ability, however, in many cases, it does not work as well with just simple features. For examples, most image classifiers may not be able to identify objects with similar shapes and textures. Rainy days reduce the signal-to-noise ratio of automatic driving detection radars, making it impossible to drive safely. In the former case, features with low intra-class variance tend to lead to recognition errors, while in the latter case, uncertain noise degrades the quality of features and affects model's performance.

It is important to make features complement each other in order to avoid the inherent defects of single feature. Feature fusion is one of the information fusion technologies in the field of pattern recognition and the early studies mainly focused on the classifier level (Kittler, Hatef, Duin, & Matas, 1998; Mangai, Samanta, Das, & Chowdhury, 2010). In

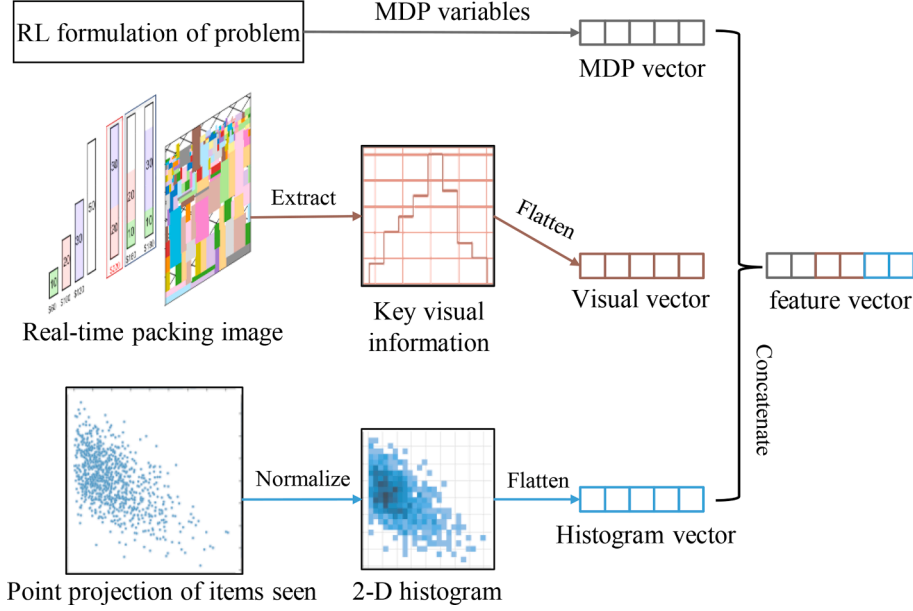


Fig. 4. State representation with feature fusion for online environments.

recent years, feature fusion has become a hot research area, and it is widely used for image recognition and recommendation systems (Fernando, Fromont, Muselet, & Sebban, 2012; Kiran, Kumar, & Bhasker, 2020; Zhang, Zhang, Peng, Xue, & Sun, 2018). Feature enhancement in image processing involves recombining abstract vectors at different levels to increase the sensitivity of the model (Chen, Papandreou, Kokkinos, Murphy, & Yuille, 2017; Zeiler & Fergus, 2014). In recommendation systems, knowledge graph embedding vectors of the recommended entries, are used as auxiliary side information to improve the accuracy and interpretability of the system (Liu et al., 2019). To address the challenge of uncertainty in the environment states in online combinatorial optimisation, this presented work explores the use of feature fusion to help represent uncertain environment states. Unlike the expansion strategy adopted in image processing, the fusion strategy is considered here by concatenating three vectors of different features, as shown in Fig. 4.

The first kind of features utilised in this mechanism is taken from problem models, which is a one-dimensional vector composed of all the variables derived from MDP formulations, here denoted as F_M , shown in the top of Fig. 4. It is the general representation of state in other studies using reinforcement learning to solve COPs (Hubbs et al., 2020; Kong et al., 2019). MDP features provide the most direct constraints and attributes of the objects that appear in online environments. For example, in the online 0–1 knapsack problem, the new state of problem environment is not only related to the weight and value of the newly seen items in the pipeline, but also based on the remaining capacity of the knapsack. These variables associated with problem constraints are clearly visible from the MDP formulation, which will be illustrated detailly in next section.

The visual vector, denoted as F_V , is shown in the middle of Fig. 4. It is naturally inspired by humans. As an example, an experienced worker can optimise the use of space during the loading process based on the contours of the packaging. Visual features reflect not only the current state of the environment, but also the cumulative impact of previous decisions. Images provide a great deal of information. In the well-known work of using deep reinforcement learning to play Atari, real-time screenshots were used to represent state (Mnih et al., 2015). Nevertheless, it met the requirement of 10 million training steps. Not all information such as the image colours is necessary for some classic problems. Furthermore, the uncertainty of online problems in this paper increases the demand for computing resources even more in comparison with

static games. So, we consider only extracting some key visual information to assist in enhancing the representation of the problem state. The highest contour of the packing, also known as Skyline (Burke et al., 2004), is used for the strip packing problem, and the value-to-weight density per unit information is adopted to the knapsack problem. The vector of visual information will be further described in the specific implementations.

This third type of feature pays attention directly to the sources of uncertainty. Though problem models are deterministic, unknown items in pipelines introduce uncertainties into environments, such as the lengths and widths of incoming rectangles in the strip packing problem, and the weights and values of newly seen items in the knapsack problem. In our feature fusion method, denoted below in Fig. 4, we consider using the histogram information of those stochastic attributes as the third component of all features. There are two main reasons for us to perform a data binning transformation, rather than directly using scatter information or storing all historical attributes. First, the deep neural network requires a fixed dimension input layer, and features can be reduced in dimensionality by transforming to histograms. Second, the histogram, as a compressed scatter plot, provides a more coarse-grained description of the data, which reduces overfitting and improves the robustness of the trained neural networks. For the two packing problems involved in this study, a 2-D histogram matrix is used as the last auxiliary feature, which can be converted into a vector after flattening. The global representation of environment state is a combination of the three kinds of features mentioned above, as Eq. (17).

$$F_{Global} = F_M + F_V + F_H \quad (17)$$

In addition to dimension reduction, our method also applies feature scaling before concatenating all features. Since deep learning models cannot preserve fundamental units when calculating, without feature scaling, features with a larger range may dominate the model training process. To make different features comparable, feature scaling is necessary. Furthermore, feature scaling can improve the speed of convergence when using gradient descent. The two most common methods for scaling features are Min-Max normalization and Z-score normalization. The Z-score normalization transforms the data into a standard normal distribution with an average of 0 and a standard deviation of 1 while the Min-Max normalisation is a linear transformation. In contrast to some classification algorithms that only focus on the boundary values of features, in our approach the intermediate values of

features are also important, such as the first type of MDP feature. Therefore, Min-Max normalization, the linear scaling method that preserves the original data distribution, is more appropriate than z-score. Thus, we apply the Min-Max normalization to scale different ranges of features to an uniform fixed interval of $[0,1]$ in most cases.

5. Applications

In this paper the proposed DRL hyper-heuristic approach is further implemented on two online packing applications, the online 0-1 knapsack problem and the online 2-D strip packing problem. This enables us to evaluate the cross-problem generality as well as the performance against the state of the arts. This section focuses on the implementations in the problem domains, including the MDP formulation, state representations, and employed LLHs to perform in problem environments.

5.1. Online 0-1 knapsack

In the 0-1 knapsack problem, the input is a sequence of N items and each item has a weight w_i and a value v_j . x_j is a binary variable for selecting ($x_j = 1$) or not selecting ($x_j = 0$) the j -th item into the knapsack. The fixed capacity of the knapsack is denoted as W . The goal is to maximize the total value of the items selected while ensuring the overall weight of these items is less than or equal to the capacity. The problem can be mathematically described as Eq. (18):

$$\begin{aligned} & \text{Max.} \sum_{j=1}^N v_j \bullet x_j, \\ & \text{Subject to} \sum_{j=1}^N w_j \bullet x_j \leq W \text{ and } x_j \in \{0, 1\}. \end{aligned} \quad (18)$$

For the online version of the 0-1 knapsack problem here, where on each item's arrival, it has to be accepted/rejected irrevocably without knowing properties of future arrivals. The online 0-1 knapsack problem can be formulated as a Markov decision process by collecting the value of each accepted item as the reward. Let F_w and F_v be distributions for the weights and values respectively, S_j be the total weights already consumed so far at step j . For $j \in [1, N]$, the state transition and the immediate reward are described as below.

