Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Enhancing column generation by reinforcement learning-based hyper-heuristic for vehicle routing and scheduling problems

Kuan Xu [a], Li Shen [b], Lindong Liu [a],*

[a] International Institute of Finance, School of Management, University of Science and Technology of China, 230026, PR China
[b] JD EXPLORE ACADEMY, 100000, PR China

ARTICLE INFO

Keywords:
Column generation
Reinforcement learning
Hyper-heuristic
Vehicle routing problems
Bus Driver Scheduling Problems

ABSTRACT

Column generation (CG) is a crucial technique to solve large-scale problems by dynamically generating variables. It has extensive applications in common combinatorial optimization, such as vehicle routing and scheduling, where each iteration involves solving a constrained shortest path problem, which is NP-hard. While heuristic acceleration methods exist, they often lack versatility across different problem domains. This study proposes RLHH, a reinforcement learning-based hyper-heuristic framework that enhances CG by integrating a selection module to accelerate convergence and improve integer solutions. At each CG iteration, the RL agent selects a low-level heuristic to construct a reduced network, retaining only the most promising edges. We apply RLHH to two combinatorial optimization problems – Vehicle Routing Problem with Time Windows (VRPTW) and Bus Driver Scheduling Problem (BDSP) – achieving up to 27.9% and 15.4% cost reductions, respectively, compared to the best standalone heuristic, with similar or lower computational costs. Furthermore, we demonstrate successful transfer learning from VRPTW to BDSP through pretraining and fine-tuning, reducing training costs significantly while maintaining performance. RLHH is the first RL-based CG method that outperforms traditional approaches in terms of solution quality, underscoring its potential to enhance CG-based optimization in various domains.

1. Introduction

Column generation (CG), also known as branch-and-price when integrated into a branch-and-bound framework, is initially applied to cutting stock problems and has significantly influenced the field of OR (Gilmore & Gomory, 1961, 1963). It solves problems with exponential large solution space by dynamically generating variables and has demonstrated remarkable effectiveness in tackling a diverse range of combinatorial optimization (CO) problems (Desaulniers et al., 2006). It decomposes the primary problem into two parts: a restricted master problem (RMP) that is consistent with the original linear program but limited to a subset of variables, and one or more pricing problems (PP) to generate new columns (representing variables) that enhance the current RMP solution. In the majority of vehicle routing or crew scheduling applications, the problem is typically formulated on a network. The master problem is a set partitioning or set covering problem. In this context, the variables are linked to vehicle routes or crew schedules. These variables, denoted as columns in CG, are generated by PP that corresponds to a shortest path problem with resource constraints (SPPRC) or elementary SPPRC (ESPPRC). An elementary path is a path in which all nodes are pairwise distinct.

It is worth noting that the SPPRC is commonly known to be NP-hard (Dror, 1994) and needs to be solved in every iteration of CG. According to Václavík et al. (2018), solving PP consumes more than 90% of the CPU time in the CG process, making it nearly unfeasible to solve vehicle routing and crew scheduling problems as their scale expands. Researchers have employed a variety of heuristic methods to speed up the PP by reducing the size of the pricing network (Alberto Santini et al., 2018; Desaulniers et al., 2008; Mauro Dell'Amico et al., 2006). And Morabit et al. (2023) propose a ML-based pricing heuristic and demonstrate its capacity to expedite the convergence of the CG process. However, their ML model does not perform as well as a straightforward heuristic algorithm in VRPTW experiments. So how to leverage the advantages of diverse heuristic methods warrants further investigation.

In this study, we introduce RLHH, a hyper-heuristic approach based

* Corresponding author.
E-mail addresses: mathkxu@mail.ustc.edu.cn (K. Xu), mathshenli@gmail.com (L. Shen), ldliu@ustc.edu.cn (L. Liu).

https://doi.org/10.1016/j.cie.2025.111138
Received 22 October 2023; Received in revised form 27 February 2025; Accepted 13 April 2025
Available online 9 May 2025

on deep reinforcement learning, integrated into the iterative framework of CG. Our method combines the powerful information extraction and generalization capabilities of neural networks with the theoretical advantages of the CG algorithm. Specifically, the approach utilizes the solution information of RMP and features of pricing networks in PP to determine the suitable low-level heuristic algorithm. This results in the construction of a reduced network, including edges with a higher likelihood of contributing to the optimal solution. The reduced network is employed at each CG iteration if it generates a satisfactory column. Otherwise, we resort to the complete network, especially in the final iteration. In the RL section, we use a Double Deep Q Network algorithm (DDQN) with experience replay (Mnih et al., 2015) to train a Q-function approximator which determines the optimal decision for the agent in each state.

We evaluate RLHH on two well-known CO problems: the vehicle routing problem with time windows (VRPTW) (Desrochers et al., 1992; Kohl et al., 1999) and the bus driver scheduling problem (BDSP) (Perumal et al., 2022; Smith & Wren, 1988), both characterized by distinct network structures. Still, RLHH enhances CG performance by pruning pricing networks, rendering the approach outlined in this paper applicable to a variety of problems. In the VRPTW experiments, RLHH achieves a 27.9% reduction in total cost within 75% of the computation time compared to the best low-level heuristic method. Similarly, in BDSP, the reductions in the total cost of up to 15.4% can be obtained within a comparable computational time. Additionally, we explore the potential of transfer learning by pretraining the RL model on VRPTW and fine-tuning it on BDSP, reducing training time by an order of magnitude while preserving solution quality.

In the end, we summarize our main contributions as below:

- We propose RLHH, a reinforcement learning-based hyper-heuristic framework that enhances column generation (CG) for combinatorial optimization. It shows the great potential of combining the stability and efficiency of OR algorithms with learning and generalization capabilities of RL models.
- We employ RLHH to solve two classic problems, VRPTW and BDSP, incorporating final solution quality into the reward function. Experimental results show that RLHH outperforms individual heuristics while maintaining or reducing computational time.
- Extensive experiments demonstrate RLHH's strong generalization across larger and more complex problem instances. We investigate the transferability of RLHH by pretraining on VRPTW and adapting the model to BDSP, achieving significant training efficiency gains with minimal performance loss.

2. Literature review

Machine Learning (ML) applied to combinatorial optimization has recently received much attention (Bengio et al., 2021; Lombardi & Milano, 2018). Yan et al. (2020) describe how learning techniques and paradigms are applied to CO. Routing problems, notably TSP and VRP, have been extensively investigated in a series of studies (Khalil, Dai et al., 2017; Kool et al., 2019; Vinyals et al., 2015). The majority of these studies adopt an end-to-end approach, directly constructing solutions using ML models.

We focus on a literature series that exploits ML to assist optimization problem-solving. Numerous studies apply ML techniques to branch-and-bound solvers, such as learning variable selection method (Gasse et al., 2019; Gupta et al., 2022; Khalil et al., 2016) and learning node selection (Furian et al., 2021; Liu et al., 2024). Khali, Dilkina et al. (2017) use ML models to dynamically decide whether to run a heuristic at each node of the search tree. Additionally, many researchers use ML to assist or control the local search (Hottung & Tierney, 2022; Ma et al., 2022, 2021), among which Lu et al. (2019) present the first RL framework that outperforms classical Operations Research (OR) solvers LKH3 on

Table 1
Notations used in the mathematical model.

Notation	Description
$\mathcal{P}$	Set of variable indices, representing feasible routes or schedules
$c_p \in \mathbb{R}$	Variable cost associated with index p
$\boldsymbol{a_p} \in \mathbb{R}^m$	Coefficients of the constraints associated with index p
$\boldsymbol{b} \in \mathbb{R}^m$	Constraints right-hand side vector
π	Dual values of RMP solution
V	Set of all nodes, including source node s and destination node t
$G = (V, E)$	Pricing network with nodes V and edges E
R	Set of all resources
T_i^r	Value of resource r accumulated on a partial path from node s to i
$[\underline{w}_i^r, \overline{w}_i^r]$	Resource window of T_i^r
c_{ij}	Cost associated with edge (i, j)
$t_{ij}^r \in \mathbb{R}$	Consumption of resource r on edge (i, j)

capacitated VRP. And Jacobs et al. (2021) solve robust optimization utilizing an RL-based heuristic method, Zhang et al. (2022) present an RL-based hyper-heuristic for CO problems with uncertains.

CG is an indispensable tool in CO for solving large-scale integer programming problems. Morabit et al. (2021) exploit supervised learning to select promising columns at each iteration within the CG procedure. The experiments show that their approach accelerates the CG algorithm by 30%. However, it requires labeled data generated by an extremely time-consuming mixed-integer linear program. In contrast, Chi et al. (2022) utilize reinforcement learning that circumvents this issue. Their proposed selection module converged faster than the greedy column selection strategy. Babaki et al. (2022) propose a neural CG architecture to select columns at each iteration by imitation learning. Both Chi et al. (2022) and Morabit et al. (2021) use ML for column selection to speed up the CG procedure, while our method greatly accelerates the pricing problem, which usually consumes most of the computing time. Additionally, Sebastian Kraul et al. (2023) propose a prediction model for optimal dual variables in the cutting stock problem, enabling automated parameter selection for stabilized column generation and eliminating the need for numerical tuning.

