Knowledge-Based Systems 300 (2024) 112173

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Token-based deep reinforcement learning for Heterogeneous VRP with Service Time Constraints

Yujun Wang [a], Xiaopeng Hong [b,*], Yabin Wang [a], Junzhou Zhao [a], Guanghui Sun [c], Baoxing Qin [d]

[a] School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an, 710049, Shannxi, China
[b] Faculty of Computing, Harbin Institute of Technology, Harbin, 150001, Heilongjiang, China
[c] School of Astronautics, Harbin Institute of Technology, Harbin, 150001, Heilongjiang, China
[d] Gaussian Robotics, Shanghai, 201210, China

ARTICLE INFO

Keywords:
Heterogeneous Vehicle Routing
Service Time Constraints
Task allocation
Task scheduling
Deep reinforcement learning

ABSTRACT

Heterogeneous Vehicle Routing aims to construct routes for various vehicles while optimizing an objective with a series of constraints. However, existing deep reinforcement learning-based methods often ignore the service time constraints, which prohibits vehicles from leaving current nodes until the service time is met. This limitation restricts their practical application. To address these concerns, we introduce the Heterogeneous Vehicle Routing Problem with Service Time Constraints (HVRP-STC) and formulate it as a Markov Decision Process with *Service Time Constraints*. We propose a novel deep reinforcement learning-based model, Token-based Deep Reinforcement Learning (TDRL), to solve this problem.

To provide sufficient and timely information for decision making, we design a State Token Coding (STC) mechanism that encodes and updates individual and overall vehicle and node states as tokens of different types. To determine the pairs of vehicles and nodes and generate actions, we propose a Heterogeneous Decoder (HD) with a vehicle-selector and multiple vehicle-specific node-selectors. This decouples the vehicle-node selection tasks and customizes the task of choosing nodes to visit for individual vehicles, better catering to the heterogeneous nature of HVRP-STC.

We evaluate the proposed method on four types of datasets with instances of different sizes, large spatial coverage, and varied mathematical model. Our results show that TDRL consistently outperforms state-of-the-art DRL methods. We will release the datasets and the source code of this benchmark with the paper via https://github.com/Vision-Intelligence-and-Robots-Group/ToDRL.

1. Introduction

Heterogeneous Vehicle Routing Problem (HVRP) is a classical combinatorial optimization problem [1], which has a significant impact in computational science and knowledge-based systems [2]. Briefly, HVRP is to construct routes for heterogeneous vehicles to optimize an objective under a series of constraints. Various well-known problems, such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP), are simplified versions of HVRP. HVRP has numerous practical applications, like path planning [3], transportation control [4], logistics optimization [5], and so forth.

Traditional researches for HVRP fall into two categories: exact methods [6–8] and heuristics [9–11]. Exact methods find the optimal solution from a solution space using strategies like LP-relaxation [6] and branch-cut-and-price (BCP) [7,8], etc. However, considering that HVRP is NP-hard [12], the theoretical computation complexity grows exponentially with the problem size, which leads to the fact that exact methods can only deal with small-scale HVRP instances in a reasonable time. Heuristics explore the solution space of HVRP through a set of strategies to find a feasible solution. They sacrifice the quality of the solution while shortening the exploration time. Nevertheless, the time consumption will also become tremendous if the approximate solution is approaching the optimal one.

With the explosion of computing power and the development of deep learning and reinforcement learning, deep reinforcement learning based HVRP methods have emerged [13–15] in recent years. They utilize the parallel processing power of the Graphics Processing Unit (GPU) for acceleration, and thus significantly reduce the solving time. Such methods can learn from massive data to improve performance.

Nonetheless, there are limitations for existing deep learning based HVRP approaches, which leaves the problem largely unaddressed.

* Corresponding author.
E-mail addresses: yujun.wang@stu.xjtu.edu.cn (Y. Wang), hongxiaopeng@hit.edu.cn (X. Hong), iamwangyabin@stu.xjtu.edu.cn (Y. Wang), junzhou.zhao@xjtu.edu.cn (J. Zhao), guanghuisun@hit.edu.cn (G. Sun), baoxing@gs-robot.com (B. Qin).

https://doi.org/10.1016/j.knosys.2024.112173
Received 30 May 2023; Received in revised form 10 June 2024; Accepted 17 June 2024
Available online 1 July 2024
0950-7051/© 2024 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

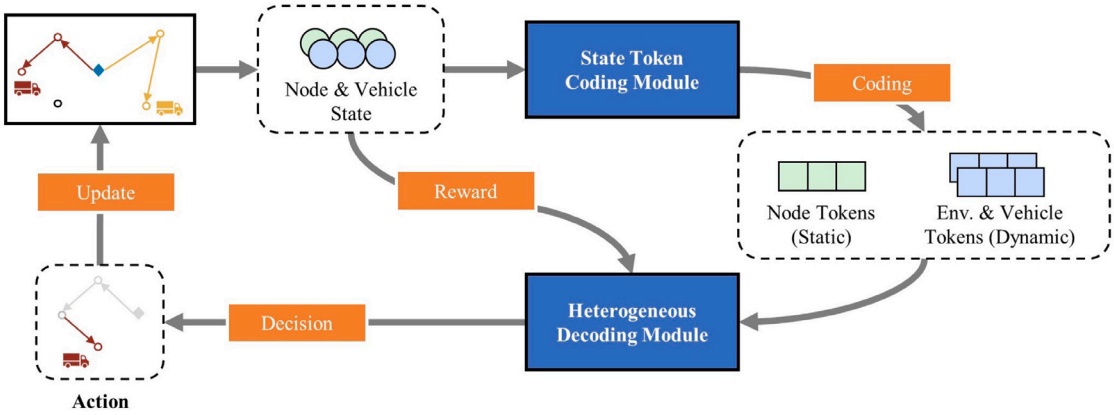


Fig. 1. The framework of our approach. The upper left image portrays an initial solution for HVRP-STC, which is incomplete as the routes of each vehicle solely consist of the depot node. Our approach utilizes the State Token Coding Module to construct the state of MDP-STC and maintain tokens based on the incomplete solution. These tokens are subsequently transferred to the Heterogeneous Decoding Module, which generates an action represented as a 2-tuple (v, p). This action dispatches vehicle v to node p, thereby updating the incomplete solution. Subsequently, our approach iterates to generate a new state and repeats the aforementioned procedure until all task nodes are visited, resulting in the formation of a complete solution. Through this iterative process, the proposed approach efficiently solves the HVRP-STC.

Firstly, existing RL-based methods [13–15] simply assume that tasks can be finished in an instant and ignore the *service time constraints* that vehicles are not allowed to leave the current nodes until the *service time* is met. However, in many real-life scenarios, such as loading and unloading goods or cleaning up garbage, service time is non-negligible. Their application in practice is thus greatly restricted. Secondly, existing vehicle selection modules, such as the vehicle selection decoder in [14], only consider the nodes visited and ignore the others when selecting vehicles. However, it is suboptimal to make a decision on which vehicles shall be selected without the information about unvisited nodes, since the purpose of VRP is to schedule vehicles for all nodes. Thirdly, typical HVRP approaches like [14] employ a composite *node selection* decoder to select nodes in routes. However, considering the complex heterogeneous nature of vehicles, it is difficult to train such a composite decoder to manage all route nodes for all vehicles.

In response to the above issues, we focus on the Heterogeneous Vehicle Routing Problem with *Service Time Constraints*, and formulate this problem as a Markov Decision Process with *Service Time Constraints* (MDP-STC). We propose a novel deep reinforcement learning model for solving it, which is termed as ToDRL. The framework is illustrated in Fig. 1. Different from existing approaches, ToDRL leverages a state token coding mechanism to encode and update the individual and overall state of vehicles and nodes as static or dynamic tokens. More concretely, we design three types of tokens, namely the *node token*, *vehicle token*, and *environment token*. A node token represents the state of a specific node, as does a vehicle token. In contrast, the environment token is an overall description of states of all vehicles and nodes. All tokens are initialized at the beginning by the initializer module. After that, ToDRL freezes the node tokens and leaves them unchanged. In each iteration, the encoder module updates the vehicles as well as the environment tokens to capture the dynamic change of the vehicles and environment. The Heterogeneous Decoder, which consists of a vehicle-selector and multiple vehicle-specific node-selectors, takes all tokens as input and pairs the vehicle and the node. Firstly, HD chooses a vehicle suitable with current environment state. Then, HD decouples the node selection task to a series of vehicle-specific subtasks. It leverages multiple vehicle-specific node-selectors to determine the nodes to visit for individual vehicles. We perform extensive experiments to validate the superiority and robustness of ToDRL based on three datasets of HVRP-STC with instances of different sizes and spatial distributions.

In summary, we raise the concern about the complicated yet practical Heterogeneous Vehicle Routing Problem with *Service Time Constraints*. The main contributions are summarized as follows:

1. We formulate the HVRP-STC problem as a Markov Decision Process with the *Service Time Constraints* and propose a novel deep reinforcement model for solving it.
2. We design a state token coding mechanism, which encodes and updates the individual and overall states of vehicles and nodes as well as different static or dynamic tokens to provide sufficient and timely clues for making decisions.
3. We propose a Heterogeneous Decoder for decision making iteratively, which caters well to the heterogeneous nature of HVRP-STC problem.