Transition:

- Draw (v_j, w_j) from $F_v \times F_w$
- If $S_j + w_j \leq W$ and the action is 'Accept', then $S_{j+1} \leftarrow S_j + w_j$; else $S_{j+1} \leftarrow S_j$
- Set $j \leftarrow j + 1$

Reward: If $S_j + w_j \leq W$ and the action is 'Accept', then the reward is v_j ; else reward is 0.

5.1.1. Enhanced state representation

Some variables such as v_j, w_j, j, S_j can be directly derived from the MDP modelling, which provides information of incoming item for decision-making and other hidden constraints. Those variables $\left[\frac{v_j}{v_{\text{Max}}}, \frac{w_j}{w_{\text{Max}}}, \frac{j}{N}, \frac{S_j}{W}\right]$ compose the MDP vector F_M which is the first part in our feature fusion method to represent the environment states. v_{Max} and w_{Max} are upper bounds on their ranges respectively. It is noted that we prefer the relative ratio rather than absolute values in order to allow the learning algorithm to be scale-invariant.

The second part of state representations comes from visual information as introduced in section 4.3. While only S_j in the MDP vector can only provide the total consumed capacity, the visual vector pays more attention on details of history decisions. As illustrated in section 4.3, we consider extracting some key visual information to avoid increasing the demand of computing resources. For the online 0-1 knapsack problem, the value-to-weight ratio (aka the Bang-per-Bunk), denoted as $r_j = \frac{v_j}{w_j}$, is used as a kind of density map to describe how the capacity S_j is used by

each accepted item. Specifically, the visual vector F_V of d_V dimension will be filled with all zero before online process starts. Whenever an item is accepted, some elements in the corresponding position of F_V will be replaced with r_j , the updating of F_V follows Eq. (19) as below:

$$F_V \left[\left\lfloor \frac{S_{j-1}}{W} \bullet d_V \right\rfloor : \left\lfloor \frac{S_j}{W} \bullet d_V \right\rfloor \right] = r_j \quad (19)$$

Each side of the colon in Eq. (19) is the pointer to the position of the part of F_V that needs to be changed. The equation does not hold when left and right pointers are the same, indicating that the newly accepted item is relatively smaller, in which case we add r_j to the element at the latest position in F_V , to ensure the sum of all elements in F_V is linearly proportional to S_j . The F_V vector further enhances the state representation with respect to the impact of historical decisions. An example of the F_V could be of the form as $[r_1, r_2, r_3, r_3, \dots, r_j, \dots, 0]$.

Even though the first two feature vectors contain information about the current item and the previously accepted items, those ones rejected are ignored. A data binned transformation in histogram format can provide a coarse-grained description of all information of seen items. A vector of 2D histogram of the items seen so far acts as the last feature vector in the augmented state representation implemented in this method. A 2D matrix filled with all zero, denoted as $(h_{mn})_{d_H \times d_H}$, is used to initialize this 2D histogram. For $j \in [1, N]$, It will update itself when each item appears during the whole online process, in line with Eq. (20).

$$h_{mn} \leftarrow h_{mn} + \frac{1}{N}, m = \left\lfloor \frac{v_j}{v_{\text{Max}}} \bullet d_H \right\rfloor \text{ and } n = \left\lfloor \frac{w_j}{w_{\text{Max}}} \bullet d_H \right\rfloor \quad (20)$$

The matrix represents an approximator of the input distributions in each instance. After a flatten operation, the histogram vector F_H can be represented as $[h_{11}, h_{12}, \dots, h_{mn}, \dots, h_{d_H d_H}]$.

State: The enhanced state representation consists of three feature vectors described above, for the online knapsack problem it can be viewed in Eq. (21) as below:

$$F_{\text{Global}} = \begin{bmatrix} \left[\frac{v_j}{v_{\text{Max}}}, \frac{w_j}{w_{\text{Max}}}, \frac{j}{N}, \frac{S_j}{W} \right], \\ [r_1, r_2, r_3, r_3, \dots, r_j, \dots, 0], \\ [h_{11}, \dots, h_{mn}, \dots, h_{d_H d_H}] \end{bmatrix} \quad (21)$$

5.1.2. Low-level heuristics

In reinforcement learning, agents choose different actions to interact with environment s to maximise long-term returns. It is better for those actions to be complete, e.g. in an electronic game where moving up, down, left or right can constitute a trajectory to an arbitrary position. Therefore, the most atomic actions of problems are generally used in the reinforcement learning community. Corresponding to the online 0-1 knapsack problem in this section, the two options for newly emerging item in the pipeline are: "Accept" and "Reject".

In addition to considering these two basic actions as LLHs, the online Bang-per-Buck algorithm is adopted as the third constructive heuristic for this problem by some adaptations. The natural Bang-per-Buck algorithm was proposed by Dantzig (1957) and its online version was also mentioned in Kong et al. (2019). There is a parameter $0 < p \ll 1$ in the online Bang-per-Buck algorithm, which divides the online process into two stages. In the first phase for items $j \leq \lfloor N \bullet p \rfloor$ they are always accepted (when possible). Based on all items seen in the first phase, find the smallest value-to-weight ratio r^* that satisfies the capacity constraint as a threshold for decisions in the second stage. For remaining items that arrive after $j > \lfloor N \bullet p \rfloor$, accept (when possible) items whose value-to-weight ratio r_j are equal to or greater than the threshold r^* . This online algorithm can be understood here as a "Dual-Learning" program which aims to acquire the best online estimate for the related dual variable. In the third LLH we adopted, the threshold r^* is initially set to zero and automatically updated at each step until $j = \lfloor N \bullet p \rfloor$. At any

Table 1
LLHs for online 0–1 knapsack problem.

Low-level heuristic	Descriptions
LLH1: ‘Accept’	Accept the current item
LLH2: ‘Reject’	Reject the current item
LLH3: ‘Bang-per-Bunk’	Accept the current item if its value-to-weight ratio r_j is equal or greater than r^* .

Table 2
Low-level heuristics for online 2-D Strip packing problem.

Low-level heuristic	Descriptions
LLH0: ‘Leftmost’	The leftmost heuristic places a non-exact fitting rectangle at the left side of the niche.
LLH1: ‘Shortest Neighbour’	The shortest neighbour heuristic places the best-fitting rectangle within the gap next to the shortest gap-defining rectangle. If the lowest gap is defined by a rectangle and the sheet side, we place next to the sheet side.
LLH2: ‘Tallest Neighbour’	The tallest neighbour heuristic is the opposite of the shortest neighbour heuristic described above.
LLH3: ‘Rightmost’	The rightmost heuristic is the opposite of the leftmost heuristic described above.

time when the third LLH is selected by the high-level heuristic, the threshold r^* will be used as the criterion. Lastly the three LLHs used for the online 0–1 Knapsack problem can be found in Table 1.

5.2. Online 2-D strip packing

The 2-D strip packing problem consists in placing a set of N rectangles (items) into a rectangular area without overlap, such that the height H of the strip is minimized. Consider a fixed length L of the strip and denote the height and width of each rectangle are ht_k and wh_k , $\forall k \in [1, N]$. The position of a rectangle packed in is identified using its Cartesian coordinates of its bottom left corner, depicted as (x_k, y_k) . The constraint for a feasible solution is that each rectangle must be within the strip area and not overlap with the others. Thereby the 2-D strip packing problem can be formulated as:

$$\begin{aligned}
 &\text{Minimise } H, \\
 &\text{Subject to} \\
 &x_k + wh_k \leq L, \\
 &x_k + wh_k \leq x_l \text{ or } x_l + wh_k \leq x_k, \\
 &y_k + ht_k \leq y_l \text{ or } y_l + ht_k \leq y_k, \forall (k, l) \in [1, N] \text{ and } k \neq l.
 \end{aligned} \tag{22}$$

An ideal packing is considered to be one in which those rectangles are placed in this rectangular area with no overlap and no wasted space. We intend to minimise the average overall strip height with online settings. Since Reinforcement learning always maximises accumulated rewards, we look at the reward design for this problem as a way to minimise penalties. After every placement of an incoming rectangle, the new height of the strip is an opposite signal to the minimisation goal of the problem, so its negative value is considered a reward signal for the problem, thus the long-term goal has been successfully decomposed into each step in the online procedures. Specifically, let F_{ht} and F_{wh} be the distribution of heights and widths for each rectangle. For $k \in [1, N]$, S_k is defined as the height of strip occupied when the packing of the k -th rectangle is finished. The transition to the next state and the reward design in this MDP formulation of this problem can be described as below.