The work most closely related to our proposed RLHH is (Morabit et al., 2023), which proposes a new heuristic pricing algorithm based on ML to prune the pricing network and accelerate the PP. Similarly, Yuan et al. (2022) present a pricing heuristic for railway crew scheduling problems based on graph neural networks. They utilize supervised learning, which requires a large amount of labeled training data to be pre-obtained through the OR solver. However, it is worth noting that the trained ML model does not perform as well as a heuristic method in VRPTW experiments (Morabit et al., 2023). And Qin et al. (2021) develops a hyper-heuristic based on RL to solve the heterogeneous vehicle routing problem, which outperforms the existing meta-heuristic algorithms.

In summary, due to the pivotal role of CG and the flourishing development of ML techniques, a diverse range of heuristic and ML-based approaches have been developed to expedite the RMP or PP. However, there remains substantial room for improvement in the quality of solutions. Therefore, this study aims to investigate a RL-based hyper-heuristic algorithm, leveraging the efficiency of heuristic methods and the learning and generalization capabilities of RL to enhance the performance of CG.

3. Preliminaries

This section initially introduces the fundamental concepts of column generation along with the essential notation (see Table 1). Subsequently, we delve into the pricing problem that is the primary focus of this article, namely Shortest Path Problem with Resource Constraints (SPPRC). Finally, we provide a concise summary of RL terminology to facilitate comprehension of the RLHH method proposed in this paper.

3.1. Column generation

Let us focus on the linear relaxation of the original integer programming, considering the following master problem (MP):

$$z^*_{MP} := \min_x \sum_{p\in\mathcal{P}} c_p x_p \tag{1}$$

$$\text{s.t.} \sum_{p\in\mathcal{P}} \boldsymbol{a}_p x_p = \boldsymbol{b}, \tag{2}$$

$$x_p \geq 0, \quad \forall p \in \mathcal{P}. \tag{3}$$

When $|\mathcal{P}|$ is prohibitively large so that the variables (columns) cannot be enumerated explicitly, we consider a restricted master problem (RMP) that only contains a subset $\mathcal{P}' \subseteq \mathcal{P}$.

We optimize RMP at each CG iteration and get the optimal dual solution π with respect to constraints (2). Then, the pricing problem (PP) is established using the dual values π to search for new columns with the least reduced cost:

$$\min_{p\in\mathcal{P}} \quad c_p - \pi^T \boldsymbol{a}_p, \tag{4}$$

which is an ESPPRC in the VRPTW case or SPPRC in the BDSP case. If the new column has a negative reduced cost, it is added to the RMP to improve its solution further. Otherwise, RMP has captured all the necessary columns, which means the optimal solution of MP is obtained. Finally, by limiting the variables of RMP to integers and solving the integer RMP (IRMP), we obtain a integer solution as the result of the experiment in this paper.

3.2. SPPRC formulation

The standard SPPRC frequently appears in problems defined on acyclic networks, such as vehicle and crew scheduling problems of which BDSP is a special case (Haase et al., 2001) and aircrew rostering problem (Gamache et al., 1999). A notable variant is the elementary SPPRC (ESPPRC), which arises in vehicle routing problems (Costa et al., 2019). An elementary path, where all nodes are pairwise different, poses greater computational challenges.

The PP aims to find the optimal path on network G that satisfies the resource constraint. It can be formulated as the following SPPRC:

$$\min \sum_{(i,j)\in E} c_{ij} X_{ij} \tag{5}$$

$$\text{s.t.} \sum_{j\in V} X_{sj} - \sum_{j\in V} X_{js} = 1, \tag{6}$$

$$\sum_{j\in V} X_{ij} - \sum_{j\in V} X_{ji} = 0, \quad \forall i \in V\backslash\{s,t\}, \tag{7}$$

$$\sum_{j\in V} X_{tj} - \sum_{j\in V} X_{jt} = -1, \tag{8}$$

$$X_{ij}(T_i^r + t_{ij}^r) - T_j^r \leq 0, \quad \forall (i,j) \in E, \tag{9}$$

$$\underline{w}_i^r \leq T_i^r \leq \overline{w}_i^r, \quad \forall i \in V, \forall r \in R, \tag{10}$$

$$X_{ij} \in \{0,1\}, \quad \forall (i,j) \in E. \tag{11}$$

The objective function (5) serves to minimize the overall path cost. Constraint set (6)–(8) implies that the flow along the path is conserved. Constraint set (9) indicates the resource consumption for each resource $r \in R$ on edge $(i,j) \in E$. Constraint set (10) confines the accumulation of resource values within their respective intervals at each node. The last constraint set (11) defines the decision variables X_{ij}.

In every CG iteration, we extract the dual values π from RMP and employ them to identify new paths within a pricing network. Consider a new path $p \in \mathcal{P}$ covering a set of edges denoted as E_p and servicing a set of customers denoted as V_p. Then if we add a path p to $\mathcal{P}'$, the amount of change that will bring to the total cost (i.e., the reduced cost) is:

$$\Delta_p = \sum_{(i,j)\in E_p} c_{ij} - \sum_{v\in V_p} \pi_v. \tag{12}$$

We define a binary parameter ρ_{ij}^k, which equals 1 if edge $(i,j) \in E$ contributes to the kth constraint in (2), 0 otherwise. Then the reduced cost Δ_p can be rewritten as follows:

$$\Delta_p = \sum_{(i,j)\in E_p} c_{ij} - \sum_{(i,j)\in E_p} \sum_{k\in V_p} \rho_{ij}^k \pi_k = \sum_{(i,j)\in E_p} \left(c_{ij} - \sum_{k\in V_p} \rho_{ij}^k \pi_k \right). \tag{13}$$

For each edge $(i,j) \in E$, we define the modified cost on it as $\bar{c}_{ij} = c_{ij} - \sum_{k\in V} \rho_{ij}^k \pi_k$.

Therefore, solving problem (5)–(11) with the modified costs in (5) results in a column with the minimum reduced cost. Dynamic programming-based methods are commonly employed to solve the SPPRC and ESPPRC. In our experiments, we utilize a labeling algorithm proposed in Tilk et al. (2017). It is noteworthy that our RLHH framework is applicable regardless of the specific algorithm used for PP.

Notably, the number of edges in the pricing network $|E|$ is proportional to n^2, where n is the number of customers in VRPTW or trips in BDSP. In a complete CG process, it may be necessary to solve PP hundreds of times. It is highly non-trivial to derive a suitable reduced network to solve PP more efficiently. Without acceleration techniques, it is almost impossible to obtain quality solutions in an acceptable computation time (Desaulniers et al., 2006).

3.3. RL terminology

Reinforcement learning is a branch of machine learning dedicated to the exploration of sequential decision-making tasks. The paradigm portrays the learning process as an agent-environment interaction across discrete time steps. At time step t, the agent observes the representation of the current environment state $S_t \in S$, selects an action $A_t \in A(S_t)$, and receives a scalar reward $R_{t+1} \in R$ from the environment. The goal of the agent is to learn a policy that chooses the appropriate action based on the present state to maximize the expected cumulative reward over time. A transition is denoted by (s, a, r, s'), where s, a and r are the state, action and reward of the current step respectively, and s' is the next state after the agent executes the action a. We use an experience replay buffer M to store the transition data generated during the interaction. RL encompasses exploration strategies to balance the trade-off between gathering information about the environment (exploration) and exploiting known knowledge to maximize rewards (exploitation).

4. Methodology

Our goal is to attain a reduced SPPRC network by employing the best heuristic method to retain the most promising edges. This significantly improves the efficiency in finding new columns of the CG iteration process. This constitutes a sequential decision task that can be modeled as a Markov decision process and addressed through RL algorithms.

4.1. Low-level heuristics

Due to the difficulty of the SPPRC, numerous heuristic methods have been proposed in the literature. Desaulniers et al. (2008) relax the dominance rule by only dominating on a subset of the resources when solving with a labeling algorithm. And several strategies revolve around reducing the size of pricing networks, which constitute the primary focus of this article. Subsequently, we provide a detailed description of the five low-level algorithms employed in this study.

BestEdges1 (BE1). Mauro Dell'Amico et al. (2006) utilize a CG-based approach to solve a variation of the capacitated vehicle routing problem. They remove edges that satisfy:

$$c_{ij} > \alpha_1 \pi_{max}, \text{ where } \pi_{max} = \max_k \{\pi_k\}, \tag{14}$$

where $\alpha_1 \in (0,1)$ is a parameter and π is the dual values. This approach may be more suitable for the relatively significant cost difference between edges. And the parameter α_1 is to find a trade-off between the quality of the solution and the speed of solving the PP.

Table 2
The parameter values we used in the experiment.