The rest of the paper is laid out as follows. The research pertaining to this paper is succinctly summarized in Section 2. Section 3 gives the mathematical description of HVRP-STC problem. Section 4 presents our methodology in a tangible manner. In Section 5, we meticulously elaborate upon the specifics of the implementation, experimental configuration, and discoveries. Section 6 concludes this paper.

2. Related work

Besides the exact methods [6–8], there are two main categories of approaches for HVRP: traditional heuristics and reinforcement learning-based approaches

2.1. Traditional heuristics

Over the past decades, many heuristics have been proposed to solve classical HVRP and its variants. These approaches can be categorized into four types.

2.1.1. Single solution-based heuristics

This kind of heuristics, such as Tabu Search (TS), Neighborhood Search (NS), generates a feasible solution first and then optimizes it iteratively. Lai et al. [16] developed a tabu search heuristic that efficiently addressed a time-constrained HVRP on a multigraph. Mancini [17] introduced a multi-depot multi-period HVRP, in which the customers may be served in different periods, and solved it based on the adaptive large neighborhood search heuristic. Wang et al. [18] integrated a constructive heuristic with two-layer simulated annealing and tabu search to solve the HVRP with cross docks and split deliveries. Simeonova et al. [19] presented a variable neighborhood search algorithm to tackle the HVRP, which has demand-dependent service time, an unlimited number of vehicles, and additional overtime constraints by hybridizing adaptive memory and classical iterative memoryless method. Meliani et al. [20] employs tabu search to construct solution for heterogeneous VRP with 3D loading constraints.

Table 1
Summary of notations.

HVRP-STC	
n	The number of task nodes.
m	The number of vehicles.
N_P	The set of integers $\{0, 1, \dots, n\}$.
N_V	The set of integers $\{1, 2, \dots, m\}$.
N_P^-	The set of integers $N_P \backslash \{0\}$.
P	The set of one depot node and n task nodes. Each node $p_i \in P$ is numerically indexed by a unique integer $i \in \{0, \dots, n\}$, where 0 is index of the depot node.
P'	The set of task nodes. $P' = P \backslash \{p_0\}$.
V	The set of vehicles. Each vehicle $v_k \in V$ is indexed by a unique integer $k \in \{1, \dots, m\}$.
p_i	It is defined as a 3-tuple $(\ell_{pi}, d_{pi}, o_{pi})$, where ℓ_{pi}, d_{pi}, and o_{pi} are the coordinate, demand, and workload of p_i respectively.
v_k	It is defined as a 2-tuple (q_{vk}, c_{vk}), where q_{vk} and c_{vk} are the speed and capacity of v_k respectively.
MDP-STC	
T	The number of decision steps. $0 \le t \le T$, where t is the index of a decision step.
X_P^t	The set of node features. $X_P^t = \{x_{p0}^t, x_{p1}^t, \dots, x_{pn}^t\}$, where $x_{pi}^t = (\ell_{pi}, d_{pi}, o_{pi}, e_{pi}^t)$. ℓ_{pi}, d_{pi}, and o_{pi} are the location, demand, and workload of node p_i. e_{pi}^t is an indicator variable. $e_{pi}^t = 1$ if node p_i has been visited before step t, and $e_{pi}^t = 0$ otherwise.
X_V^t	The set of vehicle features. $X_V^t = \{x_{v1}^t, \dots, x_{vm}^t\}$, where $x_{vk}^t = (q_{vk}, c_{vk}, b_{vk}^t, \ell_{vk}^t, w_{vk}^t)$. q_{vk} and c_{vk} are the speed and capacity of the vehicle v_k. b_{vk}^t, ℓ_{vk}^t, and w_{vk}^t are the number of remaining goods, location, and work time of vehicle v_k at step t respectively.
O	The value of objective.
S	The state space. $s^t = (X_P^t, X_V^t) \in S$ is the state at step t, where $X_P^t = \{(\ell_{pi}, d_{pi}, o_{pi}, e_{pi}^t) \mid i \in N_P\}$ is the set of node states. e_{pi}^t is an indicator variable. $e_{pi}^t = 1$ if node p_i has been visited before step t, and $e_{pi}^t = 0$ otherwise. $X_V^t = \{(q_{vk}, c_{vk}, b_{vk}^t, \ell_{vk}^t, w_{vk}^t) \mid k \in N_V\}$ is the set of vehicle features, where b_{vk}^t, ℓ_{vk}^t, and w_{vk}^t are the number of remaining goods, location, and work time of vehicle v_K at step t respectively.
A	The action space. $a^t \in A$ is the action at step t
$\mathbb{T}$	The state transition rules.
R	The reward function. $R = \sum_{t=0}^{T} r^t$, where r^t is the immediate reward at step t.
Tokens	
Z_P	The set of node tokens. $z_{pi} \in Z_P$ is the token of p_i.
Z_V^t	The set of vehicle tokens at step t. $z_{vk}^t \in Z_V^t$ is token of v_k.
Z_E^t	The token of environment at step t.
Network	
C_{en}^t	The context vector of encoder module at step t.
C_{de}^t	The context vector of heterogeneous decoder at step t.
H^ℓ	The output of the ℓth embedding layer.
π_θ	ToDRL with parameters θ.
$\varphi_\theta(\tau \mid \mathcal{X})$	The probability that ToDRL with parameters θ constructs solution τ for instance $\mathcal{X}$.
Experiment	
$\mathcal{X}$	An Instance of HVRP-STC problem, which consists of nodes P and vehicles V.
K	The number of instances in the test dataset.
AO	The Average value of Objective.
GAP	The gap between the method under consideration and the best known method so far in terms of AO.
Time	The running time of a method.

2.1.2. Evolutionary computation-based heuristics

Such methods, like Genetic Algorithm (GA), build a population of solutions first, then optimize it iteratively based on the mechanisms inspired by biological evolution. Baniamerian et al. [21] introduced a profitable HVRP with cross docks and presented a genetic algorithm-based heuristic to solve it. This method evolves the population of solutions progressively by crossover operator, mutation operator, and neighborhood search. To address the HVRP with backhauls, Sethanan and Jamrus [22] proposed a novel hybrid differential evolution algorithm based on a genetic operator with a fuzzy logic controller. Recently, the hybrid adaptive large neighborhood search [23] has been introduced to solve the two-echelon VRP.

2.1.3. Swarm intelligence-based heuristics

These methods, such as Ant Colony Optimization (ACO) and Particle swarm optimization (PSO), are generally inspired by the group behaviors of various organisms in nature. Molina et al. [24] proposed a modified ant colony optimization algorithm, combining with local search, to address the HVRP with time windows. Bansal and Wadhawan [10] developed a hybrid approach based on sine cosine algorithm and PSO to improve the efficiency of exploration of the solution space.

2.1.4. Solver-based heuristics

These methods are built based on optimization solvers, such as Gurobi [25], Google ORTools [26], and CPLEX [27], which are generic, portable, and have the ability to tackle a variety of optimization problems, including HVRP-STC.

2.2. Reinforcement learning-based methods

In recent years, reinforcement learning has shown promising performance in tasks like combinatorial optimization [28] and transportation systems [29]. But methods that directly address the HVRP are still relatively few. So we also briefly review the RL-based methods for general vehicle routing problems. According to the solution construction process, these methods can be divided into two categories: constructive methods and improvement-based methods.

2.2.1. Constructive methods

These methods select an unvisited node and add it to a partial solution at each step, which eventually forms a complete solution. Bello et al. [30] proposed a representative constructive method in 2016, which is used it to solve the Traveling Salesman Problem (TSP). Further, Nazari et al. [31] and Kool et al. [32] were the pioneers in solving the CVRP with this approach. Kool et al. [32] improved the structure of the pointer network [33] based on the attention mechanism of Transformer [34] and presented a model named AM, which has achieved milestone successes in a variety of routing problems, such as TSP, CVRP, etc. Since then, many AM-based routing methods, such as AR [35] and POMO [36], have emerged. Recently, Li et al. [14] introduced a vehicle-selector to enable adaptive selection of vehicles. and outperforms most existing methods. Bi et al. [37] uses knowledge distillation to improve the generalization ability of reinforcement learning models on unseen distributions.

2.2.2. Improvement-based methods

These methods modify an existing solution to improve its quality, and finally form a more optimal solution. Hottung and Tierney [38] utilized reinforcement learning to automatically develop the Large Neighborhood Search repair operators, improving the search efficiency. Intending to break the limitation of hand-crafted rules, Wu et al. [39] designed a novel network architecture based on the self-attention mechanism. They achieved a promising performance on TSP and VRP. Qin et al. [15] combine several heuristics through reinforcement learning to optimize a solution set of HVRP. It beats single heuristics, but is still plagued by the long solution time of the combined heuristics. Kim et al. [40] utilizes graph network and reinforcement learning to reduce the time complexity of CE and achieved better performance on Min-Max VRP.

3. Problem definition

This section provides a mathematical description of the Heterogeneous Vehicle Routing Problem with Service Time Constraints (HVRP-STC). For clarity, Table 1 lists the meanings of the symbols involved. Let n denote the number of tasks and m the number of vehicles. To facilitate a concise and clear exposition, we introduce the following definitions: $N_P = \{0, 1, \dots, n\}$, $N_P^- = N_P \backslash \{0\}$, $N_V = \{1, 2, \dots, m\}$. Let $P = \{p_i \mid i \in N_P\}$ represent the set of nodes, where p_0 signifies the depot node, while the remaining elements represent task nodes. Thus, we employ $P' = P \backslash \{p_0\}$ to represent the set of all task nodes.