Transition:

- Draw (ht_k, wh_k) from $F_{ht} \times F_{wh}$,

- If $wh_k \leq L$, then pack the k -th rectangle into the strip and update S_k with respect to S_{k-1} and ht_k ; else $S_k \leftarrow S_{k-1}$,
- Set $k \leftarrow k + 1$.

Reward: The reward at k -th time step is set to $S_{k-1} - S_k$, which cannot exceed 0.

This reward design means that the best reward is 0 when the placement can be done without adding any height to the packaging, while any action that results in adding height will be scored as a negative. Certainly, rewards with positive values are impossible in all situations.

5.2.1. Enhanced state representation

To better represent states of online uncertain environments, we also applied the feature fusion method in subsection 4.3 to the online 2-D strip packing problem. The state representation was composed of three components, the MDP feature, the visual feature, and the histogram feature.

Four variables derived from the MDP formulation firstly compose the MDP vector F_M , $\left[\frac{ht_k}{L}, \frac{wh_k}{L}, \frac{ht_k \times wh_k}{L^2}, \frac{k}{N} \right]$. The first three variables represent the information about rectangle (i.e. the length, width and area) to be packed, relative to the strip fixed length. the fourth one which indicates the process, is the fraction of queue seen so far. The current height of strip S_k is not involved since all information of heights will be given in the in the second part, the visual vector F_V .

To utilise the visual feature, rather than processing real-time images of packing end-to-end, we prefer to extract some key information for reduce the demand of computing resources as noted in subsection 4.2. For this problem, the visual vector is composed of the skyline information proposed in Burke et al. (2004). The skyline describes the current contour of the packing using an array of the same length as the strip, which contains all heights corresponding to all positions along the length of strip. If a placement wastes some space in low places even it does not add the height of strip, it is not a best choice. Therefore, it is important to have skyline information to provide the lowest position for maximizing space utilization. For $m \in [1, L]$, denote sk_l is the current height corresponding to the position l along the bottom of the strip, the vector of $[sk_1, sk_2, sk_3, \dots, sk_L]$ can be used to represent the second part of features.

The last part of enhanced state representations, the histogram vector F_V , is similar to that in section 5, since in these two problems the incoming items or rectangles have two dimensions of properties. Thus, for this online 2-D strip packing problem a same matrix $(h_{mn})_{d_H \times d_H}$ filled with all zero, is used to initialize the histograms of rectangles’ information. When each rectangle appears, it will update itself according to Eq. (23).

$$h_{mn} \leftarrow h_{mn} + \frac{1}{N}, m = \left\lfloor \frac{ht_k}{L} \bullet d_H \right\rfloor \text{ and } n = \left\lfloor \frac{wh_k}{L} \bullet d_H \right\rfloor \tag{23}$$

State: To summarise the global state representations, it can be aggregated as Eq. (24).

$$F_{Global} = \begin{bmatrix} \left[\frac{ht_k}{L}, \frac{wh_k}{L}, \frac{ht_k \times wh_k}{L^2}, \frac{k}{N} \right], \\ [sk_1, sk_2, sk_3, \dots, sk_L], \\ [h_{11}, \dots, h_{mn}, \dots, h_{d_H d_H}] \end{bmatrix} \tag{24}$$

5.2.2. Low-level heuristics

Due to the simplicity and good performance of the best-fit algorithm (Burke et al., 2004), three niche-placement heuristics obtained from it are served as low-level packing heuristics to perform in the present implementation, shown in Table 2. Each low-level heuristic indicates how a rectangle could be placed when it does not fit exactly into the

Table 3

General configurations and settings.

Name	Configurations	Hyperparameter	Value
Processor	Intel Core i5 @3.00 GHz	Discount factor	0.995
Memory	32 GB	Learning rate	5e-5
Graphics	NVIDIA GeForce GT 730	Replay buffer size	50,000
OS	Ubuntu 16.04 LTS	Batch size	128
Python	3.6.10	Number of timesteps	2,000,000
TensorFlow	1.12	Update frequency	500
		Final value of exploration	0.01
		Initial value of exploration	0.9

Table 4

Two datasets with Different settings.

Dataset	XY-type	U-type
values of items	{1}	$\mu[0, 100]$
weights of items	{1, 4, 16}	$\mu[1, 100]$
W	250	200
N	200	50

Table 5

The mean objective values achieved in two datasets.

Dataset	Method	Mean Obj.	Std. Dev. on instances	vs. Heuristic	Gap% to Opt.	p-value
XY-type	DP (Offline)	99.6	37.8	/	/	0.2
	Bang-per-bunk (Dantzig, 1957)	77.2	25	1	23%	<0.01
	Reinforce (Kong et al., 2019)	95.1	36.8	1.2	5%	0.2
	DRLHH w/o Feature fusion	95.8	37.7	1.1	4%	0.3
	DRLHH with Feature fusion	97.1	37.3	1.3	3%	Nan
U-type	MILP (Offline)	531	54.6	/	/	<0.01
	Twobins (Han et al., 2015)	262	104	1	51%	<0.01
	PPO (Hubbs et al., 2020)	309	54.2	1.2	42%	<0.01
	DRLHH w/o Feature fusion	366.4	71.9	1.6	31%	<0.01
	DRLHH with Feature fusion	408.9	80.2	1.8	23%	Nan

lowest niche or gap in the skyline. It is noted that the last heuristic LLH3 is a newly added one which is not included in the best-fit (Burke et al., 2004). The purpose of the rightmost heuristic is to introduce a degree of diversity and completeness.

6. Experiments

We present our experiments and evaluation results in this section. Algorithms were trained and tested on the same server equipped with the configurations listed in the left part of Table 3. Based on the powerful multi-core processor with 6 cores, we performed a parallelisation of the training. All neural networks are implemented in TensorFlow (Abadi et al., 2016) and we used the rectified linear unit function (ReLU) as the activation function and adopted the Adam optimiser.

6.1. Experimental setups of DRL

Neural network. For the online 0–1 knapsack problem, since the MDP vector was 4 dimensions and d_V and d_H were respectively set to 20 and 5 for coarse-grained perceptions, therefore the input layer was 49 dimensions. Regarding the hidden layers, different numbers of layers and nodes were tested and a combination of 5 hidden layers and 256 nodes per layer was chosen, so the neural network’s common part for processing input features was composed of 3 dense layers of 256 nodes each, followed by 2 dense layers served as two separate streams. For three LLHs described in section 5.1.3, the output layer produced three Q-value estimates. For this online 2-D strip packing problem, the input layer of neural networks has 189 dimensions as the same as the state vector, which mainly composed by the Skyline array in 160 dimensions and the 2-D histogram matrix in 25 dimensions. After trying several combinations, we applied 5 hidden layers and 256 nodes per layer. There are four low-level pacing heuristics, resulting in a 4-dimensional output layer.

Hyperparameters. The hyperparameters applied in the experiments are listed in the right part of Table 3. These parameters are either drawn from the classic DRL implementations (Hessel et al., 2018; Mnih et al., 2015) or are based on some trial testing. Therefore, fine-tuning of those hyperparameters with extensive experiments may lead to better results, but this is not the focus of our work. As a result of experimentation, we trained our neural networks with a fixed learning rate η of 5e-5, and the discount factor γ is set to 0.995. The replay buffer size was set as 50,000. The total number of timesteps (equal to the number of executions by agent in environment) in the whole training process was 2 million. At every step in the training process, a batch size of 128 trajectories was randomly drawn from the experience memory to train neural networks. In every 500 timesteps, the values of parameters of the online network will be copy to the target network. For the decaying ϵ -greedy strategy, where the parameter ϵ started at 0.9 and held 0.01 for fine tuning at the last 10% stage during the training process.

6.2. Online 0–1 knapsack

Baseline. In two recent studies, Kong et al. (2019) and Hubbs et al. (2020), both addressed the online 0–1 knapsack problems by using the recent DRL approaches. To our best knowledge, they are the best algorithms in terms of the average-case performance, for online version of this classic problem. Thus, our approach was compared with these two recent DRL methods and other two human designed heuristics. To illustrate the effectiveness of feature fusion method employed, we also tested the version without enhanced state representation.