Name	Parameter	Values
BE1	α_1	[0.1, 0.3, 0.5, 0.7]
BE2	α_2	[0.1, 0.2, 0.3]
BE3	N	[0.3, 0.5, 0.7] $*\,\lvert V\rvert$
BN	β	[0.9, 0.7, 0.3]
BP	K	[3, 5, 7, 9]

BestEdges2 (BE2). Alberto Santini et al. (2018) solve the problem of designing a container liner shipping feeder network by a CG algorithm. In the PP, they sort the edges by their reduced costs $\bar{c}_{ij}$ and only keep the $\alpha_2|E|$ ones with the lowest value, where $\alpha_2 \in (0,1)$ is a parameter (ratio). Different from BE1, this method takes into account the modified edge cost, which contains both the initial cost and the dual information of the RMP. So in a sense, BE2 incorporates more information than BE1. Additionally, our experiments have demonstrated that the performance of BE2 is notably superior (refer to Section 5 for details).

BestEdges3 (BE3). Desaulniers et al. (2008) exploit a CG-based method for the VRPTW. To expedite the resolution of the PP, they sort the incoming and outgoing edges of each customer node based on their reduced costs $\bar{c}_{ij}$, and only N in-edges and N out-edges with the lowest reduced costs are retained. The primary distinction between this approach and the preceding two lies in its consideration of all customer nodes, guaranteeing that each customer possesses the potential to be served.

BestNodes (BN). In addition to BE2, Alberto Santini et al. (2018) suggest another way to obtain a reduced network based on the dual values π. Firstly, normalize π into the interval $[0,\beta]$ according to the formula (15), where β is a parameter.

$$\pi'_k = \beta \cdot \frac{\pi_k - \pi_{min}}{\pi_{max} - \pi_{min}}, \quad \beta \in (0,1). \tag{15}$$

Let $\bar{i}$ and $\underline{i}$ represent the customer corresponding to π_{max} and π_{min} respectively. Then edge (i,j) is removed with a probability directly proportional to the dual value of its target node. Edges incoming to $\underline{i}$ have a probability of 0, while edges incoming to $\bar{i}$ having a probability of β. This sparsification method only considers the dual values from the RMP, irrespective of the original edge costs.

BestPaths (BP). There is a heuristic algorithm based on the shortest path problem without negative cost that is mentioned in Desaulniers et al. (2006). First, they normalize $\bar{c}$ into the interval $[-1,1]$, and then reset all negative ones to 0, as shown in (16).

$$\bar{c}'_{ij} = \max\left(0, \frac{(\bar{c}_{ij} - \bar{c}_{min}) - (\bar{c}_{max} - \bar{c}_{ij})}{\bar{c}_{max} - \bar{c}_{min}}\right), \tag{16}$$

where $\bar{c}_{max} = \max_{(i,j)\in E}\{\bar{c}_{ij}\}$ and $\bar{c}_{min} = \min_{(i,j)\in E}\{\bar{c}_{ij}\}$ represent the maximum and minimum modified edge costs in the pricing network respectively. Finally, calculate the K shortest paths (where K is a parameter) on the normalized network and obtain a reduced network exclusively comprising the edges belonging to these paths. Evidently, this method is more time-consuming than the other four methods, as illustrated in Fig. 3.

Finally, it is crucial to note that the parameters in these low-level heuristics evolve over time. In order to simultaneously address solution quality and speed, we try different values of parameters successively. The parameter values for these heuristics are adopted directly from the corresponding literature without significant adjustment. Our focus is not on fine-tuning individual heuristics but rather on effectively integrating and utilizing them within our framework. As shown in Table 2, these parameter values are used consistently in all experiments for VRPTW and BDSP. If all parameter values are tried and no column with a negative reduced cost is found, the heuristic is deemed unsuccessful, and we use the complete network.

4.2. RLHH framework

The RL-based hyper-heuristic (RLHH) approach is integrated within the CG framework (see Fig. 1). We know that a hyper-heuristic combines the efficacy of problem-specific heuristics. In addition, RLHH leverages the learning and generalization capabilities of neural networks. In the RL context, the CG process serves as the environment, while five low-level heuristics constitute our action space. These heuristics construct reduced networks according to their prescribed rules, resulting in computationally efficiency. The state includes solution information of RMP and features of the pricing network in PP. Given the current state as input, a neural network generates the action score vector. The weights of the neural network are trained by the DDQN algorithm with experience replay (Mnih et al., 2015). As shown in Section 5, whether dealing with VRPTW or BDSP, our RL model can distinguish the efficient low-level heuristic at each step to generate new valid columns that ultimately leads to excellent integer solutions.

State space. The state space represents the environment's current status and provides essential information for the agent to make decisions. The state S_t at time step t encapsulates features that describe the current state of the problem, allowing the agent to assess the situation and choose an appropriate action. In our RLHH framework, each state includes information extracted from the current relaxation solution of the RMP and the individual pricing network. This information reflects the progress of the column generation process and serves as a critical input to the RL agent.

The state space is carefully designed to capture key aspects of the underlying network structure and solution quality. It provides a dynamic representation that may exhibit significant differences at various stages of the iterative process or across distinct problem structures due to the differences in the underlying network topology. Specifically, the features used for VRPTW and BDSP considered in this paper are as follows:

- Objective Ratio: Current objective value/initial objective value
- Average Fractional Value: Sum of fractional variables/number of fractional variables
- Fractional Variable Ratio: Number of fractional variables/total number of variables
- Dual and Cost Entropy: Information entropy for dual values, cost and reduced cost of edges
- Resource Consumption Dispersion: Coefficient of variation (COV) of the resource consumption along edges for each resource

The first feature represents the ratio between the current optimal objective value and the initial objective value of the RMP, reflecting the progress made during iterations. To describe the fractional solutions in the RMP, we define two additional features. For example, if the current iteration solution contains five variables with values $[0.2, 1, 0, 0.75, 0.1]$, the average fractional value is computed as $\frac{0.2+0.75+0.1}{3} = 0.35$, while the fractional variable ratio is $\frac{3}{5} = 0.6$. Another feature is the entropy of dual values, cost values, and reduced cost values, calculated as $H = -\sum_{i=1}^{n} p_i \log p_i$, where p_i is the normalized value of each component, providing a measure of diversity. Finally, the dispersion in resource consumption along edges is captured using the coefficient of variation, defined as the ratio of the standard deviation to the absolute value of the mean.

These features are inherently scale-invariant, meaning they remain meaningful regardless of the problem size. By normalizing key metrics such as objective values, fractional solution characteristics, entropy, and resource variation, the learned agent is not restricted to a fixed problem scale. This allows the trained RL model to be naturally applicable across instances of varying sizes without requiring additional retraining.

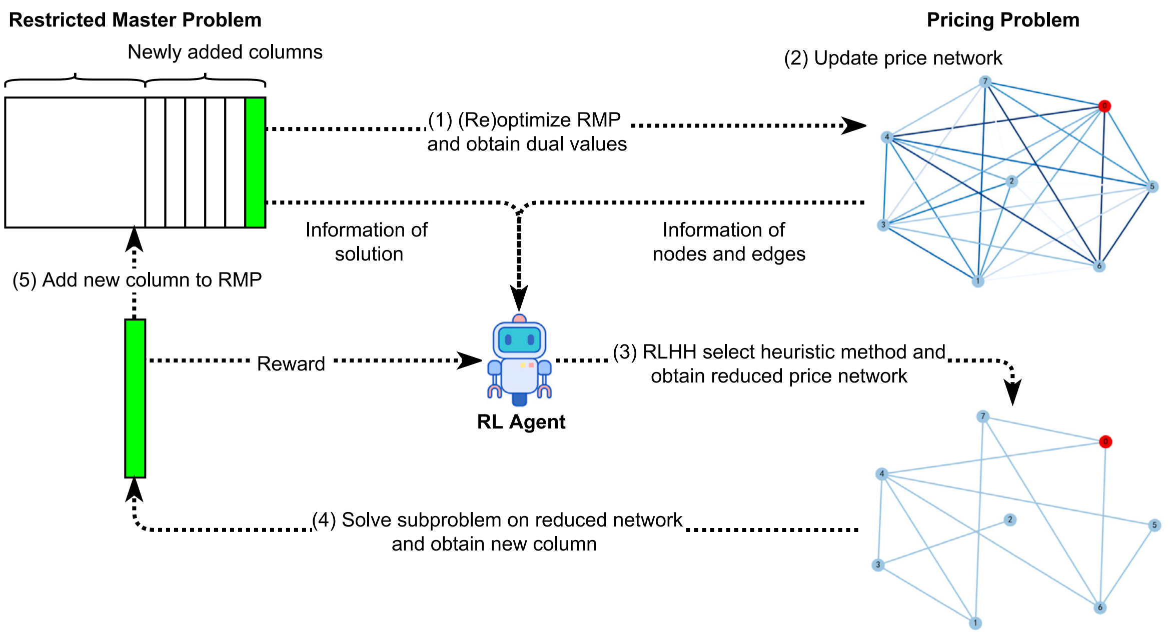
Restricted Master Problem
Newly added columns
Pricing Problem
(2) Update price network
(1) (Re)optimize RMP and obtain dual values
Information of solution
Information of nodes and edges
(5) Add new column to RMP
Reward
RL Agent
(3) RLHH select heuristic method and obtain reduced price network
(4) Solve subproblem on reduced network and obtain new column

Fig. 1. Our RL-based hyper-heuristic approach in column generation.