Each node p_i is characterized by the tuple $(\ell_{pi}, d_{pi}, o_{pi})$, where ℓ_{pi}, d_{pi}, and o_{pi} respectively denote the coordinates, demand, and workload associated with p_i. Assume that there is a heterogeneous fleet with m vehicles represented by V, where $V = \{v_k \mid k \in N_V\}$. Each vehicle v_k is defined as (q_{vk}, c_{vk}), where q_{vk} and c_{vk} represent the speed and capacity of v_k respectively. The *service time* required for v_k to handle p_i is then defined as o_{pi}/q_{vk}. Furthermore, let $Z_P^t = \{z_{pi}^t \mid i \in N_P\}$, $Z_V^t = \{z_{vk}^t \mid k \in N_V\}$, and Z_E^t denote the tokens of nodes, vehicles and environment at step t respectively. Define a set of variables Y as follows:

$$Y = \{y_{ij}^k \mid i, j \in N_P, k \in N_V\}, \tag{1}$$

where y_{ij}^k serves as an indicator variable. We define that $y_{ij}^k = 1$ if vehicle v_k travels directly from p_i to p_j, and $y_{ij}^k = 0$ otherwise. Let Z denote another set of variables:

$$Z = \{z_{ij} \mid i, j \in N_P\}, \tag{2}$$

where z_{ij} is the quantity of goods carried by the vehicle directly from node p_i to node p_j. The objective of the HVRP-STC problem is formulated as:

$$O = \min_{y_{ij}^k} \max_k \sum_{i=0}^{n} \sum_{j=0}^{n} \frac{(\|l_j - l_i\|_2) y_{ij}^k}{q_k} + \frac{o_j y_{ij}^k}{q_k}. \tag{3}$$

It minimizes the time consumption of the vehicle that requires the most time, which considers travel time and *service time* jointly. The objective is subject to the following six constraints:

$$\sum_{k=1}^{m} \sum_{i=0}^{n} y_{ij}^k = 1, \qquad j \in N_P^-, \tag{4}$$

$$\sum_{i=0}^{n} y_{ij}^k = \sum_{l=0}^{n} y_{jl}^k, \qquad j \in N_P, k \in N_V, \tag{5}$$

$$z_{0j} \le \sum_{k=1}^{m} c_{vk} \times y_{0j}^k, \qquad j \in N_P, \tag{6}$$

$$\sum_{i=0}^{n} z_{ij} - \sum_{l=0}^{n} z_{jl} = d_{pj}, \qquad j \in N_P^-, \tag{7}$$

$$z_{ij} \le M \times \sum_{k=1}^{m} y_{ij}^k, \qquad i, j \in N_P, \tag{8}$$

$$y_{ij}^k \in \{0, 1\}, z_{ij} \ge 0, \qquad i, j \in N_P, k \in N_V, \tag{9}$$

Constraint (4) guarantees that each node, except the depot, is visited exactly once. Constraint (5) ensures that the routes of each vehicle are closed and continuous. Constraint (6) guarantees that the total demand for goods on any route does not exceed the capacity of the corresponding vehicle. Constraint (7) guarantees that if a vehicle visits node i, the quantity of its good will be reduced by d_{pi}. Constraint (8) enforces that the transportation of goods must rely on vehicles; M is a sufficiently large integer. Constraint (9) restricts the range of values for y_{ij}^k and z_{ij} according to their physical significance.

4. Methodology

In this section, we elaborate on our approach. We first introduce the Markov Decision Process with *Service Time Constraints* for solving the HVRP-STC problem. Subsequently, we describe the framework of our approach. Moreover, we detail the network architecture of ToDRL. Finally, we present the training policy of ToDRL.

4.1. Markov decision process with service time constraints

In this study, the route construction problem is formulated as a Markov Decision Process with *Service Time Constraints*, which converts the objective and constraints of HVRP-STC problem into a cumulative reward function and a series of meticulously defined state transition rules. Notably, empirical evidence, as reported in prior works [14,32,41], underscores that maximizing the cumulative reward of MDP is equivalent to minimizing the objective of HVRP problem.

The MDP-STC comprises four fundamental components: the state space S, the action space A, the state transition rules $\mathbb{T}$, and the reward function R, which are described as follows.

4.1.1. State space

S represents the set of all possible environment states. Each $s^t \in S$ signifies the state of the environment at time step t, which is defined as a 2-tuple (X_P^t, X_V^t).

$X_P^t = \{(\ell_{pi}, d_{pi}, o_{pi}, e_{pi}^t) \mid p \in N_P\}$ constitutes a set of node features, where ℓ_{pi}, d_{pi}, and o_{pi} symbolize the location, demand, and workload of node p_i. e_{pi}^t serves as an indicator variable: $e_{pi}^t = 1$ if node p_i has been visited before step t, and $e_{pi}^t = 0$ otherwise.

$X_V^t = \{(q_{vk}, c_{vk}, b_{vk}^t, \ell_{vk}^t, w_{vk}^t) \mid k \in N_V\}$ is a set of vehicle features at step t, where q_{vk} and c_{vk} are the speed and capacity of the vehicle v_k. b_{vk}^t, ℓ_{vk}^t, and w_{vk}^t are the number of remaining goods, location, and work time of vehicle v_k at step t respectively. Based on the aforementioned notations, the *service time* for vehicle v_k to serve node p_i is denoted as o_{pi}/q_{vk}.

4.1.2. Action space

A represents the set of all possible actions. The specific action executed at step t is denoted as $a^t \in A$. Each action $a^t = (v_J^t, p_I^t)$ assigns a node p_I^t to the route of v_J^t.

4.1.3. State transition rules

$\mathbb{T}$ transits state s^t to state s^{t+1} according to action $a^t = (v_J^t, p_I^t)$ at step t. The transition rules comprise the following four subrules:

$$e_{pi}^{t+1} = \begin{cases} 1, & \text{if } i = I, \\ e_{pi}^t, & \text{if } i \neq I, \end{cases} \tag{10}$$

$$b_{vk}^{t+1} = \begin{cases} b_{vk}^t - d_{pI}, & \text{if } k = J, \\ b_{vk}^t, & \text{if } k \neq J, \end{cases} \tag{11}$$

$$\ell_{vk}^{t+1} = \begin{cases} \ell_{pI}, & \text{if } k = J, \\ \ell_{vk}^t, & \text{if } k \neq J, \end{cases} \tag{12}$$

$$w_{vk}^{t+1} = \begin{cases} w_{vk}^t + \frac{\|\ell_{pI} - \ell_k^t\|_2}{q_{vk}}, & \text{if } k = J, \\ w_{vk}^t, & \text{if } k \neq J, \end{cases} \tag{13}$$

Subrule (10) updates the indicator variable of node p_i^t. Subrules (11)–(13) update the number of goods b_{vJ}, location ℓ_{vJ}, and work time w_{vJ} of vehicle v_J^t. It is worth mentioning that the work time includes not only travel time but also *service time* of vehicle v_J^t.

4.1.4. Reward function

The reward function $R = \sum_{t=1}^{T} r^t$ is the cumulative reward over an episode, where r^t is the immediate reward at step t. The immediate reward is defined as:

$$r^t = -\max(0, w_{vJ}^{t+1} - \max_k w_{vk}^t), \tag{14}$$

where $\max_k w_{vk}^t$ is the maximum work time among all the vehicles until step t, and w_{vJ}^{t+1} is the work time of vehicle v_J^t after performing action a_t. If $w_{vJ}^{t+1} > \max_k w_{vk}^t$, then r^t is the negative increase in maximum work time, and $r^t = 0$ otherwise. The cumulative reward is then defined as

$$\begin{aligned} R &= \sum_{t=1}^{T} -\max(0, w_{vJ}^{t+1} - \max_k w_{vk}^t) \\ &= -\max_k \sum_{i=0}^{n} \sum_{j=0}^{n} \frac{(\|\ell_{pj} - \ell_{pi}\|_2 + o_{pj}) y_{ij}^k}{q_{vk}}, \end{aligned} \tag{15}$$

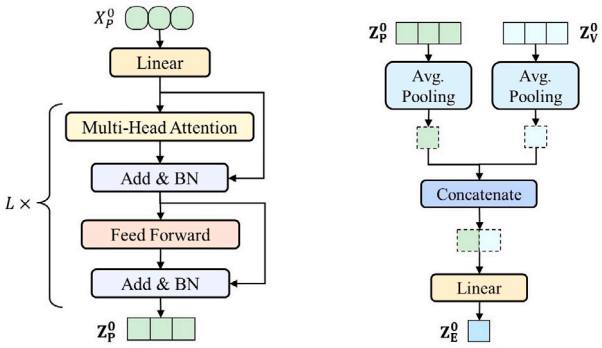
where R is the opposite of the objective in the HVRP-STC problem, encompassing both travel time and *service time*. Throughout the reinforcement learning process, the expectation of the cumulative reward R is maximized, thereby effectively minimizing the objective of the HVRP-STC problem.