Dataset. Two types of datasets were considered to evaluate our approach’s performance. The first set, labelled as XY-type in Table 4 is characterised by mixed distributions, while the second one, the U-type set has relatively small capacity constraint.

Kong et al. (2019) introduced the former XY-type set with very few types of item, similar as that shipping containers generally has very few common classes. However, the distributions of instances in this dataset are quite different. In the XY-type set there are two different kinds of instances X and Y. The X-type instances don’t have item which weight is 16, while the Y-type instance don’t have item which weight is 1. The instance that appears in the XY-type dataset may be either X or Y thus

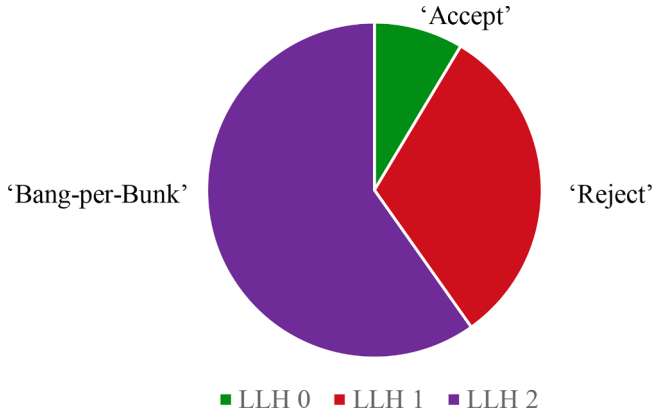


Fig. 5. The distribution of LLH selections for online knapsack problem.

the optimal solution is different depending on which type of instance is being dealt with. Thus, any single threshold on the XY-type set will result in suboptimal performance for either X or Y instances.

The latter one U-type are derived from Hubbs et al. (2020). $F_v \mu[0, 100]$ and $F_w \mu[1, 100]$ are the distributions of the values and weights of items respectively. Therefore, there are over 10^4 kinds of item to appear in an instance. The capacity constraint of each instance is the same, but it is relatively very small in comparison with the range of weights, only being twice the maximum limit. The U-type set is somewhat representative of real-life scenario such as portfolio investments, where choices are abundant, but one choice may cost a lot sometimes. For the XY-type and U-type dataset each, 500 instances with random seeds were generated to act as the related test sets. The training instances for the XY-type and U-type were generated in real-time during training processes, meanwhile matching the two different distributions of datasets respectively. This allows the DRL agent to explore a wider range of scenarios.

We trained neural networks and performed experimental evaluations on the XY-Type and U-Type set, respectively. Due to our parallel implementations and the linear complexity of state representations, the training time of our proposed approach took almost two more hours compared to the version without feature fusion, but both were less than one day. Table 5 compares the average of the cumulative values achieved on test sets while satisfying their capacity constraints.

Performance Comparison. On the XY-type set, the Dynamic Programming (DP) was used to demonstrate the performance of offline optimal solutions in the first line in Table 5, and it was marked as the upper bound of the other online algorithms. In the second row, as a heuristic baseline, the well-known online Bang-per-bunk algorithm (Dantzig, 1957) achieved a performance of 77% to optimal solutions. The third one is the use of Reinforce algorithm to address the online knapsack problem by Kong et al. (2019), with achieving 95% to optimal solutions. Our approach achieved the best performance, with a performance of 97% to offline optimal solutions.

The greedy heuristic of Bang-per-bunk uses a fixed strategy (a certain threshold) to select items, which seems very natural and efficient. Nevertheless, Table 5 shown that this algorithm could not handle discrete value-to-weight ratios well and different distributions on instances. For the XY-type set, it is necessary for an optimal policy to employ different policies on different types of instances separately, which requires to accept items with different probabilities as opposed to a fixed threshold. This is the reason of which forms the gap between solutions of the online Bang-per-bunk and the optimal solutions. Those DRL based methods all outperformed the baseline heuristic and their performances were much closer to optimal solutions at a same level. It proved that they all learned the successful strategy that items can be accepted and rejected differently based on the type they belong to. Our approach performed better compared to Kong et al. (2019), another DRL

approach using the Reinforce algorithm, by more diversity of actions and enhanced feature representations in our approach. Oliveto (2021) and Burke, McCollum, Meisels, Petrovic, and Qu (2007) also have shown that it is beneficial to combine multiple heuristics rather than applying any of them alone. Due to only using MDP vectors as the state representations, the version without feature fusion performed slightly worse.

On the U-type set, the performance of optimal solutions was given through the Mixed-Integer Linear Programming (MILP) solvers. Items in the U-type set are more variable, with a difference between 0.01 and 100 in their value-to-weight ratio. As shown in Table 5 this leads to a relatively large difference between each instance. It should be noted that the standard deviations here are based on values from various instances, not on the mean values. The first online algorithm evaluated is a randomized algorithm called *TwoBins* (Han et al., 2015). It virtually keeps two bins and puts items into either of the bins if possible and results show its performance was of almost 50% to the optimal. Another counterpart is Hubbs et al. (2020), which used the Proximal Policy Optimisation (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), a DRL algorithm to tackle the same problem. On this set it performed nearly 60% close to the optimal, with only using basic actions and mainly relying on MDP variables as its state representations. Results indicate that our proposed approach is the best one among all online algorithms, with achieving a performance of 77% to the optimal. Additionally, we also tested the version without feature fusion and found its performance fell by 8%. However, it still was better than the other two counterparts.

In addition, the p-values (from one-tailed paired samples t-tests) are provided in the rightmost column in Table 5 to illustrate significance levels of difference between our method and others. On the XY-type dataset, the difference from the Bang-per-bunk algorithm (Dantzig, 1957) is significant, while it is not enough to distinguish our methods from the other two methods. However, the p-value between ours and the offline DP method that provides the upper bound, shows that our method has already achieved fairly good performance. On the U-type dataset, all the p-values are less than 0.01, which suggests that the superiority of our approach is statistically significant against the others.

Behaviour analysis. We recorded a total of 17,365 state-LLH pairs based on the U-type set, for analysing behaviour patterns of the learned selection policy by our approach. The distribution of LLH selection is shown in Fig. 5, where the green part of LLH0 is “Accept”, which only takes 8% of all actions. The red one of LLH1 is “Reject”, which accounts for 32%. The purple part is the biggest one, representing the LLH2, which takes nearly 60% of all. Because of the tight constraints in the settings, it is reasonable to see more “Reject” in red relative to “Accept” in green. However, the learn strategy decided to employ LLH2 in most cases, which was different from our expectation. The initial assumption was that the learned strategy utilises LLH2 as a complement, mainly focusing on combining LLH0 and LLH1 to some complex strategies. In fact, the behaviour of the strategies showed more preference to use the heuristic with domain knowledge embedded.

For concerning the selections made by the neural network at different states, we created a demonstration by visualizing state-LLH pairs along with temporal information. We first reduced the dimensionality of state vectors by using the t-distributed stochastic neighbour embedding (t-SNE), which is a nonlinear dimensionality reduction technique (Van der Maaten & Hinton, 2008). To avoid making it difficult to distinguish between too many markers, we randomly selected 750 state-LLH pairs to show in Fig. 6. The position of points represents the vector of different states, and the colour and shape of the points indicate the decision made by the learned strategy for each state. The shades of colouring points correspond to the progress of episodes, which indicate the stage when the decision is made. In other words, the lighter markers the earlier stage, and the darker markers the closer to the end of episode.