Action space. In RL, the action space represents the set of possible decisions the agent can take at each step, which directly influences the state transition and overall learning process. In this study, each action corresponds to the selection of a low-level heuristic for constructing a reduced pricing network. The agent selects an action $a_t \in \{\text{BE1}, \text{BE2}, \text{BE3}, \text{BN}, \text{BP}\}$ at each iteration, and the corresponding heuristic is applied to compute a reduced network. This reduced network helps accelerate the pricing problem by retaining only promising edges that are more likely to be part of the optimal solution.

It is important to note that the parameters in these heuristic methods are time-evolving, allowing the heuristics to adapt as the column generation process progresses. And the parameter values for all heuristics are adopted directly from the corresponding literature without significant adjustment. Taking BE1 as an example, the algorithm initially identifies numerous columns with negative reduced costs when the parameter α_1 is set to 0.1. However, maintaining $\alpha_1 = 0.1$ for subsequent iterations leads to a failure in finding such columns. To address this, we sequentially increase the value of α in subsequent trials. If no column with a negative cost reduction is identified when α_1 reaches 0.7, the heuristic is considered unsuccessful, and we fall back to using the complete network. This adaptive strategy ensures that the agent can explore different configurations of the reduced network and improve the likelihood of identifying high-quality solutions.

Reward function. The reward function serves as a critical feedback mechanism in RL that guides the agent's learning process. It quantifies the immediate benefit of an action taken in a specific state, helping the agent distinguish between effective and suboptimal decisions. The ultimate goal of the agent is to maximize the cumulative reward over time by learning a policy that selects actions leading to high rewards.

In previous work, Chi et al. (2022) use an advantage-based reward function $(\text{obj}_{t-1} - \text{obj}_t)/\text{obj}_0$, where obj_t is the objective value of the RMP_t. It focuses on the quality of the solution improvement at each iteration, i.e., the change in the RMP objective value. We note that actions frequently achieve significant cost reductions in early iterations, whereas achieving the same becomes increasingly difficult in later steps. In particular, the potential for objective value reduction, and the magnitude of that reduction, decreases with each iteration. Consequently, it seems unfair to give larger rewards for early actions.

In the CG framework, we encourage the RL agent to choose the action that can generate a reduced network containing negative reduced cost columns. So we set the step reward r_t as follows:

$$r_t = \begin{cases} 1, & \text{if a new column is found and improves the RMP solution,} \\ 0, & \text{if a new column is found but does not improve the RMP solution,} \\ -1, & \text{if no column is found in the reduced network.} \end{cases}$$

Furthermore, our primary aim is to attain an optimal solution. Therefore, at the end of an episode, we use the reverse integrality gap (RIG) to evaluate the integer solution's quality:

$$r_T = 100^{\text{RIG}}, \text{where RIG} = \frac{\text{obj}_{\text{frac}}}{\text{obj}_{\text{int}}},$$

where obj_{int} and obj_{frac} denote the objective value of the integer solution and the relaxed solution, respectively. The reverse integrality gap ensures that higher rewards are assigned to integer solutions that are closer to the relaxed solution in terms of objective value. For instance, when RIG = 1, indicating that the integer solution equals the relaxed solution, the agent receives a terminal reward of 100. Compared to linear and exponential functions, the exponential function is more sensitive to high-quality solutions, effectively encouraging the agent to converge toward an optimal solution. The rationale behind the function and parameter settings is detailed in Appendix D.

This design serves a dual purpose. The step reward encourages the agent to iteratively identify columns that can improve the RMP solution. Meanwhile, the exponential terminal reward $r_T = 100^{\text{RIG}}$ supports the agent in exploring columns that might not offer immediate improvements, as long as the final solution is of high quality. This is particularly beneficial when RIG approaches 1, as the agent can receive a sufficiently large reward for achieving an optimal or near-optimal solution.

Training and execution. The RL agent is defined by a value function $Q(s, a)$, linked to a given state s and action a. This function serves as the core of the low-level heuristic selection module within the CG framework. The agent selects an action according to the state at each iteration. After that, it receives a reward value from the PP solution. The Q function, represented by (17), embodies the anticipated

Algorithm 1 RLHH: RL-based hyper-heuristic in Column Generation

```
Input: Problem instance & trained Q-function
Output: Integer solution of the original CO problem
 1: Let $t = 0$; Build the $\text{RMP}_0$ with initial columns.
 2: Solve $\text{RMP}_0$; Use dual values to get the complete pricing network.
 3: Calculate initial state $s_0$.
 4: while exist new columns with negative reduced cost do
 5:     $a_t^* = \arg\max_{a_t \in A_t} Q(s_t, a_t)$;
        Utilize selected low-level heuristic to get a reduced network.
 6:     Attempt to find a new negative reduced cost column in the reduced network.
 7:     if not found then
 8:         Find a new negative reduced cost column in the complete network.
 9:     end if
10:     Add new column to $\text{RMP}_t$ and solve $\text{RMP}_{t+1}$;
11:     Solve $\text{RMP}_t$; Use dual values to get the complete pricing network.
12:     Calculate next state $s_{t+1}$.
13:     Set $t = t + 1$
14: end while
15: Set the variable to integer type and solve.
16: return Final IRMP solution
```

cumulative rewards after discounting, with γ denoting the discounted factor and R signifying the reward. The details of the training process are presented in Algorithm 2 in Appendix A.

$$Q(s,a) = E[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, A_t = a]. \tag{17}$$

Algorithm 1 demonstrates the implementation of a trained RLHH agent within CG to solve CO problems. The Steps 1–2 incorporate an initial set of columns, for example, in the case of VRPTW, where each customer is initially assigned a separate vehicle. These columns are introduced into the basis, and the first pricing network is construct based on dual values. In the while loop, the RL agent selects the optimal action from the candidate low-level heuristic methods and gets a reduced network. Then we attempt to find a new column in the reduced network with a negative reduced cost, which could theoretically reduce the value of the objective function. If that fails, we proceed to Step 8 to search for such a new column in the complete network. The RMP is updated in the same way as the traditional CG method. Finally, the variable is converted to integer type, and the original integer programming problem is solved.

5. Experiments

The proposed framework is versatile and applicable to a range of routing and scheduling problems. In this study, we choose two classic cases, namely VRPTW and BDSP, for experimentation. In recent years, VRPTW is increasingly becoming an invaluable tool for modeling all aspects of supply chain design and operation. BDSP constitutes a significant challenge in urban mass transit systems. With the adoption of suitable modeling techniques, our method can be directly applied to other scheduling problems, including railway crew scheduling problems. While routing problems (e.g., VRPTW) and scheduling problems (e.g., BDSP) exhibit distinct network structures, our approach enhances CG performance by selecting promising edges. This renders the RLHH described in this paper sufficiently versatile for application to diverse problem domains.

We follow Morabit et al. (2021) by focusing on solving the root node of the branch-and-price (B&P) search tree. However, it is worth noting that the proposed RLHH strategy is equally applicable to other nodes within the tree. We compare the five low-level heuristics described in Section 4.2 and report two metrics. to show that our method can obtain superior integer solutions, i.e. upper bounds in B&P, in less time. In the course of the B&P process, a node can be safely pruned without further expansion if its lower bound is no more favorable than the global upper bound. This results in a remarkable reduction in the search time required to attain the optimal solution.

We use a single RTX 3060 GPU for RL training, implement with Python 3.7 and PyTorch 1.10.0. And we use Gurobi 9.1.2 to solve the final IRMP. All experiments implemented on a personal computer with an i7-11700 CPU at 2.5 GHz and 16 GB RAM. A full training session consists of 10,000 episodes, taking approximately 50 h for VRPTW and 80 h for BDSP.

5.1. Vehicle routing problem with time windows

VRPTW is an extension of the capacitated vehicle routing problems where vehicles are required to commence service within the customer's designated time window. It aims to find feasible routes, each requires a dedicated vehicle, for delivering goods to a geographically scattered customer base while minimizing the total travel cost. A feasible route originates from and returns to the depot, while adhering to capacity and time window constraints. The most effective approaches for solving this problem are founded on CG techniques. The problem can be formulated as a set covering problem whose linear relaxation is (MP) (1)–(3), where each column corresponds to a feasible route (refer to Appendix B for details).

Datasets. For training and testing data, we employ the well-known Solomon benchmark (Solomon, 1987). The dataset contains six distinct problem types (i.e., C1, C2, R1, R2, RC1, RC2), each comprising 8–12 instances with 100 customers. "C" refers to geographically clustered customers, "R" denotes randomly situated customers, and "RC" signifies a combination thereof. Categories "1" and "2" correspond to scenarios with either narrow time windows and small vehicle capacity, or broad time windows and large vehicle capacity, respectively. These categories reflect different problem structures, where instances from category "2" have a greater level of complexity. From each original Solomon instance, we can generate smaller instances by considering only the first $n < 100$ customers.

We use small-size instances from types C1, R1 and RC1 for training. On each episode, we choose one of all instances of the three types with equal probability and consider the first n customers, where n is uniformly sampled from 25 to 35. To test the generalization ability of our model, we consider two sets of instances: (1) small-size ($25 \le n \le 35$) instances from types C2, R2 and RC2; (2) large-size ($n = 50, 75, 100$) instances from all six types.