4.2. Token-based deep reinforcement learning

Algorithm 1: Construct Solution

Input: $\mathcal{X} = (P, V)$: an instance of HVRP-STC problem, π_θ: ToDRL with parameters θ.
Init: Time step $t = 0$;
Environment state $s^0 = (X_P^0, X_V^0)$.
Initialize node tokens: Z_P = Node Initializer (X_P^0);
Initialize veh. tokens: Z_V^0 = Veh. Initializer (X_V^0);
Initialize env. token: Z_E^0 = Env. Initializer (Z_P, Z_V^0);
while *There are unvisited nodes* **do**
  $t = t + 1$;
  if *t equals to* 1 **then**
    $s^1 = s^0$;
  else
    Update environment state: $s^t = \mathbb{T}(s^{t-1}, a^{t-1})$;
  end
  Construct encoder context C_{en}^t;
  for *k from* 1 *to m* **do**
    Update token of v_k:
    z_{vk}^t = Veh.-Encoder k (C_{en}^t, z_{vk}^{t-1});
  end
  Update env. token:
  Z_E^t = Env.-Encoder (C_{en}^t, Z_E^{t-1});
  Construct decoder context C_{de}^t;
  Select a suitable vehicle:
  v_J^t = **Vehicle-Selector** (Z_E^t, Z_V^t);
  Select a node for that vehicle:
  p_I^t = **Node-Selector** (C_{de}^t, Z_P^t);
  Output action $a^t = (v^t, p_t)$;
end
$T = t$;
Output: Solution $\tau = \{a^1, \ldots, a^T\}$ for instance $\mathcal{X}$.

Fig. 1 depicts the framework of ToDRL, which employs a neural network to acquire a policy aimed at maximizing the cumulative reward of MDP-STC. Initially, every vehicle starts with an *empty* route comprising solely the depot node. Subsequently, other task nodes are incrementally incorporated into these routes until all task nodes have been visited.



(a) Node/Veh. Initializer (b) Env.-Initializer

Fig. 2. Structure of different initializers.

The detailed solution procedure of ToDRL is outlined in Alg. 1. Firstly, the initial state $s^0 = (X_P^0, X_V^0)$ is established according to the instance $\mathcal{X}$ of HVRP-STC problem, where X_P^0, X_V^0 represent the sets of features of nodes and vehicles, respectively. Subsequently, these features are fed into a neural network, which consists of state token coding mechanism and heterogeneous decoding mechanism. The state token encoding mechanism generates various types of tokens Z_P, Z_V^t, Z_E^t for nodes, vehicles and environment, respectively. These tokens are continuously updated throughout the solving process, ensuring that they accurately and promptly capture the latest state information of nodes, vehicles, and the environment. Leveraging these tokens, the heterogeneous decoding mechanism employs independent decoders to construct routes for different vehicles, effectively modeling the different routing patterns of heterogeneous vehicles.

In contrast to existing methods [14,32] that do not explicitly model the evolution of the environment, ToDRL incorporates a state token coding mechanism to capture the state transitions of different elements of environment, such as nodes and vehicles, throughout the solution process. This mechanism consists of a token initialization module and a token update module, which collaborate to effectively represent and update the state tokens of different elements.

4.2.1. State token initialization

The state token initialization module incorporates three distinct initializers, namely the node initializer, vehicle initializer, and environment initializer, to generate tokens specific to each corresponding element. The structure of each initializer is shown in Fig. 2.

The node initializer takes node features $X_P^0 \in \mathbb{R}^{n\times 4}$ as input and generates node tokens Z_P through a series of operations. Initially, X_P^0 is transformed into a high-dimensional feature vector H^0 using linear mapping. Let $H^0 = \{h_0^0, h_1^0, \ldots, h_n^0\} \in \mathbb{R}^{n\times d_x}$, where h_i^0 represents the high-dimensional feature of node p_i. Next, h_i^0 undergoes a series of updates using L embedding layers [32]. Each embedding layer consists of a Multi-Head Attention (MHA) sublayer and a Feed-Forward Network (FFN) sublayer. The input to the ℓth embedding layer is denoted as $H^{\ell-1} = \{h_0^{\ell-1}, h_1^{\ell-1}, \ldots, h_n^{\ell-1}\}$. The MHA sublayer processes $H^{\ell-1}$ as follows:

$$\mathrm{MHA}(H^{\ell-1}) = [\mathrm{head}_1, \mathrm{head}_2, \ldots, \mathrm{head}_h] W^O, \tag{16}$$

where $[\cdot,\cdot]$ represents matrix concatenation, $head_i$ denotes the output of the ith attention head, h is the number of attention heads, and $W^O \in \mathbb{R}^{hd_v\times d_x}$ is a trainable parameter matrix. The calculation procedure of $head_i$ is described by:

$$\begin{aligned} \mathrm{head}_i &= \mathrm{Attention}(Q_i, K_i, V_i) \\ &= \mathrm{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}}) V_i, \end{aligned} \tag{17}$$

where $Q_i = H^{\ell-1}W_i^Q, K_i = H^{\ell-1}W_i^K, V_i = H^{\ell-1}W_i^V$. $W_i^Q \in \mathbb{R}^{d_x \times d_k}, W_i^K \in \mathbb{R}^{d_x \times d_k}$, and $W_i^V \in \mathbb{R}^{d_x \times d_v}$ are parameter matrices. d_k represents the dimension of the query and key vectors, while d_v represents the dimension of the value vector. Furthermore, To improve convergence and numerical stability, the attention sub layer comes with skip connections [42] and batch normalization [43]:

$$\hat{H}^{\ell-1} = \mathrm{BN}(H^{\ell-1} + \mathrm{MHA}(H^{\ell-1})), \tag{18}$$

where $\hat{H}^{\ell-1}$ is the output of the MHA sublayer. It is then fed into the feed-forward network sublayer. This sublayer also comes with skip connection and batch normalization.

$$H^{\ell} = \mathrm{BN}(\hat{H}^{\ell-1} + \mathrm{FFN}(\hat{H}^{\ell-1})), \tag{19}$$

where $H^{\ell} = \{h_0^{\ell}, h_1^{\ell}, \ldots, h_n^{\ell}\} \in \mathbb{R}^{n \times d_x}$ is the output of the ℓth embedding layer. After passing through all the embedding layers, the final output of the node initializer is $H^L = \{h_0^L, h_1^L, \ldots, h_n^L\}$, which is also termed as Z_P.

The vehicle initializer follows a similar structure to the node initializer, with the exception of adjusting the dimension of the linear layer to accommodate the different input. It takes vehicle features X_V^0 as input and initializes the vehicle tokens Z_V^0.

The environment initializer receives the tokens of vehicles and nodes as input and initializes the environment token Z_E^0. To achieve this, the module initially reduces the dimensionality of these tokens using average pooling, which helps to reduce computational complexity. Subsequently, the initializer fuses the reduced tokens together through concatenation and linear projection to initialize the environment token Z_E^0.

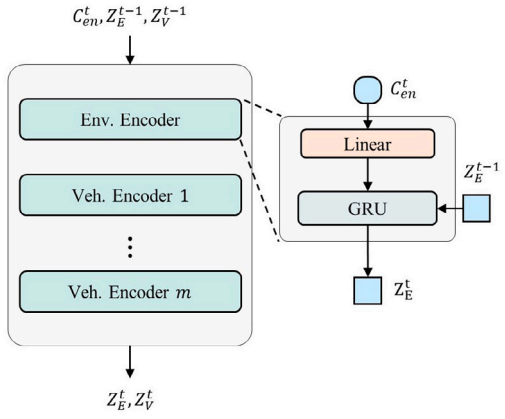
4.2.2. State token update

To manage the variable environment state throughout the route construction process and furnish dependable references for decision-making, we have devised the state token update module. This module is responsible for updating the dynamic tokens of vehicles and the environment using separate encoders. It ensures that the heterogeneous decoder has access to up-to-date and relevant information for decision-making. Notably, the dynamic information of nodes is encoded into the environment tokens, hence there is no node-encoder in this module, as node tokens remain static.

The token update module comprises an environment encoder and multiple vehicle encoders, each exhibiting a similar structure as depicted in Fig. 3(b). At each step, these encoders update the tokens of the environment and vehicles based on their most recent states, thus ensuring the timeliness and accuracy of the information. ToDRLutilize the Gated Recurrent Unit (GRU) [44] to remember the states of static elements while simultaneously updating the states of dynamic elements. This approach effectively manages the retention and forgetting of previous states and integrates new information through the utilization of reset gates and update gates, respectively.

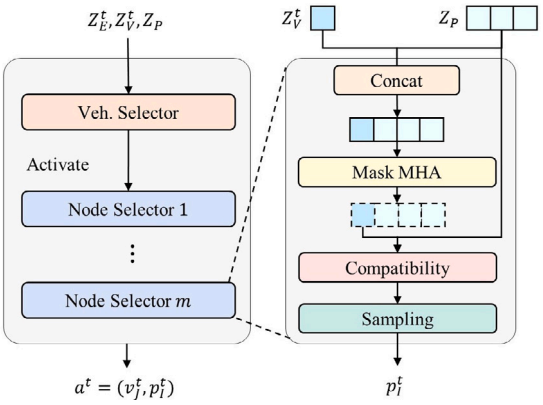
We create one GRU to update the environment token. This GRU takes the previous token $Z^{t-1}E$ as the hidden state and a linear-projected context vector as input to compute a new hidden state, which represents the updated token Z^tE. The context vector, denoted as C_{en}^t, is defined as follows: $C_{en}^t = (b_{vk}^t, \ell_{vk}^t, w_{vk}^t) \mid k \in N_V$, where b_{vk}^t, ℓ_{vk}^t, and w_{vk}^t represent the number of remaining goods, the location, and the work time of vehicle v_k at time t. It is important to note that the context vector only captures the dynamic elements' state of the environment, as the state of static elements is already embedded in the previous token Z_E^{t-1}.