As shown in the plan view on the left of Fig. 6, the state vectors are found to have no obvious aggregation, indicating that the state vectors vary widely, which is consistent with the big variety of items and the uniform distribution. An observation is that the LLH selection by the

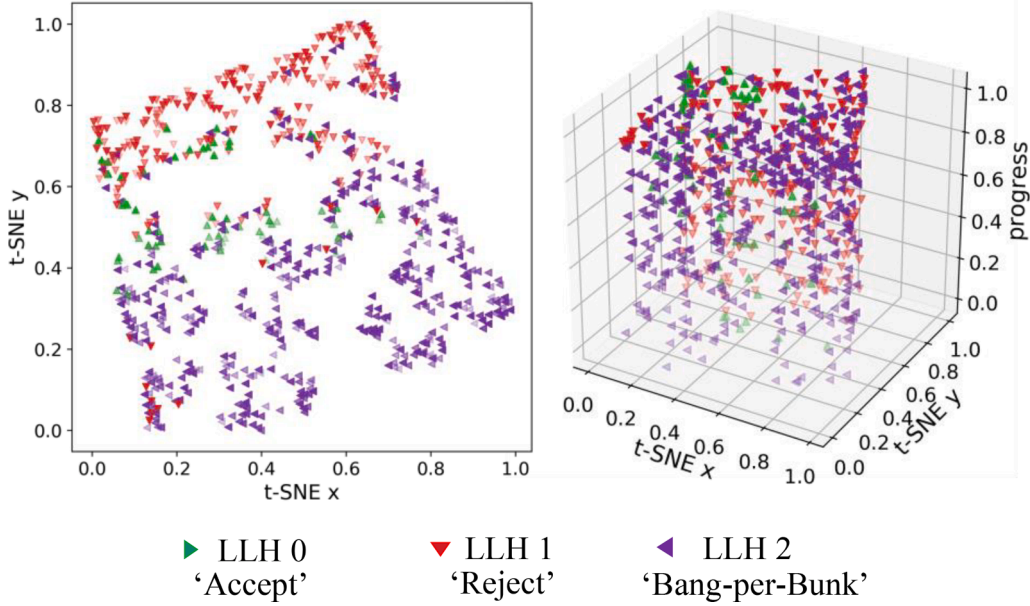


Fig. 6. t-SNE visualization of LLH selection for online knapsack problem.

Table 6

The average heights achieved by different approaches for the online 2-D strip packing.

Method	Best-fit (Offline) (Burke et al., 2004)	GSHA (Han et al., 2016)	Online Best-fit with fixed strategy (Burke et al., 2004)	Online Best-fit with stochastic strategy (Burke et al., 2004)	DDQN Hyper-heuristic (Zhang et al., 2021)	DRLHH with feature fusion
Mean height	245.6	374.3	271.7	271.9	269.1	267.5
Std. Dev.	0.48	18.5	18	17.4	18.3	17.4
Rank	/	/	3.2	3.2	2.2	1.3
p-value	/	/	<0.01	<0.01	<0.01	Nan

Table 7

Results of Scalability Experiment. The average heights under different levels of uncertainty.

Method	Level of uncertainty = 5		Level of uncertainty = 100		Level of uncertainty = 150	
	Mean height/ Std. Dev.	Rank	Mean height/ Std. Dev.	Rank	Mean height/ Std. Dev.	Rank
Best-fit with fixed strategy	258.4/2.6	3.3	275.7/20.5	3.4	268.7/12.3	3.2
Best-fit with stochastic strategy	258.9/2.7	3.4	276/20.4	3.3	269/12.7	3.2
DDQN Hyper-heuristic (Y. Zhang et al., 2021)	256.6/2.5	2	273.7/20.3	2.1	267.4/12.4	2
DRLHH with feature fusion	254.1/2.4	1.3	271.4/21.1	1.2	265.3/13.3	1.6

learned strategy was not uniform. Those markers located in the above region were mostly for rejection, showing that the neural network-based high-level heuristic considered those states to be of low value in terms of the item involved. Those markers located in the below area were mostly for conditional acceptance or rejection by LLH2. Therefore, despite the black-box nature of neural networks, we argue that the learned strategy by our approach was able to distinguish between different states in uncertain online environments. To better inspect the decisions made at different stages, we added a temporal dimension to show the right 3D

diagram in Fig. 6. For the “Reject” action of LLH 1 in red, there are more dark colours compared to the lighter colours in the earlier stages. This suggests that the learned strategy was performed with different criteria at different stages of online processes, rather than a single metric for all stages. Such a pattern reveals that applying stricter acceptance conditions at later stages may help to maximise the total values under relatively tight capacity constraints.

6.3. Online 2-D strip packing

Baseline. Our proposed method was also evaluated on the online strip packing problem. Two variants of the best-fit heuristics (Burke et al., 2004) were used to serve as the baselines, the first variant is the online best-fit fixed-strategy, which only selects a fixed placement strategy throughout the whole packing process whilst the second variant is online best-fit with stochastic strategy, which randomly selects different placement strategies at different decision points. According to a review of packing algorithms (Christensen et al., 2017), the grouped super harmonic algorithm (GSHA) proposed by Han, Iwama, Ye, and Zhang (2016), which is commonly considered as a state-of-the-art online method, was also included in the comparison. As far as we know, a recent DRL Hyper-heuristic study by Zhang et al. (2021), represents the state of the art approach for the online version of 2-D strip packing problem, reporting the best average-case performance. This competing hyper-heuristic also employed the Deep Q-learning Network but lacked full enhanced state features, served as our last competitor, to show the difference of the version without feature fusion.

Dataset. The same data set as in Burke et al. (2004) was employed. The data source contains seven categories (C1 to C7), each containing three instances (P1, P2 and P3). Further, a new instance generator was

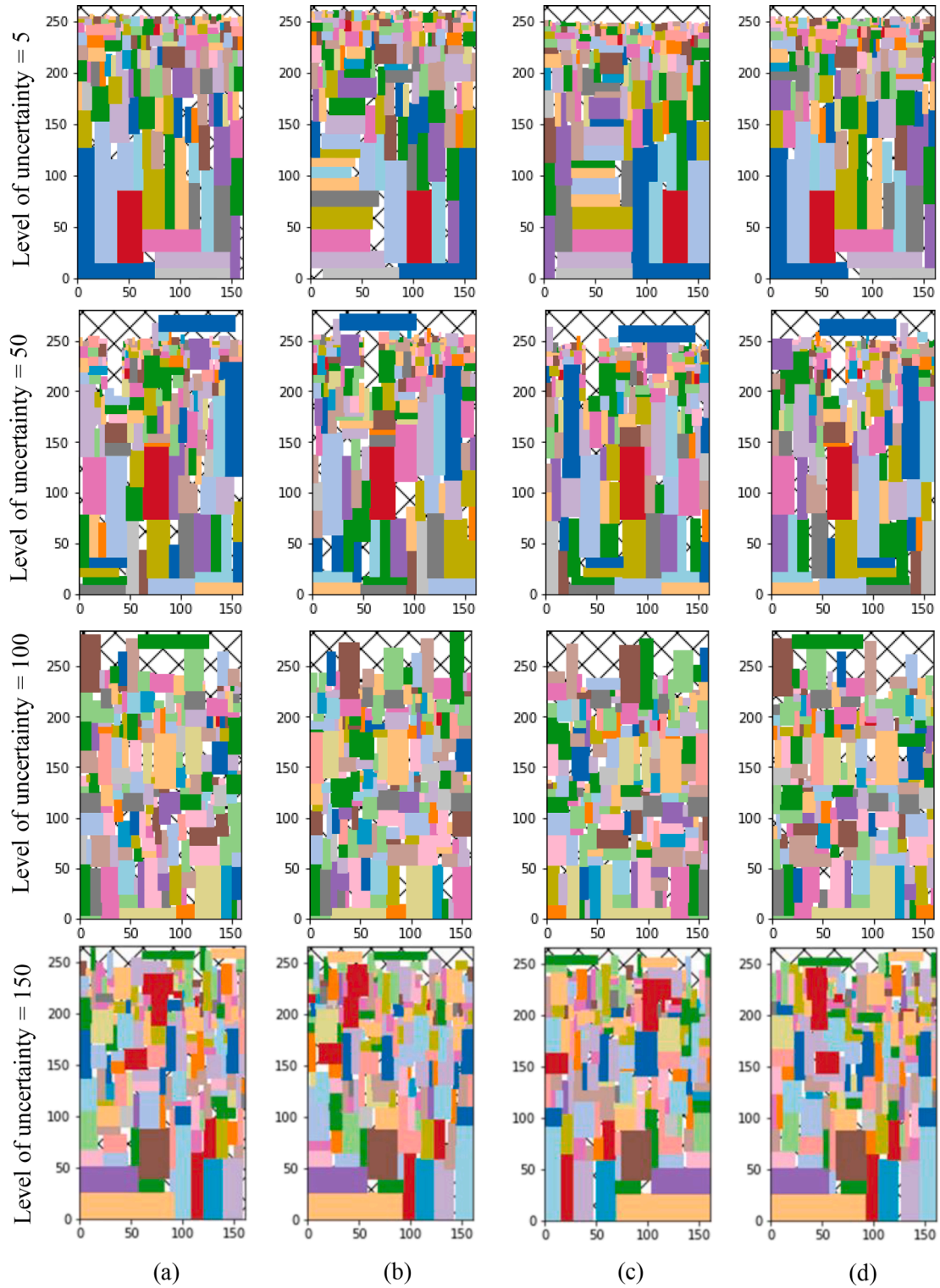


Fig. 7. Online strip packings under four levels of uncertainty. The packings in each column were obtained separately by (a) online best-fit with fixed strategy (Burke et al., 2004), (b) online best-fit with stochastic strategy (Burke et al., 2004), (c) our proposed approach and (d) DDQN (Zhang et al., 2021).

adopted to generate more diverse instances based on C7. By controlling how many rectangles in the original instance change their position and how far away they can move, different levels of uncertainty have been introduced into the newly generated instances. Based on a setting of randomly changing rectangles' positions with a maximum distance of 50 in P1, P2 and P3, there are a total of 1000 different instances created to provide a sufficient diversity of online instances, half of which acted as

the training set and the other half of which acted as test set. For testing the scalability of the learned strategy under different levels of uncertainty, three additional test sets of 500 instances each were produced, by setting the maximum movable distance to 5, 100, and 150 respectively.