Training process. We select a fixed problem instance to assess the RL agent's performance during the training process. In other words, in each episode, the RL agent solves the identical VRPTW instance. We know that CG is an exact method for solving LP, irrespective of the heuristic employed, the algorithm will eventually converge to the same relaxation objective value. Nonetheless, since different heuristics yield different feasible routes at each CG iteration, the final integer solution may vary. At the end of each episode, we calculate the integer solution

Table 3
VRPTW: The objective values of integer solution for all small-size R2 instances (in parentheses is the solving time in seconds). The last column represents the ranking of RLHH among all the methods.

No.	Instance	n	BE1	BE2	BE3	BN	BP	**RLHH(Ours)**	Rank
1	r203	25	669.77 (12)	669.04 (58)	768.30 (67)	689.00 (79)	714.12 (256)	**520.25** (46)	1
2	r202	30	**440.98** (13)	522.45 (65)	509.47 (14)	516.33 (45)	639.64 (62)	480.86 (28)	2
3	r201	27	520.28 (5)	506.05 (5)	506.71 (7)	506.05 (8)	507.69 (11)	**504.17** (7)	1
4	r209	31	**715.36** (12)	788.04 (22)	858.35 (81)	820.94 (32)	1160.49 (132)	819.72 (7)	3
5	r208	28	1291.17 (600)	**1113.49** (540)	1291.17 (590)	1291.17 (600)	1232.41 (600)	1200.78 (4)	2
6	r204	32	1504.75 (600)	1221.90 (31)	1504.75 (600)	1504.75 (600)	1408.62 (600)	**1139.31** (8)	1
7	r202	25	559.35 (6)	499.75 (11)	550.35 (12)	543.28 (22)	552.21 (29)	**496.58** (13)	1
8	r211	34	**1151.35** (161)	1210.89 (70)	1626.38 (600)	1352.74 (326)	1488.33 (600)	1467.85 (7)	3
9	r209	25	821.21 (5)	739.32 (16)	1121.88 (47)	917.29 (12)	868.09 (63)	**643.02** (7)	1
10	r202	27	490.71 (9)	**463.77** (22)	577.41 (9)	549.26 (34)	684.89 (53)	533.66 (14)	3
11	r210	28	786.53 (11)	660.88 (34)	903.06 (17)	847.16 (40)	775.87 (75)	**657.92** (5)	1
12	r211	26	1018.51 (92)	1085.38 (158)	1268.52 (600)	1253.23 (129)	1148.47 (242)	**1007.23** (2)	1
13	r202	28	534.73 (10)	543.75 (46)	537.56 (13)	468.72 (53)	526.55 (53)	**456.28** (30)	1
14	r209	26	**627.79** (7)	766.72 (10)	723.77 (50)	760.53 (14)	747.84 (76)	704.66 (9)	2
15	r201	32	583.40 (7)	545.18 (12)	573.84 (8)	**545.12** (15)	564.58 (23)	561.28 (11)	3
16	r203	29	876.79 (34)	817.75 (43)	792.64 (64)	876.15 (42)	1267.10 (78)	**608.75** (45)	1
17	r205	33	912.83 (342)	**685.97** (199)	1045.75 (239)	957.57 (440)	1055.67 (289)	839.28 (59)	2

Table 4
VRPTW: The average objective values for large-size instances (in parentheses is the solving time in minutes). Speedup is the ratio between the solving time of the best baseline and our method.

Type	n	BE1	BE2	BE3	BN	BP	**RLHH(Ours)**	Gain (%)	Speedup
C1	50	479.59 (2.3)	487.16 (1.4)	484.88 (3.1)	**447.51** (2.6)	482.20 (3.1)	461.80 (2.3)	−3.19	**1.1**×
C1	75	811.97 (18.8)	776.63 (11.5)	781.43 (26.8)	816.62 (26.1)	801.36 (25.2)	**776.60** (23.0)	0.00	0.5×
C1	100	949.65 (29.1)	924.07 (23.9)	962.20 (43.7)	1017.13 (36.1)	1337.58 (37.7)	**919.42** (35.9)	0.50	0.7×
C2	50	1461.24 (15.2)	**1080.63** (3.5)	1461.22 (12.1)	1293.50 (19.2)	1379.02 (19.9)	1118.97 (2.1)	−3.55	**1.6**×
C2	75	2823.63 (31.9)	2310.73 (18.1)	2842.39 (24.3)	2879.89 (33.9)	2649.13 (43.0)	**1666.66** (13.5)	27.87	**1.3**×
C2	100	3462.20 (37.4)	2828.73 (28.8)	3325.89 (38.2)	4062.15 (52.6)	4108.05 (56.0)	**2598.43** (31.9)	8.14	0.9×
R1	50	916.89 (0.7)	918.65 (0.6)	923.25 (0.9)	917.47 (1.4)	941.99 (2.0)	**900.45** (0.6)	1.79	**1.1**×
R1	75	**1244.93** (4.5)	1260.64 (3.0)	1291.90 (5.3)	1254.84 (7.8)	1300.88 (11.6)	1253.08 (3.1)	−0.65	**1.4**×
R1	100	**1520.71** (20.8)	1569.93 (9.1)	1559.86 (21.5)	1610.02 (29.1)	1812.55 (35.0)	1552.53 (10.6)	−2.09	**2.0**×
R2	50	2048.09 (40.8)	1638.07 (14.9)	2289.24 (45.8)	2078.57 (33.2)	2300.92 (50.0)	**1632.37** (5.6)	0.35	**2.7**×
R2	75	3566.78 (53.3)	2828.35 (32.3)	3600.78 (54.3)	3538.96 (52.0)	3525.12 (55.5)	**2818.15** (33.5)	0.36	1.0×
R2	100	4678.47 (55.8)	4223.64 (46.8)	4633.36 (55.6)	4689.24 (56.6)	4480.23 (58.6)	**4184.69** (47.2)	0.92	1.0×
RC1	50	1421.91 (1.3)	**1134.31** (0.8)	1429.31 (1.4)	1271.90 (1.7)	1410.54 (2.5)	1165.87 (0.8)	−2.78	**1.0**×
RC1	75	1505.07 (7.3)	1643.00 (3.8)	1635.30 (5.7)	1654.54 (9.4)	1644.06 (12.7)	**1500.41** (2.7)	0.31	**2.7**×
RC1	100	1922.00 (16.1)	**1882.32** (8.4)	1971.17 (14.4)	1968.17 (24.3)	2258.36 (30.1)	1886.23 (8.3)	−0.21	**1.0**×
RC2	50	3464.67 (31.1)	3288.29 (23.0)	3579.94 (33.2)	3614.39 (30.8)	3392.25 (38.7)	**3134.03** (2.0)	4.69	**11.4**×
RC2	75	4350.91 (38.6)	4245.13 (28.1)	4514.18 (39.8)	4535.39 (37.2)	4425.03 (51.3)	**4173.75** (21.0)	1.68	**1.3**×
RC2	100	5867.65 (52.0)	5302.88 (36.4)	5805.75 (51.6)	5864.54 (53.8)	5934.63 (55.7)	**5297.46** (38.1)	0.10	1.0×

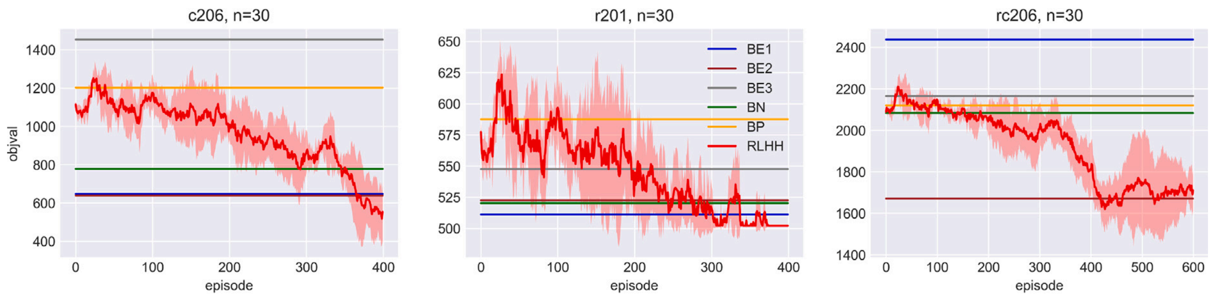

Fig. 2. VRPTW: training curves of RL agent for fixed instances.

value of the original problem. As shown in Fig. 2, we plot the training curves of the RL agent for fixed instances. The solid red lines represent the average value obtained with different random seeds, while the shaded region surrounding each line indicates the range within one standard deviation above and below the mean. The results suggest that the RL agent does not perform as well as most low-level heuristic methods when approaching a random selection strategy in the early stage. But over the course of training, the RL agent progressively learns the skill of selecting the appropriate low-level heuristic at each iteration, ultimately surpassing all baseline methods after approximately 400 episodes.