We adopts m vehicle-specific encoders, where each encoder corresponds to a specific vehicle and is responsible for updating the token of that vehicle. The structure of these encoders is similar to that of the environment encoder. For the encoder associated with vehicle v_k, it takes the previous token $z^{t-1}vk$ of vk as the hidden state of the GRU. Additionally, it takes the context vector C_{en}^t as input to compute a new hidden state, which is considered as the updated token z^tvk of vehicle vk. This mechanism ensures that each vehicle's token is updated based on its previous state and the relevant dynamic information from the environment.



(a) Encoder Module. (b) Env.-Encoder.

Fig. 3. Structure of the encoder module.



(a) Heterogeneous Decoder. (b) Node-Selector v.

Fig. 4. Structure of the heterogeneous decoder.

4.3. Heterogeneous decoding

Different from Li et al. [14] that employs a composite decoder to select node for all vehicles, our heterogeneous decoding module decomposes and customizes the node selection task into a series of vehicle-specific subtasks that choose a node to visit for a specific vehicle. The module consists of a vehicle-selector and m vehicle-specific node-selectors, where each node-selector corresponds to a specific vehicle. At each step of the decoding process, the vehicle-selector takes the state tokens of the environment and vehicles as input. It comprehensively evaluates the states of each vehicle and the overall environment to determine the most suitable vehicle, denoted as v. Subsequently, the node-selector associated with vehicle v is activated to select an appropriate node, denoted as p, for vehicle v. This selection process results in the formation of an action, represented as $a = (v, p)$. Note that for the sake of brevity, the superscript and subscript of action a are omitted here (see Fig. 4).

4.3.1. Vehicle selection

In order to select a suitable vehicle based on the latest tokens, ToDRLconcatenate the token of environment Z_E^t with that of vehicles Z_V^t first and then process them by a MHA layer as Eq. (16). Let $\hat{h}_e^t$ represents the processed data corresponding to Z_E^t in the output of MHA layer. After that, A compatibility layer is employed to calculate

the compatibility between $\hat{h}_t^e$ and each vehicle token. For each vehicle v, the compatibility u_v is calculated as:

$$u_v = \begin{cases} C \cdot \tanh\left(\frac{Q^T K_v}{\sqrt{d_k}}\right), & \text{if } \text{mask}_V^t[v] = 1, \\ -\infty, & \text{otherwise}, \end{cases} \tag{20}$$

where $Q = \hat{h}_e^t W^Q$ is the query vector of the compatibility layer, $K_v = h_t^v W^K$ is the key vector of vehicle v, $mask_V^t[v]$ indicators whether vehicle v can be selected at time step t, and u_v is the compatibility between $\hat{h}_t^e$ and h_t^v. Following Kool et al. [32] and Li et al. [14], we scale the output of the hyperbolic tangent function to $[-C, C]$, where $C = 10$ in our implementation.

Since $\hat{h}_e^t$ corresponds to environment state, and h_v^t represents the state of vehicle v, u_v is the compatibility between vehicle v and environment. Further, the probabilities are calculated using the softmax function:

$$\varphi_\theta\left(v \mid s_t\right) = \frac{e^{u_v}}{\sum_j e^{u_j}}, \tag{21}$$

where $\varphi_\theta\left(v|s_t\right)$ is the probability of selecting vehicle v in state s^t, θ is the set of the network parameters. We embrace two prevalent strategies, namely greedy and sampling [14,32], for selecting a vehicle based on this distribution. The greedy strategy prioritizes local optimality by selecting the action with the highest probability at each decision point. However, it does not guarantee to find the global optimal solution and may only excel in specific scenarios where the problem adheres to the greedy-choice property and optimal substructure. As outlined in the Introduction, the heterogeneous vehicle routing problem is NP-Hard, thereby breaking the conditions required for the greedy strategy to find the global optimal [45]. On the other hand, the Sampling strategy involves randomly selecting a vehicle in accordance with the probability distribution. Although this strategy may forego local optimality, it enhances the exploration of the solution space, potentially leading to superior combinations of decisions.

4.3.2. Node selection

Considering that vehicles are heterogeneous, we establish m vehicle-specific node-selectors. node-selectors have the same structure as that of the vehicle-selector. After the vehicle-selector selects vehicle v^t, we activate the node-selector corresponding to v^t for selecting a suitable node.

For the activated node-selector, we first construct a context vector $C_{de}^t = [z_p^t, z_v^t, b_v^t]$, where z_p^t and z_v^t are tokens of the node where vehicle v is located, and vehicle v respectively. b_v^t is the number of remaining goods of vehicle v. C_{de}^t is then linear projected and concatenated with node tokens. After that, the concatenated vector is fed into the node-selector. Finally, the node-selector outputs the selected node.

4.4. Training policy

We adopt the rollout policy gradient (RPG) algorithm as the gradient estimator for the network parameters [32]. RPG comprises of two networks with an identical structure, namely the target network (π_θ) for learning and the frozen auxiliary network for stabilizing gradient. Suppose that $\mathcal{X}$ is an instance of HVRP-STC, $\tau = \{a^1, \ldots, a^T\}$ and $\hat{\tau} = \{\hat{a}^1, \ldots, \hat{a}^{T'}\}$ are the action sequences constructed for $\mathcal{X}$ by the target and the auxiliary networks, respectively. R and $\hat{R}$ are the reward by formula (15) of τ and $\hat{\tau}$ respectively. In RPG, the gradient is given by:

$$\nabla\mathcal{L}(\theta \mid \mathcal{X}) = \mathbb{E}_{\tau\sim\varphi_\theta(\tau|\mathcal{X})}\left[(\hat{R} - R)\nabla \log \varphi_\theta(\tau \mid \mathcal{X})\right], \tag{22}$$

where $\varphi_\theta(\tau \mid \mathcal{X})$ is the probability that the target network π_θ outputs the action sequence τ. $\varphi_\theta(\tau \mid \mathcal{X})$ is calculated as:

$$\varphi_\theta(\tau \mid \mathcal{X}) = \prod_{t=0}^{T} \varphi_\theta(p_I^t \mid s^t, v_J^t)\varphi_\theta(v_J^t \mid s^t), \tag{23}$$

where $\varphi_\theta(p_I^t \mid s^t, v_J^t)$ and $\varphi_\theta(v_J^t \mid s^t)$ are the output of the vehicle selector and node selector. Detailed derivation can be found in [32]. We then use the Adam optimizer for optimization. Specifically, the learning rate is set to 1×10^{-4} and decays by 0.995 per epoch. we also adopt a 'warm-up' stage in the first training epoch and clip the norm of all gradient vectors to be within 3.0 as DRL [14].

Algorithm 2: Learning of ToDRL

Input: number of epochs N_e, batch size N_b, number of iterations per epoch N_i, learning rate β, significance of the paired t-test α;
Init: parameters θ for the target network π_θ, parameters $\hat{\theta}$ for the baseline network $\pi_{\hat{\theta}}$;

```
Generate the validation dataset D_v;
for epoch from 1 to N_e do
    Generate the training dataset D_t;
    for iter from 1 to N_i do
        Retrieve N_b instances {X_1, ..., X_{N_b}} from D_t;
        for i from 1 to N_b do
            τ_i = Construction Solution (X_i, π_θ) according to Alg. 1;
            τ̂_i = Construction Solution (X_i, π_θ̂) according to Alg. 1;
            Estimate gradient ∇L(θ | X_i) by Eq. (22);
        end
        ∇L = (1/N_b) Σ_{i=1}^{N_b} ∇L(θ | X_i)
        θ ← Adam(β, θ, ∇L);
    end
    if OneSidedPairedT-Test(θ, θ̂) < α then
        θ̂ ← θ;
    end
end
```

Output: A trained model with parameters θ

The detailed training policy is shown in Algorithm 2. Firstly, we set the hyperparameters, such as the number of epochs N_e, batch size N_b, number of iterations N_i, etc, and initialize parameters of target network π_θ and baseline network $\pi_{\hat{\theta}}$ randomly. Then, we generate a validation set, which is used to evaluate the performance of the two networks. The training process consists of N_e epochs, each epoch contains N_i iterations. At each iteration, we construct solutions for each instance and estimate the gradients of θ according to Eq. (22). After that, Adam algorithm [46] is employed to optimize the parameters based on their gradients.

After each epoch, the performance of π_θ and $\pi_{\hat{\theta}}$ is evaluated on the validation dataset. The parameters of $\pi_{\hat{\theta}}$ will be replaced by those of π_θ if π_θ outperforms $\pi_{\hat{\theta}}$ according to a one sided paired T-Test with significance level α.

5. Experiment

In this section, we bring our experiments on four types of datasets. Primarily, we elucidate our experimental settings, including evaluation metrics, baselines, and implementation details. Subsequently, we furnish the results of comparison experiments between ToDRL and nine prevailing methods. Further, we show the results of robustness experiments on two datasets containing HVRP-STC instances of different sizes and large spatial coverage. We also tested the performance of ToDRL under uncertainty. Finally, we take MVRP as an example to demonstrate the generalization ability of ToDRL.

5.1. Experimental settings

5.1.1. Evaluation metrics

Three popular performance metrics namely *the Average value of Objective* (AO), *the GAP between the method under consideration and the*

best known method so far in terms of AO (GAP), and *the running time* (Time) are used for evaluation as in [14,32,41]. Specifically, AO is defined as follows:

$$\mathrm{AO} = \frac{\sum_{i=1}^{K} O_i}{K}. \tag{24}$$

Notation K is the number of instances used for test. i is the index of an instance. O_i is the objective value of the solution for ith instance. The gap between a method and the best method is represented by GAP, which is defined as

$$\mathrm{GAP} = \frac{\mathrm{AO} - \mathrm{AO}_{min}}{\mathrm{AO}_{min}}, \tag{25}$$

where AO_{min} is the AO of the best known method so far. The running time a method takes to solve all instances used in test is denoted by Time.