The training time was longer than in the previous problem, but no more than two days since the interaction with environments was more relatively time-consuming. Table 6 compares the average heights of the

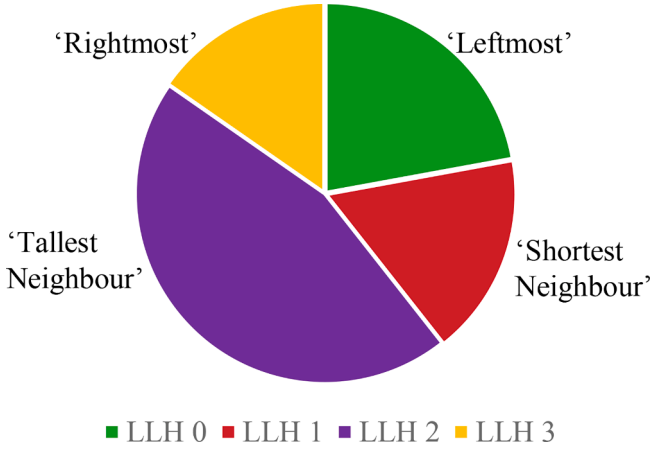


Fig. 8. The distribution of LLH selections for online strip packing problem.

strip achieved by four approaches over 500 different instances, on the test set with the level of uncertainty being 50. The value of height is smaller implies that the method has a better space utilisation.

Performance comparison. By the best-fit (Burke et al., 2004), we obtained an offline performance on the test set. The result of 245.6 shows a very small gap to the theoretical optimal value of 240. For online algorithms, the recently proposed GSHA algorithm, to our surprise, did not perform well. This is probably due to its lack of internal learning mechanism to the given problem set. Our proposed approach demonstrated the best performance among all online algorithms, with a 0.7% reduction in optimality gap. For better comparing the other four online algorithms, we supplemented the statistical ranking, even though the numerical differences are not significant on their performances. The p-values (by one-tailed paired samples t-tests) are also provided in the last row in Table 6 in order to show the significant level of difference between our method and the other three online algorithms. Results show that our proposed approach demonstrated the best performance among all online algorithms and despite the slightly degraded performance of the version without feature fusion (Zhang et al., 2021), in the most left two DRL approaches were still significantly better than the two online best-fit variants.

Scalability. We further evaluated the performance of the four online algorithms against the other three test sets as the scalability experiment, where the levels of uncertainty were 5, 100, and 150. Related results are

shown in Table 7. An observation is that even though the strategies learned by our method were not trained in the other three cases, they were generalizable to similar online scenarios.

Those two DRL approaches retained their advantage over the two online variants of best-fit, although advantage started to weaken when the level of uncertainty was set to 150. On the test set of the level of uncertainty being 5, subject to the relatively small level of uncertainty, which results in the lowest average height. In the test set, where the rectangle moveable range was set to 100, almost half the queue length, the rectangle sizes vary greatly in the intermediate stages of the online process, thus led to the highest average strip height. Fig. 7 shows some examples of final packings. Each row in the figure represents a randomly selected online instance under different uncertainties, while each column stands for a method. The white mesh background is the space wasted during the packing process. It is clear that the packings by our approach in column (c) had the greatest remaining distance at the top, indicating that there was relatively little wastage in its interior, while in the other packings in other columns, there was more space wasted. This reveals that our proposed approach yielded best space utilisations. The DRL approach without feature fusion (Zhang et al., 2021) shown relatively different solutions and achieved suboptimal performance.

Behaviour analysis. To further examine how did the learned strategy select for the low-level packing heuristic for minimising the total heights, we recorded a total of 19,633 state-LLH pairs from our test experiments. Fig. 8 shows the distribution regarding LLH selections, where the purple one represents the LLH2 of the Tallest Neighbour packing heuristic, which accounts for 45%. The rest part of 55% is shared by the other three low-level packing heuristics. The learned strategy preferred to apply LLH2, while still utilising other heuristics, rather than consistently selecting a placement strategy in the best-fit algorithm.

We randomly selected 750 state-LLH pairs from the recorded pairs for t-SNE visualization and the visual presentation is given in Fig. 9. In the left sub-figure, we can see the state vectors for this problem are not very dispersed compared to the previous problem. This is probably due to two reasons: first, the total number of rectangles in C7 is less than 600, whereas in the knapsack problem the number of items is over 10^4 ; second, the placement strategy in our approach are heuristics involving human knowledge rather than arbitrary actions, which reduces the state space significantly, e.g. to place rectangles always in the lowest gap rather than to put it in any position. Therefore, the state vector positions appear more concentrated. In the 3D diagram on the right of Fig. 9, we find that the learned strategy did not always select LLH2 during the

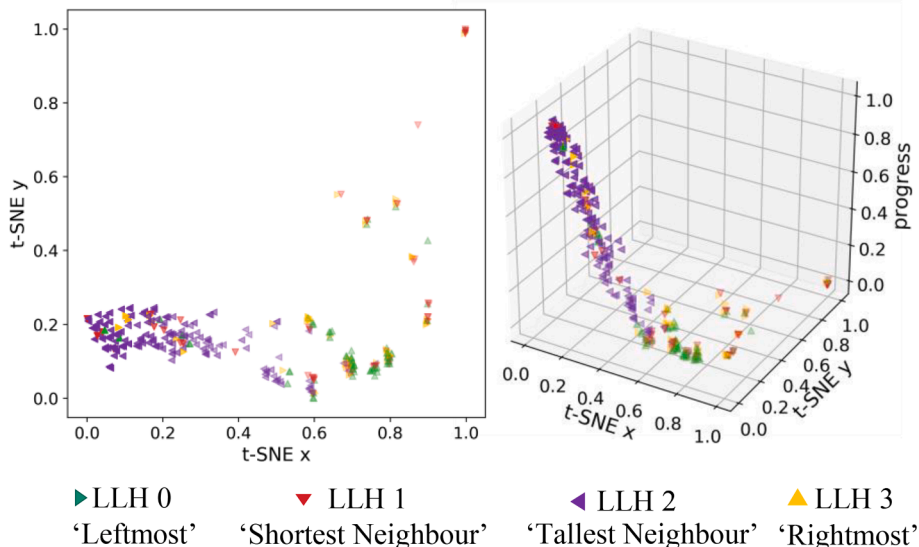


Fig. 9. t-SNE visualization of LLH selection for 2-D strip packing.