Test results. We evaluate the performance of the different methods using 50 small-size test instances sampled from types C2, R2, and RC2. The detail results of type R2 are presented in Table 3. For results pertaining to other types, please refer Table 8 in Appendix E. The RLHH method obtains the best integer solution for most instances, and consistently rank in the top 3 for all instances. The inclusion of computational time within parentheses significantly reduces. RLHH successfully solves all instances within 1 min, whereas other methods may not solve the problem until the specified time (10 min for small-size instances). Despite the RL agent exclusively utilizing C1, R1 and RC1 training data, it far outperforms other methods for instances from type C2, R2, RC2. It

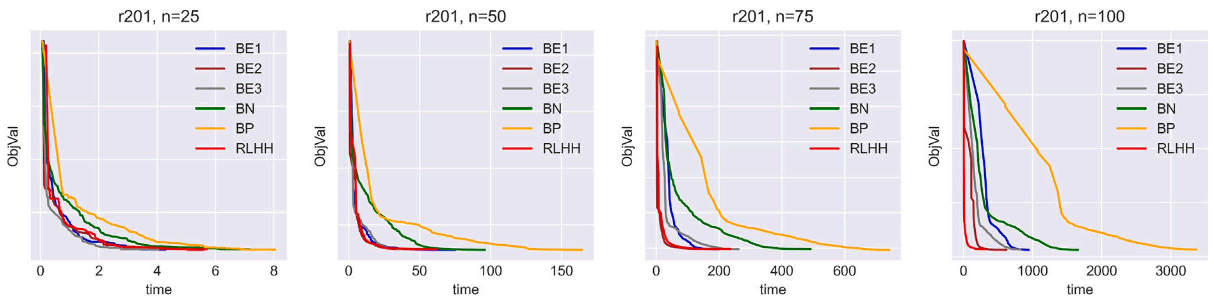

Fig. 3. VRPTW: Iteration process of all methods under different size (time is in seconds).

consistently achieves the optimal outcome in a majority of cases, which shows that RLHH has great ability to generalize across varying problem structures.

In addition, we generalize the trained model on small-size instances from types C1, R1 and RC1 to large-size ($n \in \{50, 75, 100\}$) instances across all types. In this set of experiments, we impose a maximum solving time limit of 1 h. For all instances of a specific type and size, we calculate the average of their objective values and report the mean computation time in minutes (see Table 4). The last two columns show the reduction of the objective value gained and the speedup in comparison with the best-performing baseline heuristic method in each case. For type C2 with $n = 75$, the RLHH obtains the reduction in the objective value of up to 27.87% within only 74.6% computation time (i.e. 1.34 times faster).

According to the results, our approach leads to a significant improvement over the best-performing low-level heuristic in most cases. It demonstrates that our method successfully combines the strength of OR heuristics with the generalization capabilities of RL, learning to discern the suitable low-level heuristic for each CG iteration. Furthermore, we test some larger instances with $n \geq 100$ to show the generalization capability of our method across problem sizes, refer to Table 9 in Appendix E.

Convergence process. We record the objective values of the RMP at each CG iteration for the given method using *r*201 as a test instance, considering problem size from 25 to 100 (see Fig. 3). There is little difference between the methods when $n \leq 50$, with BP and BN performing slightly worse. Nonetheless, as the number of customers increases, the advantages of RLHH become evident. Specifically, for $n = 100$, our approach surpasses all others and converges in approximately half the CPU time required by the top-performing low-level heuristic BE2. In large-size situations, RLHH dominates almost all other methods throughout the CG iterations. It means that if CG had to be terminated early, RLHH would give the best (minimum) objective value compared to other methods.

5.2. Bus driver scheduling problem

In a public transit system, the scheduling process involves multiple stages. The first step is to define the set of bus trips, specifying their start and end times, locations, and required transitions. Given this predefined timetable, the next step is to construct driver schedules that assign trips to drivers. This process is formulated as the Bus Driver Scheduling Problem (BDSP), where the objective is to determine the minimum number of drivers required to cover all trips while minimizing total working hours and ensuring compliance with labor agreement rules. In this study we consider maximum continuous driving time, maximum total working time, minimum break time between consecutive trips, and maximum total shift duration.

BDSP can be formulated as a set covering problem, where each column represents a feasible driver schedule. The linear relaxation of this formulation is given by the master problem (MP) (1)–(3). The set of feasible schedules is generated dynamically using column generation, by solving a pricing problem formulated as SPPRC.

Datasets. To the best of our knowledge, there is no publicly available dataset for BDSP, so we construct a random data generator referring (Haase et al., 2001). Given a total number of trips as input, we randomly distribute the trips over time, with an emphasis on peak hours. More precisely, the starting hour of each trip is determined according to the probability distribution given in Table 7 of Appendix C, while the starting minute follows a uniform distribution. Subsequently, we randomly sample an integer between 60 and 90 as the duration (in minutes) for each trip. We generate instances with $50 \leq n \leq 60$ for training, and $30 \times 5 = 150$ new instances with $n \in \{50, 75, 100, 150, 200\}$ are generated for testing. For instances with $n \leq 100$, we set the solution time limit to 10 min; for larger instances, the limit is extended to 1 h.

Test results. As in previous sections, Table 5 displays the average objective values for instances of different sizes. This illustrates that the RLHH performs better on large-size problem instances. When $n \geq 100$, RLHH exceeds all other methods, and the objective value at $n = 200$ is 15.41% lower than the best low-level heuristic, representing a great improvement. On average, the RLHH is faster than the best baseline for instances of all sizes. Notably, when $n = 200$, none of the methods can complete the CG convergence process within the designated time limit.

Convergence process. Fig. 4 displays the CG convergence process of various methods, considering instance size ranging from 50 to 200. The names of the instances are written in the form "bdsp_*Size*_*Id*", where *Size* is the number of trips, and *Id* is the instance identifier. Similar to BE2 and BN, RLHH converges quickly and is significantly better than other methods.

5.3. Adapting pretrained VRPTW model to BDSP

One of the key challenges in applying RL to combinatorial optimization problems is the extensive training time required to obtain well-performing policies. In industrial applications, where fast deployment is crucial, reducing the training burden without sacrificing solution quality is highly desirable. A promising approach to address this challenge is transfer learning, which leverages pre-trained models from a source problem to accelerate learning in a related target problem (Torrey & Shavlik, 2010).

The column generation subproblems in VRPTW and BDSP are highly similar, they may share certain patterns in low-level heuristic selection despite differences in network structure. Thus, we explore the transferability of our learned RLHH policy by pretraining the model on VRPTW and fine-tuning it on BDSP. Specifically, we examine whether this approach accelerates convergence and improves the final solution quality.

Experimental setup. To evaluate the transferability of the learned RLHH policy, we compare models with varying degrees of training and transfer learning strategies. The experiments are designed to assess the impact of pretraining on VRPTW and subsequent fine-tuning on BDSP

Table 5
BDSP: The average objective values for test instances (in parentheses is the solving time in minutes). Speedup is the ratio between the solving time of the best baseline and our method.

n	BE1	BE2	BE3	BN	BP	**RLHH(Ours)**	Gain (%)	Speedup
50	6736 (1.4)	5644 (0.7)	5972 (0.6)	**5605** (0.6)	6542 (0.8)	5627 (0.2)	−0.39	**2.3×**
75	12 829 (6.8)	**7538** (3.6)	11 307 (5.2)	10 039 (4.1)	11 389 (5.4)	7995 (1.9)	−6.06	**1.9×**
100	15 252 (10.4)	10 796 (9.9)	14 837 (10.3)	11 958 (10.8)	18 097 (10.1)	**9933** (8.2)	7.99	**1.2×**
150	18 887 (57.7)	14 597 (60.0)	20 926 (60.0)	19 005 (60.0)	17 681 (58.3)	**13 552** (57.9)	7.16	**1.0×**
200	26 303 (60.0)	27 858 (60.0)	42 671 (60.0)	22 762 (60.0)	25 441 (60.0)	**19 255** (60.0)	15.41	**1.0×**

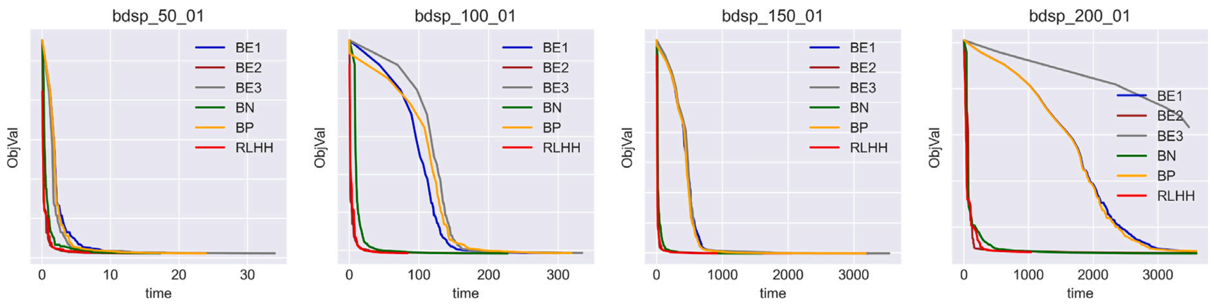

Fig. 4. BDSP: Iteration process of all methods under different size instances (time is in seconds).