5.1.2. Baselines

To verify the superiority of our approach, we implement **nine** baselines, including five no-DRL heuristics, two solver-based methods, and two DRL-based methods, which include: (1) Genetic Algorithm (GA), an evolutionary computation-based heuristic. (2) Ant Colony Optimization (ACO), a swarm intelligence-based heuristic. (3) Adaptive Large Neighborhood Search (ALNS), a single solution-based heuristic. (4) Slack Induction by String Removals (SISR) [47], a ruin and recreate approach for solving CVRP. (5) Hybrid Genetic Search (HGS) [48], one of the state-of-the-art no-DRL methods for solving CVRP and its variants. (6) Gurobi [25], a solver for mixed integer programming (MIP). We employ it as a baseline because HVRP-STC problem belongs to MIP. (7) ORTools [26], an open source combinatorial optimization solver developed by Google AI. (8) AM [32], an attention-based reinforcement learning method for solving the routing problems. Following Li et al. [14], we adopt it to solve HVRP-STC problem. (9) DRL [14], the **state-of-the-art** deep reinforce learning-based method for solving the heterogeneous vehicle routing problem.

For GA, ACO, and ALNS, we utilize publicly available implementations of them[1] for solving the HVRP-STC. The hyperparameters of these heuristics are optimized using random search [49], which are shown in Table 2. Given that SISR and HGS were initially developed for the CVRP, we have customized their implementations to accommodate the Heterogeneous Vehicle Routing Problem (HVRP). This adaptation has been carried out leveraging their open-source implementations[2,3] as the foundation. We limit the maximum computation time for Gurobi and ORTools of solving an HVRP-STC instance. Otherwise, their execution can become excessively time-consuming. For AM and DRL, we keep the hyperparameters in their source code[4,5] without any changes.

5.1.3. Implementation details

To facilitate comparison with existing methods and avoid the influence of tuning hyperparameters, we maintain the training hyperparameters of ToDRL *consistent with* those of DRL [14], as detailed in Table 2. Specifically, we employ the Adam optimizer for training the neural network. The learning rate is set to 1×10^{-4} and decays by a factor of 0.995 per epoch. we also incorporate a 'warm-up' phase during the first training epoch and clip the norm of all gradient vectors to be within 3.0 to enhance the training stability.

Regarding the network structure, we designate the number of embedding layers L in the initializer module as 3. The dimension of the state vectors of nodes and vehicles are 128. Considering that the environment token needs to represent the state of all vehicles and nodes, we set its dimension to 512 to enhance its representation capability. The number of hidden neurons in the feed-forward layers is 512, while each multi-head attention layer comprises eight attention heads.

To minimize the influence of computing platform on the experimental results, all methods are implemented in Python, and all experiments are executed on the same server with an Intel Xeon E5-2699 v4 @ 2.20 GHz CPU, 128G RAM, and one single server with 10 TITAN XP GPUs.

Table 2
Hyperparameters of baselines and ToDRL.

Method	Hyperparameters
GA	$P_c = 0.12, P_m = 0.96, N_{select} = 44,\ N_{population} = 50, iter = 1800.$
ACO	$\alpha = 3, \beta = 2, Q = 120.95, rho = 0.64,\ tau = 7.08, m = 20, iter = 1800.$
ALNS	$P_{min} = 0.09, P_{max} = 0.12, N^{min}_{worst} = 4, N^{max}_{worst} = 19, N_{regret} = 4, r_1 = 20.19, r_2 = 22.38, r_3 = 36.50, rho = 0.52, phi = 0.32, pu = 2, iter = it(size)$[a], $it(20) = 250, it(50) = 45, it(100) = 15.$
SISR	$T_{init} = 100, T_{final} = 1, C_{bar} = 10, L_{max} = 10, M_{\alpha} = 0.01, iter = it(size)$[a], $it(20) = 1e5, it(50) = 6e4, it(100) = 3.2e4$
HGS	$nbGranular = 20, mu = 25, lambda = 40, nbElite = 4, nbClose = 5, TimeLimit = TL(size)$[b], $TL(20) = 60$ s, $TL(50) = 120$ s, $TL(100) = 240$ s
Gurobi	$TimeLimit = TL(size)$[b], $TL(20) = 60$ s, $TL(50) = 120$ s, $TL(100) = 240$ s
ORTools	$TimeLimit = TL(size)$[b], $TL(20) = 60$ s, $TL(50) = 120$ s, $TL(100) = 240$ s
AM	$lr = 10^{-4}, lr\_decay = 1.0, \alpha = 0.05, \beta_{exp} = 0.8, L = 3, tanh\_clipping = 10, max\_grad\_norm = 1, warmup\_epochs = 1,\ embedding\_dim = 128, FFN\_hidden\_dim = 512$
DRL	$lr = 10^{-4}, lr\_decay = 0.995, \alpha = 0.05, \beta_{exp} = 0.8, L = 3, tanh\_clipping = 10, max\_grad\_norm = 3, warmup\_epochs = 1,\ embedding\_dim = 128, FFN\_hidden\_dim = 512$
ToDRL[c]	$lr = 10^{-4}, lr\_decay = 0.995, \alpha = 0.05, \beta_{exp} = 0.8, L = 3, tanh\_clipping = 10, max\_grad\_norm = 3, warmup\_epochs = 1,\ embedding\_dim = 128, FFN\_hidden\_dim = 512$

[a] The number of iterations of ALNS and SISR decrease as the size of instance increases.
[b] The time limit of Gurobi, ORTools and HGS increase as the size of instance increases. This is to make the time consumption of those heuristics similar, so as to compare their solution quality fairly.
[c] Hyperparameters of ToDRL are exactly the same as that of DRL.

5.2. Comparison analysis

5.2.1. Datasets

Similar to Kool et al. [32] and Li et al. [14], we generate several datasets according to the following rules: coordinates of the nodes ℓ_{pi} are randomly sampled from a two-dimensional uniform distribution defined on $[0, 1] \times [0, 1]$. The demand of the node d_i is an integer belonging to $\{1, 2, \ldots, 9\}$, which is also randomly sampled. $o_i = d_i \times 0.1$ is the workload required to service node i. The demand and workload of the depot node are set to 0. Then, We establish three fleets, named V3, V5, and V10. They consists of 3, 5, and 10 heterogeneous vehicles, respectively. Details of them are shown below Table 3. Further, based on the above data generation rules and fleets, we generate a dataset, which consists of three subset named V3-N20, V5-N50, and V10-N100. instances of them contain fleet V3 and 20 task nodes, fleet V5 and 50 task nodes, fleet V10 and 100 task nodes respectively.

Each subset consists of three parts: training, validation, and test. The training part has 1,280,000 instances (except for V10-N100, which has 640 000 instances). The validation part has 10,000 instance. The test set has 1280 instances. They are generated with random seeds 1234 and 4321 respectively, to avoid overfitting.

[1] https://github.com/yangchb/Algorithms_for_solving_VRP.
[2] https://github.com/chkwon/PyHygese.
[3] https://github.com/chenmingxiang110/tsp_solver.
[4] https://github.com/wouterkool/attention-learn-to-route.
[5] https://github.com/Demon0312/HCVRP_DRL.

Table 3
Results of comparison experiments.

Method	V3-N20			V5-N50			V10-N100		
	AO	GAP (%)	Time	AO	GAP (%)	Time	AO	GAP (%)	Time
GA	10.02	16.24	20.62 h	16.32	40.57	46.22 h	17.63	55.33	83.97 h
ACO	9.21	6.84	20.30 h	13.64	17.48	55.04 h	14.15	24.66	125.89 h
ALNS	9.58	11.14	25.96 h	15.89	36.86	49.64 h	17.82	57.00	103.71 h
SISR	10.45	21.22	20.27 h	16.64	43.32	43.40 h	18.55	63.43	83.97 h
HGS	10.71	24.24	23.96 h	15.16	30.57	47.74 h	15.20	33.92	97.98 h
Gurobi	8.77	1.74	21.33 h	13.61	17.23	42.67 h	36.87	224.85	87.58 h
ORTools	12.98	50.58	21.33 h	76.65	560.21	42.67 h	153.83	1255.33	94.73 h
AM (Greedy)	9.37	8.70	0.64 s	13.47	16.02	**0.92 s**	13.19	16.21	**2.62 s**
DRL (Greedy)	9.10	5.57	**0.62 s**	12.26	5.60	1.74 s	12.31	8.46	2.91 s
ToDRL (Greedy)	8.88	3.02	0.71 s	11.95	2.93	2.20 s	11.72	3.26	3.04 s
AM (Sampling)	8.84	2.55	206.92 s	12.50	7.67	331.76 s	12.24	7.84	1367.98 s
DRL (Sampling)	8.70	0.93	214.59 s	11.72	0.95	747.37 s	11.66	2.73	2307.10 s
ToDRL (Sampling)	**8.62**	**0.00**	329.22 s	**11.61**	**0.00**	934.67 s	**11.35**	**0.00**	2001.11 s

Fleet V3 comprises three heterogeneous vehicles with capacities of 10, 24, 40, and speeds of 1.0, 0.75, 0.5, respectively.
Fleet V5 comprises five heterogeneous vehicles with capacities of 10, 16, 24, 34, 40, and speeds of 1.0, 0.85, 0.75, 0.6, 0.5, respectively.
Fleet V10 comprises ten heterogeneous vehicles with capacities of 10, 14, 16, 20, 24, 26, 30, 34, 36, 40, and speeds of 1.0, 0.95, 0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5, respectively.