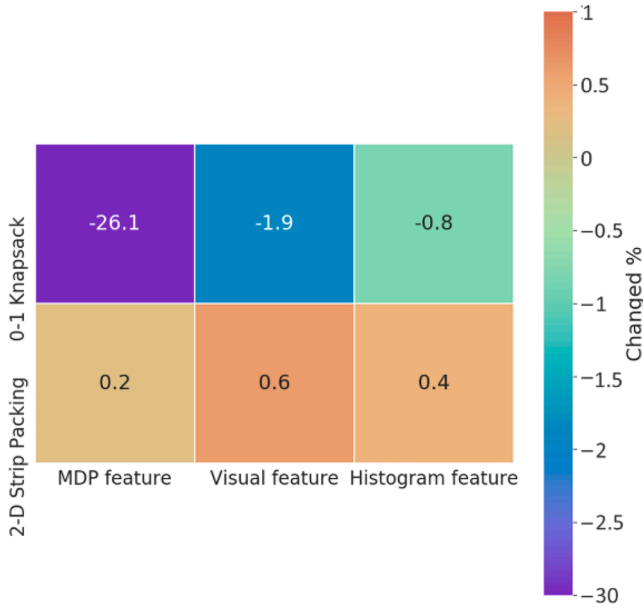


Fig. 10. The mean objective value change in percent by feature ablation.

whole process despite its greater weight in the distribution of Fig. 8. This may be because some rectangles in each instance still preserve their positions, rectangles with larger sizes are more likely to be presented in the early stages. To apply LLH2 to those one that are much larger than the gap height is inefficient, resulting in multiple tower spikes which are difficult to handle further. At later stages more small rectangles appear and LLH2 allows packing of those small one adjacent to large existing rectangles without increasing the height of the whole strip and does not form new gaps. In contrast to fixed and random strategies, we believe that the learned strategy by our approach can take full advantage of the phase characteristics in the online processes, which shows a multi-stage behavioural policy that is adaptive to different instance distributions, thereby improving the performance in online environments.

6.4. Feature importance analysis

According to the experimental results of the online packing problems, our proposed method with feature fusion performed better than other counterparts without feature fusion. Since our feature-enhanced state representations contain three different types of features, we evaluated the effects of each different feature type further. Because the neural networks trained by reinforcement learning produce a vector of values for heuristic selection rather than an objective value in other machine learning, it is meaningless to analyse feature importance for specific state inputs. Specifically, in our feature ablation experiments, the real values in vector of different feature type were individually replaced with their mean values based on the whole test set. Fig. 10 illustrates how much the change in mean objective value is, and reflects the extent to which each feature impacts the sequential decision process.

Based on feature ablation, we conducted the feature importance analysis for the two packing problems applied in this work. The first row of the figure shows the 0–1 knapsack problem, which is a maximisation problem. After masking on each type of feature we can see that all objective values were all decreased. When the MDP feature was occluded, performance suffered the most, declining by 26%, and followed by the visual features with a decline of 1.9%. This suggests that the MDP feature appears to be heavily involved in the decision-making for this problem. We believe this is because in the Knapsack problem, the MDP feature contains critical constraints that affect the LLH selection, such as the remaining capacity of the knapsack. The second row in Fig. 10 is about the 2-D strip packing problem, in which the desired

objective values were all increased by blocking each type of features individually. As can be seen, the most important feature is the visual feature with an increase of 0.6%, while the MDP one acted as the smallest role. The reason is that the LLHs, the placement strategy based on the best-fit heuristic, which heavily relies on skyline information to determine where to place the rectangle.

7. Discussion

It is always expected that computers will be able to handle different problems as well as humans. This includes whether the algorithm solves multiple problems and the cost of applying it to different problems. The proposed approach combines DRL Hyper-heuristic and feature fusion to deal with online problems. Hyper-heuristic can solve cross-domain problems more easily (Drake et al., 2020). When applying it to other problems, only problem-dependent components need to be replaced, without changing algorithms in control domain.

In problem domain, the low-level heuristics could contain atomic actions and problem-specific rules that already exist, without having to be redeveloped. Although some poorly designed heuristics/rules may lead to poor performances if used alone, the proposed DRL hyper-heuristic has ability to learn the strengths and weaknesses of different heuristics and intelligently combine them according to different problem-solving states. When it comes to feature fusion, this work shows the importance of the visual abstraction of solutions. However, if providing all raw visual information, it may lead to poor performance and cause more difficulties in training neural networks (even the convergence is not guaranteed). Therefore, when extracting key visual features for different problem, some domain knowledge is needed.

8. Conclusions and future work

Uncertainty in online environments poses a significant challenge to traditional packing algorithms. The paper presents a reinforcement-learning-based hyper-heuristic for solving two classic online packing problems, the 0–1 knapsack problem and the 2-D strip packing problem. Our approach provides a generic framework, which not only uses existing heuristics derived from human domain knowledge, but also adopts feature fusion to better characterize uncertain online environments. Experimental results show an obvious advantage of our approach for both different problems in average-case performance. The behaviour analysis reflects that the learned strategy by neural networks has the characteristic of multi-stage decision-making, through making full use of the distributions in instances. As a result of the linear complexity in the enhanced state representation, our approach does not require powerful GPU to process real-time images, unlike other end-to-end image processing tasks. Even though different problems rely to a varying degree on different types of features, our proposed approach can be beneficial in improving average performances when facing online environments with uncertainty, at the cost of only negligible computing resources.

In further work, considering the discrepancy between the decisions learned by the neural network and human intuition, better methods can be explored to analyse their decision sequences. It would be very interesting to use the patterns and knowledge gained from these analyses to improve the design of traditional heuristics which do not need offline training. Also, neural networks with flexible structures that automatically adapt to different input dimensions may be explored, as well as their application to other real-life scenarios in booking systems, dynamic pricing and revenue management.

CRediT authorship contribution statement

Chaofan Tu: Conceptualization, Methodology, Writing – original draft, Software, Investigation. **Ruibin Bai:** Supervision, Writing – review & editing, Funding acquisition. **Uwe Aickelin:** Supervision,

Investigation, Project administration. **Yuchang Zhang**: Data curation, Validation, Visualization. **Heshan Du**: Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgements