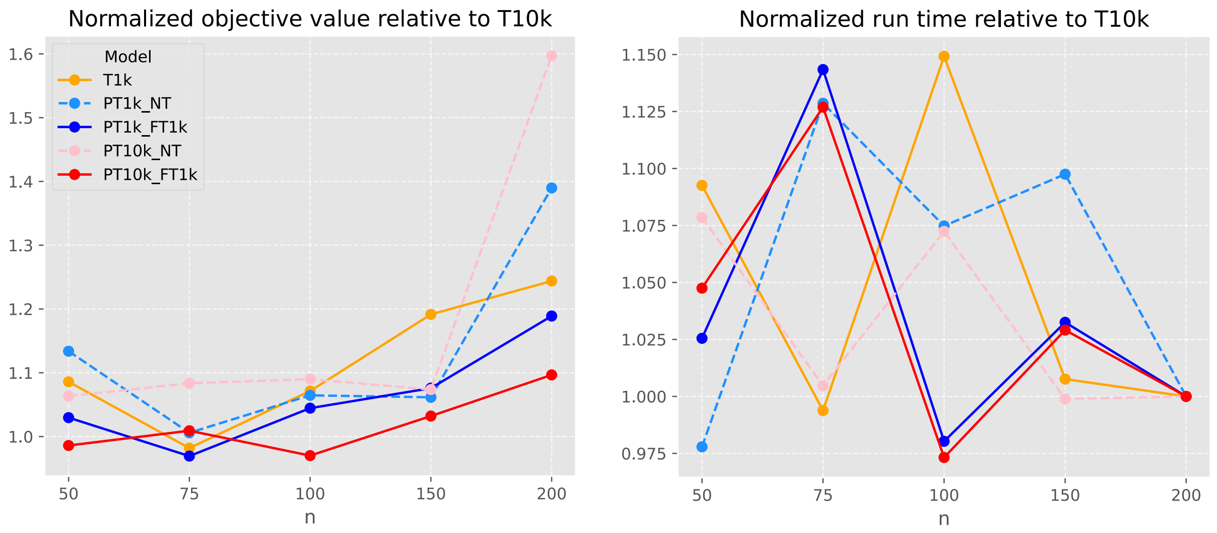

Fig. 5. Comparison of all models. The left plot shows normalized objective values relative to T10k, while the right plot presents normalized runtime, with lower values indicating better performance.

in terms of convergence speed, final integer solution quality, and computational efficiency. The six models are defined as follows:

- Train1k (T1k): A baseline model trained from scratch on BDSP for 1000 episodes.
- Train10k (T10k): A fully trained BDSP model trained from scratch for 10,000 episodes (as presented in Section 5.2).
- PreTrain1k_NoTune (PT1k_NT): A model pretrained on 1000 episodes of VRPTW instances, directly applied to BDSP without any further fine-tuning.
- PreTrain1k_FineTune1k (PT1k_FT1k): A model pretrained on 1000 episodes of VRPTW instances, then fine-tuned on BDSP for 1000 additional episodes.
- PreTrain10k_NoTune (PT10k_NT): A model pretrained on 10,000 episodes of VRPTW, directly applied to BDSP without fine-tuning.
- PreTrain10k_FineTune1k (PT10k_FT1k): A model pretrained on 10,000 episodes of VRPTW, then fine-tuned on BDSP for 1000 additional episodes.

Test results. Table 6 shows that pretraining significantly reduces training time while maintaining competitive performance. The PT10k_FT1k model, which requires only one-tenth of the training episodes compared to T10k, achieves a similar overall ranking (2.2 vs. 1.8) and even outperforms T10k on instances with n = 50 and n = 100. This highlights the potential of transfer learning to accelerate training without substantial loss in solution quality. Additionally, both PT1k_NT and PT10k_NT show clear improvements after fine-tuning, reinforcing the importance of adapting pretrained models to the target domain.

Interestingly, PT10k_NT underperforms PT1k_NT, suggesting that while extensive VRPTW pretraining captures richer patterns, it may also introduce biases that hinder direct application to BDSP. For example, we can find from the previous experimental results that BE1, which is good at VRPTW, does not perform well in BDSP, while BN does the opposite. This indicates that excessive pretraining on a different domain without fine-tuning can be counterproductive. In terms of runtime (Fig. 5), all models exhibit similar computational efficiency, with differences within 15%, as they share the same RLHH framework. Overall, these results demonstrate that transfer learning can significantly reduce

Table 6
Fine-tuning experiment results. Obj. denotes the objective value, while Avg. represents the average value across different problem scales.

Model	T1k		T10k		PT1k_NT		PT1k_FT1k		PT10k_NT		PT10k_FT1k	
n	Obj.	Rank	Obj.	Rank	Obj.	Rank	Obj.	Rank	Obj.	Rank	Obj.	Rank
50	6110	5	5627	2	6380	6	5794	3	5984	4	5548	1
75	7850	2	7995	3	8042	4	7750	1	8663	6	8068	5
100	10 642	5	9933	2	10 575	4	10 375	3	10 826	6	9636	1
150	16 148	6	13 552	1	14 386	3	14 580	5	14 551	4	13 987	2
200	23 950	4	19 255	1	26 761	5	22 894	3	30 749	6	21 116	2
Avg.	12 940	4.4	11 272	1.8	13 229	4.4	12 279	3.0	14 155	5.2	11 671	2.2

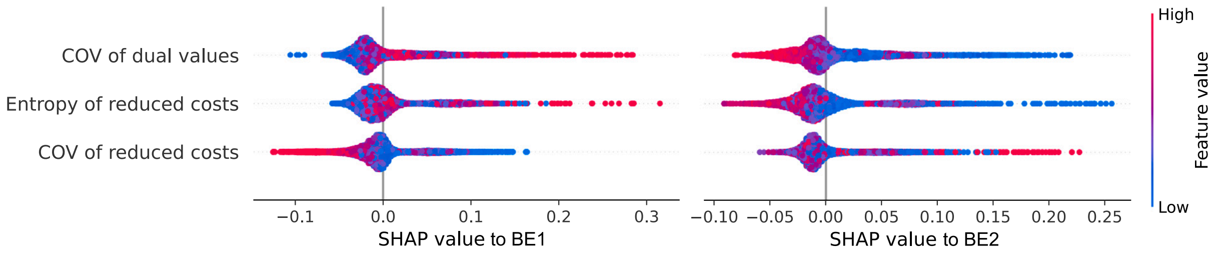

Fig. 6. Contribution of the features to action BE1 and BE2.

the training burden while ensuring strong performance, provided that appropriate fine-tuning is applied.

5.4. Feature interpretability analysis

Understanding the contribution of different features to the decision-making process of the RL agent is crucial for both model interpretability and practical deployment. While deep RL models often function as black boxes, interpreting their decisions helps validate learned policies and provide insights into the problem structure.

To achieve this, we employ SHapley Additive exPlanations (SHAP), a widely used interpretability method based on cooperative game theory (Lundberg et al., 2018). SHAP assigns each feature a contribution value by computing the marginal effect of its inclusion across all possible feature combinations, ensuring a fair and consistent measure of feature importance. Unlike traditional feature importance metrics, SHAP values account for feature interactions, making it particularly well-suited for complex models.

We analyzed 10,000 sampled state–action pairs (s_t, a_t) from the VRPTW-trained model and focused on the two most frequently selected heuristics: BestEdges1 (BE1) and BestEdges2 (BE2). These two heuristics were also the top-performing methods in the VRPTW experiments. Fig. 6 presents the SHAP value distribution for the three most influential features: COV of dual values, entropy of reduced costs and COV of reduced costs. These features provide key insights into how the RL agent selects between BE1 and BE2 in different scenarios.

The SHAP analysis reveals that BE1 selection is positively influenced by the COV of dual values, as greater variability helps identify and remove high-cost edges, while lower entropy of reduced costs further enhances its effectiveness by making cost differences more distinct. In contrast, BE2 selection is primarily driven by the entropy of reduced costs, as a more diverse cost distribution allows for better differentiation of low-cost edges, with COV of reduced costs also having a positive effect by reinforcing this distinction. This aligns with BE2's mechanism, which selects a fraction of the lowest-cost edges, benefiting from a wider spread in reduced costs.

Since both COV and entropy can be used to measure the dispersion of a distribution, we analyze the interaction between these two features. Fig. 7 demonstrates the interaction between COV of reduced costs and entropy of reduced costs for BE1 and BE2 selection. For BE1, higher COV of reduced costs decreases SHAP values, indicating that BE1 is favored when cost variability is low, especially when entropy is high. In contrast, for BE2, SHAP values increase with COV of reduced costs, suggesting that BE2 is more effective when reduced costs exhibit greater variability, particularly when entropy is also high. These results confirm that BE1 relies more on entropy-driven cost distinctions, while BE2 benefits from higher cost variability.