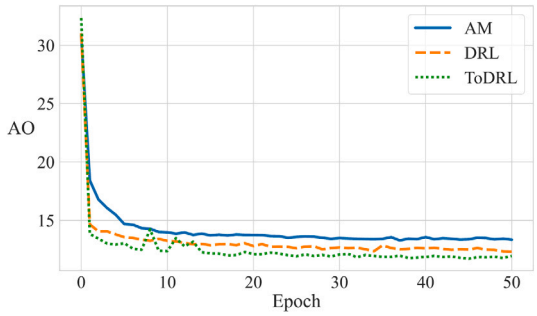


Fig. 5. The variation of AO with epoch during training on the V10-N100 dataset.

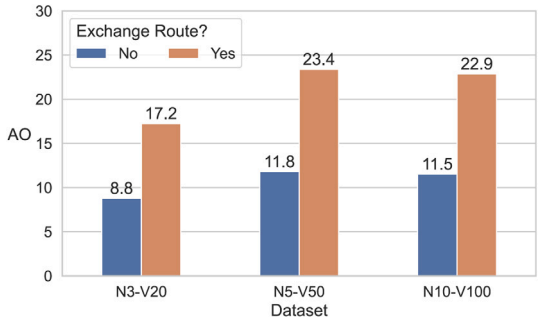


Fig. 6. Average of objective with and without exchanging routes.

5.2.2. Training and test

Heuristics and solver-based methods do not require training and can be tested directly.

Reinforcement learning methods, including AM, DRL, and ToDRL are trained for 50 epochs. Each epoch contains 2500 iterations with batch size set to 512. (Except for the instances with 100 customer nodes, on which we set the training batch size to 256 due to limited GPU memory.) The performances of these methods are evaluated on the validation set after every epoch. The model that performs best on the validation set is used for test.

In the same level with [14,32], each training epoch of ToDRL consumes an average of 26.55 min, 59.20 min, and 102.76 min on V3-N20, V5-N50, and V10-N100 respectively with one single TITAN XP GPU. Fig. 5 shows the variation of AO with Epoch for AM, DRL and ToDRL when training on the V10-N100 dataset. ToDRL significantly outperforms drl for most of the training time.

We test their performance under two decoding strategies, greedy and sampling, respectively. Greedy decoding greedily selects the action with the maximum probability at each step, and finally generates a solution. Sampling decoding randomly selects an action according to the probability distribution. This action may not be the best at present, but may improve the quality of future decisions. Similar to Kool et al. [32] and Li et al. [14], when using sampling decoding, we sample 1280 solutions for each instance and select the best one as the final solution.

5.2.3. Results

Table 3 shows the detailed comparison experiment results, which are summarized as follows:

- In terms of AO, i.e. the quality of solutions, ToDRL consistently and significantly outperforms all baselines on all datasets. The superiority of ToDRL becomes more obvious as the fleet size and number of tasks increase, which shows the effectiveness of ToDRL for instances with strong heterogeneity and large state space.
- In terms of Time, ToDRL is slightly worse than AM and DRL, because the network structure of ToDRL is more complicated. In addition, AM, DRL and ToDRL all outperform heuristics and solver-based baselines significantly, which demonstrates the advantage of learning-based methods.
- Compared with greedy decoding, sampling decoding improves the quality of the solution, but also significantly increases the time consumption. In addition, the worse method obtains a more significant improvement, e.g., on V3-C20, it reduces the AO of AM by 0.53, but only reduces that of ToDRL by 0.26. If we only consider greedy decoding, the advantage of ToDRL becomes more evident.

5.2.4. Exchange route experiment

Fig. 7 shows the routes constructed by ToDRL for heterogeneous vehicles. The specific configuration of each vehicle is shown by the footnote of Table 3. It can be preliminarily seen that heterogeneous vehicles have different behavior patterns. In order to further clarify the impact of heterogeneous behavior patterns on the optimization objective, we conducted an exchange route experiment. This experiment focuses on the first and last vehicles within the fleets V3, V5, and V10, while disregarding the other vehicles. During the testing phase, we computed the time required for the two aforementioned vehicles to fulfill their tasks under standard conditions and subsequent to the exchange of paths, denoted as AO. The experimental results are shown in Fig. 6.

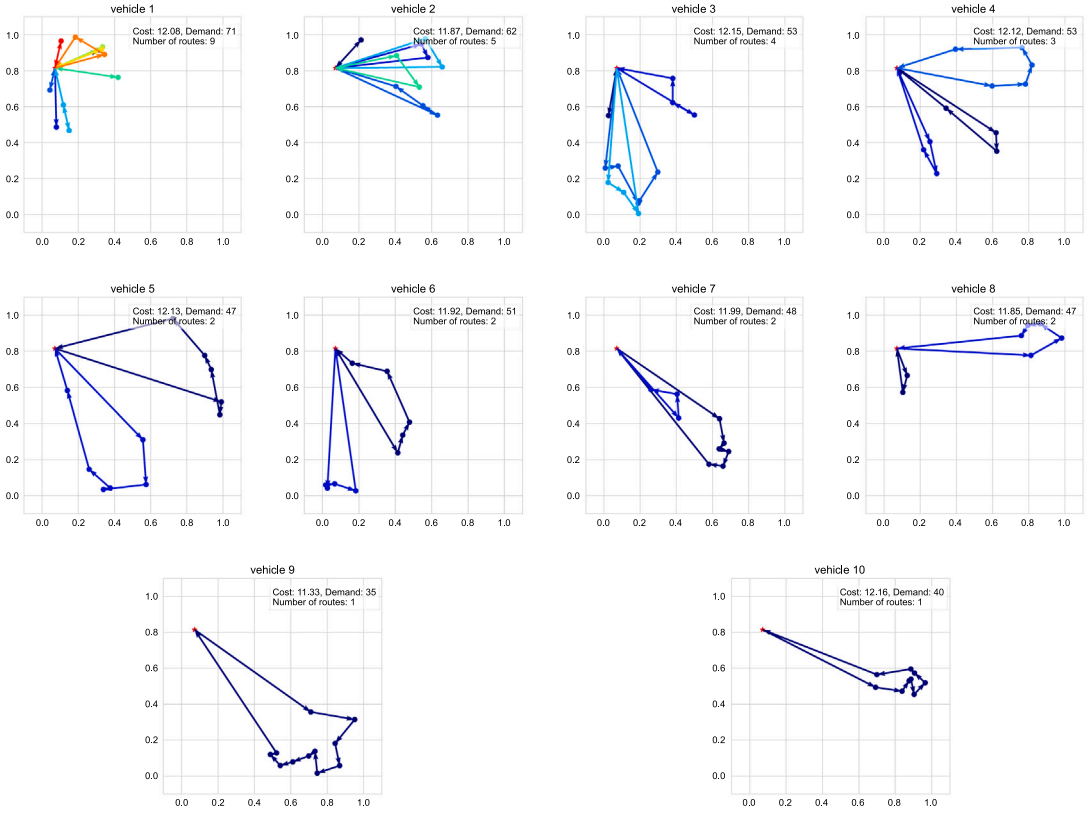


Fig. 7. This figure shows the routes constructed by ToDRL for an instance of HVRP-STC comprising 10 heterogeneous vehicles (see footnote of Table 3 for detailed configuration of each vehicle) and 100 task nodes. Each subfigure shows the route of an individual vehicle. The upper right section of each subfigure presents the time consumption and goods demand of the corresponding vehicle. Despite the diverse behavior patterns of the heterogeneous vehicles, ToDRL reasonably constructs routes for each vehicle, ensuring their operational duration remains approximately 12 units, consequently minimizing the objective of HVRP-STC.

According to the experimental results, it is evident that AO nearly doubles following the exchange of routes between two heterogeneous vehicles. This observation underscores that scheduling inappropriate routes for heterogeneous vehicles will make a disastrous effect on AO. Correspondingly, adequately modeling the routing patterns of heterogeneous vehicles is beneficial to minimize AO.

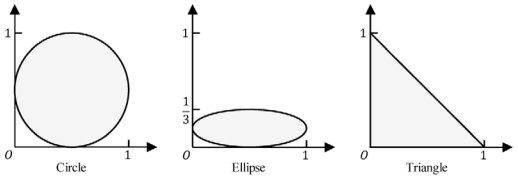


Fig. 8. The distribution area of different dataset.

5.3. Robustness analysis

In practical scenarios, the spatial distribution and number of tasks are diverse, so it is impossible to train a specific model for every case. Therefore, the model should be robust to changes in the spatial distribution and number of tasks. In this section, we test the robustness of ToDRL to these changes.

5.3.1. Datasets

1. To test the robustness of ToDRL for instances with varied number of tasks, we generate a dataset, which consists of three subsets named V5-N150, V5-N250, V5-N500. They all contain 1280 instances with fleet V5, but the number of tasks are 150, 250, and 500, respectively.
2. To test the robustness of ToDRL for varied spatial distributions, we generate a dataset, which consists of three subsets named V5-N50-Circle, V5-N50-Ellipse, V5-N50-Triangle. They all have 1280 instances. Each instance contains Fleet V5 and 50 tasks. But their tasks are evenly distributed in three different spatial shapes, as shown in Fig. 8.