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

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... Isard, M. (2016). *Tensorflow: A system for large-scale machine learning*. Paper presented at the 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16).
- Abdel-Basset, M., El-Shahat, D., & Sangaiah, A. K. (2019). A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem. *International Journal of Machine Learning and Cybernetics*, 10(3), 495–514.
- Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, 10(1), 43–66.
- Bai, Ruibin, Chen, Xinan, Chen, Zhi-Long, Cui, Tianxiang, Gong, Shuhui, He, Wentao, Jiang, Xiaoping, Jin, Huan, Jin, Jiahuan, Kendall, Graham, Li, Jiawei, Lu, Zheng, Ren, Jianfeng, Weng, Paul, Xue, Ning, & Zhang, Huayan (2023). Analytics and machine learning in vehicle routing research. *International Journal of Production Research*, 61(1), 4–30. <https://doi.org/10.1080/00207543.2021.2013566>
- Baker, B. S., Coffman, J., Edward, G., & Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on computing*, 9(4), 846–855.
- Bayliss, C., Currie, C. S., Bennell, J. A., & Martinez-Sykora, A. (2019). Dynamic pricing for vehicle ferries: Using packing and simulation to optimize revenues. *European Journal of Operational Research*, 273(1), 288–304.
- Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5), 679–684.
- Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.
- Ben-David, S., Borodin, A., Karp, R., Tardos, G., & Wigderson, A. (1994). On the power of randomization in on-line algorithms. *Algorithmica*, 11(1), 2–14.
- Blum, C., Pinacho, P., López-Ibáñez, M., & Lozano, J. A. (2016). Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68, 75–88.
- 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(12), 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., Hyde, M., Kendall, G., & Woodward, J. (2010). A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6), 942–958.
- Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1), 63–89.
- Burke, E. K., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4), 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(1), 177–192.
- Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, ATRous convolution, and fully connected CRFS. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4), 834–848.
- Chen, X., Bai, R., Qu, R., & Dong, H. (2022). Cooperative double-layer genetic programming hyper-heuristic for online container terminal truck dispatching. *IEEE Transactions on Evolutionary Computation*. <https://doi.org/10.1109/TEVC.2022.3209985>.
- Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 6281–6292.
- Choong, S. S., Wong, L.-P., & Lim, C. P. (2018). Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436, 89–107.
- Christensen, H. I., Khan, A., Pokutta, S., & Tetali, P. (2017). Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24, 63–79.
- Colomi, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., & Trubian, M. (1996). Heuristics from nature for hard combinatorial optimization problems. *International Transactions in Operational Research*, 3(1), 1–21.
- Cook, S. (2006). The P versus NP problem. *The millennium prize problems*, 87–104.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*: MIT press.
- Dai, H., Dai, B., & Song, L. (2016). *Discriminative embeddings of latent variable models for structured data*. Paper presented at the International conference on machine learning.
- Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations Research*, 5(2), 266–288.
- Denzinger, J., Fuchs, M., & Fuchs, M. (1997). *High performance ATP systems by combining several AI methods*. Paper presented at the IJCAI (1).
- Domović, D., Rolich, T., & Golub, M. (2019). Evolutionary hyper-heuristic for solving the strip-packing problem. *The Journal of The Textile Institute*, 1–11.
- Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2), 405–428.
- Drake, J. H., Özcan, E., & Burke, E. K. (2016). A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. *Evolutionary Computation*, 24(1), 113–141.
- Duhart, B., Camarena, F., Ortiz-Bayliss, J. C., Amaya, I., & Terashima-Marín, H. (2018). *An experimental study on ant colony optimization hyper-heuristics for solving the knapsack problem*. Paper presented at the Mexican Conference on Pattern Recognition.
- Feng, Y., Yang, J., Wu, C., Lu, M., & Zhao, X.-J. (2018). Solving 0–1 knapsack problems by chaotic monarch butterfly optimization algorithm with Gaussian mutation. *Memetic Computing*, 10(2), 135–150.
- Fernando, B., Fromont, E., Muselet, D., & Sebban, M. (2012). *Discriminative feature fusion for image classification*. Paper presented at the 2012 IEEE conference on computer vision and pattern recognition.
- Gogna, A., & Tayal, A. (2013). Metaheuristics: Review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4), 503–526.
- Gómez-Herrera, F., Ramírez-Valenzuela, R. A., Ortiz-Bayliss, J. C., Amaya, I., & Terashima-Marín, H. (2017). *A quartile-based hyper-heuristic for solving the 0/1 knapsack problem*. Paper presented at the Mexican International Conference on Artificial Intelligence.
- Han, X., Iwama, K., Ye, D., & Zhang, G. (2016). Approximate strip packing: Revisited. *Information and Computation*, 249, 110–120.
- Han, X., Kawase, Y., & Makino, K. (2015). Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562, 395–405.
- Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... Silver, D. (2018). *Rainbow: Combining improvements in deep reinforcement learning*. Paper presented at the Thirty-second AAAI conference on artificial intelligence.
- Hopper, E., & Turtun, B. C. H. (2001). An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing Problem. *European Journal of Operational Research*, 128(1), 34–57.
- Hu, H., Zhang, X., Yan, X., Wang, L., & Xu, Y. (2017). Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*.
- Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., & Wassick, J. M. (2020). OR-Gym: A reinforcement learning library for operations research problem. *arXiv preprint arXiv:2008.06319*.
- Imahori, S., & Yagiura, M. (2010). The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. *Computers & Operations Research*, 37(2), 325–333.
- Iwama, K., & Zhang, G. (2010). Online knapsack with resource augmentation. *Information Processing Letters*, 110(22), 1016–1020.
- Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). *Learning combinatorial optimization algorithms over graphs*. Paper presented at the Advances in Neural Information Processing Systems.
- Kiran, R., Kumar, P., & Bhasker, B. (2020). DNNRec: A novel deep learning based hybrid recommender system. *Expert Systems with Applications*, 144, Article 113054.
- Kittler, J., Hatef, M., Duin, R. P., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis And Machine Intelligence*, 20(3), 226–239.
- Kong, W., Liaw, C., Mehta, A., & Sivakumar, D. (2019). *A new dog learns old tricks: RL finds classic optimization algorithms*. Paper presented at the International Conference on Learning Representations.
- Kool, W., van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.
- Lim, A. (1998). The berth planning problem. *Operations Research Letters*, 22(2–3), 105–110.
- Liu, T., Wang, Z., Tang, J., Yang, S., Huang, G. Y., & Liu, Z. (2019). *Recommender systems with heterogeneous side information*. Paper presented at the The world wide web conference.
- Mangai, U. G., Samanta, S., Das, S., & Chowdhury, P. R. (2010). A survey of decision fusion and feature fusion strategies for pattern classification. *IETE Technical Review*, 27(4), 293–307.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). *Asynchronous methods for deep reinforcement learning*. Paper presented at the International conference on machine learning.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Ostrovski, G. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

- Olivas, F., Amaya, I., Ortiz-Bayliss, J. C., Conant-Pablos, S. E., & Terashima-Marin, H. (2020, July). A Fuzzy Hyper-Heuristic Approach for the 0-1 Knapsack Problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8). IEEE.
- Oliveira, J. F., Neuenfeldt Júnior, A., Silva, E., & Carravilla, M. A. (2016). A survey on heuristics for the two-dimensional rectangular strip packing problem. *Pesquisa Operacional*, 36(2), 197–226.
- Oliveto, P. S. (2021). Rigorous performance analysis of hyper-heuristics. In *Automated design of machine learning and search algorithms* (pp. 45–71). Springer.
- Pardalos, P. M., Du, D.-Z., & Graham, R. L. (2013). *Handbook of combinatorial optimization*. Springer.
- Pillay, N., & Qu, R. (2018). *Hyper-Heuristics: Theory and Applications* (Vol. 10.1007/978-3-319-96514-7). Springer.
- Ryser-Welch, P., & Miller, J. F. (2014). A review of hyper-heuristic frameworks. Paper presented at the Proceedings of the evo20 workshop, aish.
- Sánchez-Díaz, X., Ortiz-Bayliss, J. C., Amaya, I., Cruz-Duarte, J. M., Conant-Pablos, S. E., & Terashima-Marin, H. (2021). A feature-independent hyper-heuristic approach for solving the knapsack problem. *Applied Sciences*, 11(21), 10209.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Bolton, A. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676), 354–359.
- Sun, B., Zeynali, A., Li, T., Hajiesmaili, M., Wierman, A., & Tsang, D. H. (2020). Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging. *Proceedings of the ACM on measurement and analysis of computing systems*, 4(3), 1–32.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Vaezi, F., Sadjadi, S. J., & Makui, A. (2019). A portfolio selection model based on the knapsack problem under uncertainty. *PLoS One*, 14(5), e0213652.
- Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11).
- Van Hasselt, H., Guez, A., & Silver, D. (2016). *Deep reinforcement learning with double q-learning*. Paper presented at the Thirtieth AAAI conference on artificial intelligence.
- Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.
- Verstichel, J., De Causmaecker, P., & Berghe, G. V. (2013). An improved best-fit heuristic for the orthogonal strip packing problem. *International Transactions in Operational Research*, 20(5), 711–730.
- Volodymyr, M., Kavukcuoglu, K., Silver, D., Graves, A., & Antonoglou, I. (2013). *Playing atari with deep reinforcement learning*. Paper presented at the NIPS Deep Learning Workshop.
- Wang, J., Liu, T., Liu, K., Kim, B., Xie, J., & Han, Z. (2018). *Computation offloading over fog and cloud using multi-dimensional multiple knapsack problem*. Paper presented at the 2018 IEEE Global Communications Conference (GLOBECOM).
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). *Dueling network architectures for deep reinforcement learning*. Paper presented at the International conference on machine learning.
- Wei, L., Hu, Q., Leung, S. C., & Zhang, N. (2017). An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation. *Computers & Operations Research*, 80, 113–127.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3), 229–256.
- Yi, W., Qu, R., & Jiao, L. (2023). Automated algorithm design using proximal policy optimisation with identified features. *Expert Systems with Applications*, 216, 119461.
- Zaidi, B. H., Bhatti, D. M. S., & Ullah, I. (2018). Combinatorial auctions for energy storage sharing amongst the households. *Journal of Energy Storage*, 19, 291–301.
- Zeiler, M. D., & Fergus, R. (2014). *Visualizing and understanding convolutional networks*. Paper presented at the European conference on computer vision.
- Zhang, Y., Bai, R., Qu, R., Tu, C., & Jin, J. (2021). A Deep Reinforcement Learning Based Hyper-heuristic for Combinatorial Optimisation with Uncertainties. *European Journal of Operational Research*.
- Zhang, Z., Zhang, X., Peng, C., Xue, X., & Sun, J. (2018). *Exfuse: Enhancing feature fusion for semantic segmentation*. Paper presented at the Proceedings of the European Conference on Computer Vision (ECCV).
- Zhao, Q., Zhang, Y., Friedman, D., & Tan, F. (2015). *E-commerce recommendation with personalized promotion*. Paper presented at the Proceedings of the 9th ACM Conference on Recommender Systems.
- Zhou, Y., Chakrabarty, D., & Lukose, R. (2008). *Budget constrained bidding in keyword auctions and online knapsack problems*. Paper presented at the Proceedings of the 17th international conference on world wide web.