6. Conclusion

In this study, we present a novel approach that leverages the strength of OR heuristics while incorporating the learning and generalization capabilities of RL models. Specifically, we introduce an RL-based hyper-heuristic that enhances the CG technique for vehicle routing and scheduling problems, where the pricing problem is formulated as the SPPRC or its variants. During each CG iteration, the RL agent selects a low-level heuristic to generate a reduced network that retains only the most promising edges. Our comparative experiments demonstrate that, after appropriate training, the proposed algorithm effectively combines the advantage of various low-level heuristics, achieving significant improvements compared with single-heuristic approaches. Furthermore, our RL model demonstrates strong generalization, successfully transferring learned policies across instance scales and problem domains. This highlights RLHH's potential for cross-domain adaptation, further enhancing its practicality in industrial applications.

Beyond VRPTW and BDSP, our approach is adaptable to a broader range of real-world combinatorial optimization problems. For more intricate scheduling scenarios, such as integrated vehicle and crew scheduling problems, we need to utilize intricate network structures. Additional constraints can be incorporated by defining new resources in SPPRC or constructing more sophisticated network structures. Future research may explore the extension of RLHH to railway crew scheduling, airline crew scheduling, and other graph-based combinatorial optimization problems within the CG framework, such as graph coloring problems.

While the reinforcement learning task considered in this study is relatively straightforward, future work could enable the agent to not only select heuristics but also dynamically adjust their parameters. This may require a more comprehensive exploration of the state space, potentially employing techniques like graph neural networks (GNNs). Alternatively, policy-based RL methods could be explored to address more complex decision spaces. These directions could further enhance the adaptability and effectiveness of RLHH across a wider range of challenging optimization problems.

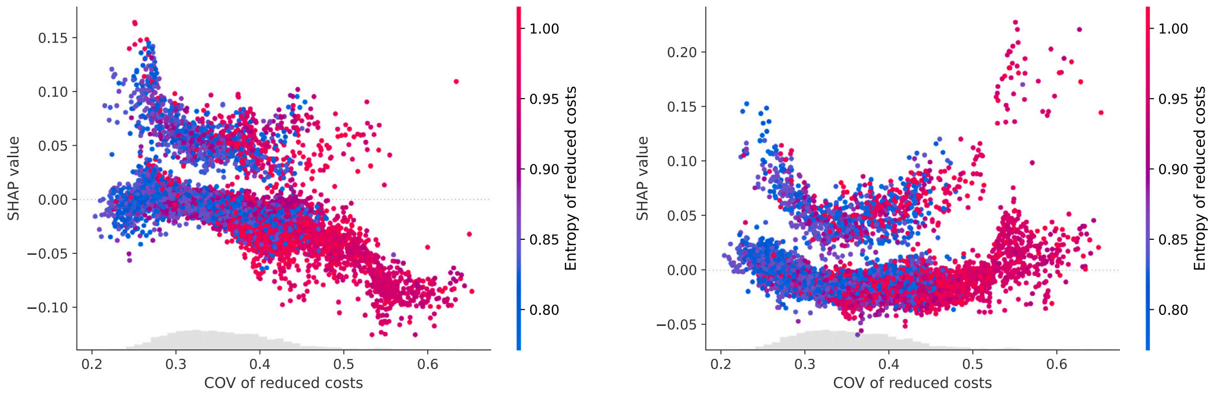

Fig. 7. SHAP interaction effects between COV of reduced costs and entropy of reduced costs for BE1 (left) and BE2 (right). Color represents entropy values, with higher entropy in red and lower entropy in blue.

CRediT authorship contribution statement

Kuan Xu: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Software. **Li Shen:** Conceptualization, Methodology, Writing – review & editing. **Lindong Liu:** Funding acquisition, Writing – review & editing, Conceptualization, Methodology.

Acknowledgments

The work was supported by the National Natural Science Foundation of China [Grant Nos. 72471216, 72022018, 72091210] and Youth Innovation Promotion Association, Chinese Academy of Sciences [Grant No. 2021454].

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cie.2025.111138.

Data availability

Data will be made available on request.

References

Alberto Santini, Plum, C. E., & Ropke, S. (2018). A branch-and-price approach to the feeder network design problem. *European Journal of Operational Research*, *264*(2), 607–622.

Babaki, B., Charlin, L., & Jena, S. D. (2022). *COIL: A Deep Architecture for Column Generation*. Bureau de Montreal, Université de Montreal.

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

Chi, C., Aboussalah, A., Khalil, E., Wang, J., & Sherkat-Masoumi, Z. (2022). A deep reinforcement learning framework for column generation. *Vol. 35*, In *Advances in neural information processing systems* (pp. 9633–9644).

Costa, L., Contardo, C., & Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, *53*(4), 946–985.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2006). *vol. 5*, *Column Generation*. Springer Science & Business Media.

Desaulniers, G., Lessard, F., & Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*(3), 387–404.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*, 342–354.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, *42*(5), 977–978.

Furian, N., O'Sullivan, M., Walker, C., & Çela, E. (2021). A machine learning-based branch and price algorithm for a sampled vehicle routing problem. *OR Spectrum*, *43*(3), 693–732.

Gamache, M., Soumis, F., Marquis, G., & Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, *47*(2), 247–263.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., & Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Vol. 32*, In *Advances in neural information processing systems*. Curran Associates, Inc..

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, *9*(6), 849–859.

Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—Part II. *Operations Research*, *11*(6), 863–888.

Gupta, P., Khalil, E. B., Chételat, D., Gasse, M., Lodi, A., Bengio, Y., & Kumar, M. P. (2022). Lookback for learning to branch. *Transactions on Machine Learning Research*.

Haase, K., Desaulniers, G., & Desrosiers, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, *35*(3), 286–303.

Hottung, A., & Tierney, K. (2022). Neural large neighborhood search for routing problems. *Artificial Intelligence*, *313*, Article 103786.

Jacobs, T., Alesiani, F., Ermis, G., & Zhou, Z. (2021). Reinforcement learning for route optimization with robustness guarantees.. In *International joint conference on artificial intelligence* (pp. 2592–2598).

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *vol. 30*, In *Advances in neural information processing systems*.

Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., & Shao, Y. (2017). Learning to run heuristics in tree search.. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence* (pp. 659–666).

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

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., & Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, *33*(1), 101–116.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems!. In *International conference on learning representations*.

Liu, X., Peng, Y., Zhang, G., & Zhou, R. (2024). An efficient node selection policy for Monte Carlo tree search with neural networks. *INFORMS Journal on Computing*, Article ijoc.2023.0307.

Lombardi, M., & Milano, M. (2018). Boosting combinatorial problem modeling with machine learning. In *Proceedings of the 27th international joint conference on artificial intelligence* (pp. 5472–5478).

Lu, H., Zhang, X., & Yang, S. (2019). A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*.

Lundberg, S. M., Erion, G. G., & Lee, S.-I. (2018). Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888.

Ma, Y., Li, J., Cao, Z., Song, W., Guo, H., Gong, Y., & Chee, Y. M. (2022). Efficient neural neighborhood search for pickup and delivery problems. In *Proceedings of the thirty-first international joint conference on artificial intelligence vienna, Austria* (pp. 23–29).

Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., & Tang, J. (2021). Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *vol. 34*, In *Advances in neural information processing systems* (pp. 11096–11107).

Mauro Dell'Amico, Righini, G., & Salani, M. (2006). A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, *40*(2), 235–247.

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

Morabit, M., Desaulniers, G., & Lodi, A. (2021). Machine-learning–based column selection for column generation. *Transportation Science*, *55*(4), 815–831.

Morabit, M., Desaulniers, G., & Lodi, A. (2023). Machine-learning-based arc selection for constrained shortest path problems in column generation. *INFORMS Journal on Optimization*, *5*(2), 191–210.

Perumal, S. S., Larsen, J., Lusby, R. M., Riis, M., & Christensen, T. R. (2022). A column generation approach for the driver scheduling problem with staff cars. *Public Transport*, *14*(3), 705–738.

Qin, W., Zhuang, Z., Huang, Z., & Huang, H. (2021). A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem. *Computers & Industrial Engineering*, *156*, Article 107252.

Sebastian Kraul, Seizinger, M., & Brunner, J. O. (2023). Machine learning–supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation. *INFORMS Journal on Computing*, *35*(3), 692–709.

Smith, B. M., & Wren, A. (1988). A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, *22*(2), 97–108.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, *35*(2), 254–265.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., & Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, *261*(2), 530–539.

Torrey, L., & Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques* (pp. 242–264). IGI global.

Václavík, R., Novák, A., Šůcha, P., & Hanzálek, Z. (2018). Accelerating the branch-and-price algorithm using machine learning. *European Journal of Operational Research*, *271*(3), 1055–1069.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *vol. 28*, In *Advances in neural information processing systems*.

Yan, J., Yang, S., & Hancock, E. (2020). Learning for graph matching and related combinatorial optimization problems. In *Proceedings of the twenty-ninth international joint conference on artificial intelligence* (pp. 4988–4996).

Yuan, H., Jiang, P., & Song, S. (2022). The neural-prediction based acceleration algorithm of column generation for graph-based set covering problems. In *2022 IEEE international conference on systems, man, and cybernetics* (pp. 1115–1120). IEEE.

Zhang, Y., Bai, R., Qu, R., Tu, C., & Jin, J. (2022). A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, *300*(2), 418–427.