5.3.2. Training and test

We train the model on V5-N50 and test it on the above six datasets. All hyperparameters are consistent with the comparison experiments, except for the batch size when testing on V5-N500, which we changed from 1280 to 640 due to limited GPU memory.

5.3.3. Results

Table 4 shows the results of the robustness experiment, which are summarized as follows:

- In terms of AO, ToDRL outperforms AM and DRL in all datasets, which demonstrates its advantage in robustness. On the other hand, as the number of tasks increases, the performance of AM gradually approaches ToDRL. This is due to the very simple structure of AM, so it is more robust to changes in the number of tasks.
- In terms of Time, ToDRL is slightly worse than AM, but is better than DRL on large-scale instances, such as V5-N250 and V5-N500, because DRL needs to construct a complex *context*, which is quite time-consuming when facing large-scale instances.

Table 4
Results of robustness experiments.

Method	V5-N150			V5-N250			V5-N500		
	AO	GAP (%)	Time	AO	GAP (%)	Time	AO	GAP (%)	Time
AM	37.66	2.76	**1.70 s**	61.83	0.90	**2.95 s**	121.51	0.17	**10.01 s**
DRL	37.32	1.83	2.82 s	61.74	0.75	5.46 s	123.02	1.42	22.01 s
ToDRL	**36.65**	**0.00**	2.94 s	**61.28**	**0.00**	5.32 s	**121.30**	**0.00**	17.49 s
Method	V5-N50-Circle			V5-N50-Ellipse			V5-N50-Triangle		
	AO	GAP (%)	Time	AO	GAP (%)	Time	AO	GAP (%)	Time
AM	12.72	12.37	**0.69 s**	11.41	8.98	**0.66 s**	12.31	11.30	**0.74 s**
DRL	11.65	2.92	1.40 s	10.91	4.20	0.98 s	11.43	3.35	1.54 s
ToDRL	**11.32**	**0.00**	1.31 s	**10.47**	**0.00**	1.23 s	**11.06**	**0.00**	1.13 s

Table 5
Results of uncertainty experiments.

Method	V3-N20-R			V5-N50-R			V10-N100-R		
	AO	GAP (%)	Time	AO	GAP (%)	Time	AO	GAP (%)	Time
AM	10.48	2.54	0.48	16.00	8.55	**0.60**	15.98	7.10	**1.15**
DRL	10.32	0.98	**0.43**	15.18	2.99	0.67	15.27	2.35	1.24
ToDRL	**10.22**	**0.00**	0.54	**14.74**	**0.00**	0.91	**14.92**	**0.00**	1.70

Table 6
Results of generalization experiments.

Method	V3-N20-MTSP			V5-N50-MTSP			V10-N100-MTSP		
	AO	GAP (%)	Time	AO	GAP (%)	Time	AO	GAP (%)	Time
AM	3.08	17.11	**0.48 s**	3.61	33.70	**0.60 s**	3.54	36.68	1.77 s
DRL	2.67	1.52	0.56 s	2.75	1.85	0.83 s	2.59	3.60	**1.37 s**
ToDRL	**2.63**	**0.00**	1.00 s	**2.70**	**0.00**	2.18 s	**2.50**	**0.00**	2.62 s

5.4. Uncertainty analysis

Considering that in practical applications, the estimated demand d_{pi} of node p_i may deviate from the real demand $\hat{d_{pi}}$. The solver can only plan routes based on the estimated value d_{pi}; the real value $\hat{d_{pi}}$ can only be obtained when the vehicle arrives at node p_i. The routes planned for a vehicle based on the estimated demand may not satisfy the capacity constraint of the HVRP-STC problem in real situations, resulting in the vehicle to change its routes. To test the performance of the ToDRLin this case, uncertainty tests are conducted in this paper.

5.4.1. Datasets

In the uncertainty experiment, we created three datasets named V3-N20-R, V5-N50-R and V10-N100-R. They all contain 1280 instances. The estimated demand d_{pi} is randomly selected from the integer set $\{1, 2, \ldots, 9\}$, and the real value $\hat{d_{pi}} = d_{pi} \times \omega$, where $\omega \sim \mathcal{N}(1, 0.3^2)$. The other attribute generation rules of the instances are consistent with the comparison experiments.

5.4.2. Training and test

Instead of retraining the model, we directly use the pre-trained model in the comparison experiment to perform inference directly on the uncertain experiment data set.

5.4.3. Results

The experimental results are shown in Table 5. The solution quality of ToDRLis significantly better than that of AM and DRL. As the problem size increases, the advantage of ToDRLover other methods is gradually expanded. This proves that the present method can still give high-quality routing solutions in uncertainty scenarios.

5.5. Generalization analysis

Although ToDRL is designed to handle the heterogeneous fleets and dynamic states of HVRP-STC problem, it can generalize to other routing problems with these two traits. To demonstrate this, we conduct experiment with another well-known combinatorial optimization problem - the Multiple Traveling Salesman Problem (MTSP) [50].

5.5.1. Datasets

We generate a dataset, which consists of three subsets named V3-N20-MTSP, V5-N50-MTSP, and V10-N100-MTSP, each with 1280 MTSP instances. The coordinates of the nodes l_i in these instances are randomly sampled from a two-dimensional uniform distribution defined on $[0, 1] \times [0, 1]$.

5.5.2. Training and test

All hyperparameters and settings are consistent with the comparative experiments of HVRP-STC problem, except that the dimensions of some linear mapping layers are slightly changed due to changes in the input data.

In the same level with Li et al. [14], each training epoch of ToDRL consumes an average of 13.59 min, 35.20 min, and 73.06 min, on V3-N20-MTSP, V5-N50-MTSP, and V10-N100-MTSP respectively with one single TITAN XP GPU.

5.5.3. Results

Table 6 shows the results of the generalization experiment. With comparable computation time, the quality of our solution is significantly and consistently better than that of AM and DRL. Our advantage gradually expands as the instance size and fleet heterogeneity increases. The experimental results of MTSP problem remain consistent with that of HVRP-STC problem, thus demonstrating the generalization ability of ToDRL on other routing problems.

6. Conclusion

In this paper, we propose a deep reinforcement learning based approach, ToDRL, for solving the challenging yet practical Heterogeneous Vehicle Routing Problem with *Service Time*. The most important features of ToDRL are the comprehensive environment coding and the heterogeneous decoder mechanisms, which respond to specific requirements of HVRP-STC problem well. We provide extensive experiments to validate the proposed method. In terms of solution quality, ToDRL consistently outperforms by a large margin **nine** baselines including

heuristics, solver-based, and learning-based methods, with a moderate solution time close to learning-based methods and much shorter than that of heuristics and solver-based methods. The superiority of ToDRL in running time becomes more obvious as the size of fleet and tasks increase. Moreover, we have also successfully applied ToDRL to the multiple-TSP problem. Thus it is reasonable to study in future work how ToDRL can be generalized to other challenging routing problems like dynamic VRP.

CRediT authorship contribution statement

Yujun Wang: Writing – original draft, Software, Methodology. **Xiaopeng Hong:** Writing – review & editing, Supervision, Project administration, Methodology, Conceptualization. **Yabin Wang:** Visualization, Software, Investigation. **Junzhou Zhao:** Writing – review & editing, Resources. **Guanghui Sun:** Writing – review & editing, Visualization. **Baoxing Qin:** Writing – review & editing, Investigation, Data curation.

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

Data will be made available on request.

Acknowledgments

This work is funded by the National Key Research and Development Project of China (2019YFB1312000), the National Natural Science Foundation of China (62076195, 62376070), and the Fundamental Research Funds for the Central Universities (AUGA5710011522).

References

[1] G.B. Dantzig, J.H. Ramser, The truck dispatching problem, Manag. Sci. 6 (1) (1959) 80–91.
[2] P. Toth, D. Vigo, Vehicle Routing: Problems, Methods, and Applications, SIAM, 2014.
[3] P. Deng, G. Amirjamshidi, M. Roorda, A vehicle routing problem with movement synchronization of drones, sidewalk robots, or foot-walkers, Transp. Res. Procedia 46 (2020) 29–36.
[4] Y.-H. Jia, Y. Mei, M. Zhang, A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem, IEEE Trans. Cybern. (2021).
[5] A. Anderluh, P.C. Nolz, V.C. Hemmelmayr, T.G. Crainic, Multi-objective optimization of a two-echelon vehicle routing problem with vehicle synchronization and 'grey zone'customers arising in urban logistics, European J. Oper. Res. 289 (3) (2021) 940–958.
[6] R. Baldacci, A. Mingozzi, A unified exact method for solving different classes of vehicle routing problems, Math. Program. 120 (2) (2009) 347–380.
[7] A. Pessoa, R. Sadykov, E. Uchoa, F. Vanderbeck, A generic exact solver for vehicle routing and related problems, Math. Program. 183 (1) (2020) 483–523.
[8] H. Zhou, H. Qin, C. Cheng, L.-M. Rousseau, An exact algorithm for the two-echelon vehicle routing problem with drones, Transp. Res. B 168 (2023) 124–150.
[9] F.Y. Vincent, P. Jewpanya, A.P. Redi, Y.-C. Tsao, Adaptive neighborhood simulated annealing for the heterogeneous fleet vehicle routing problem with multiple cross-docks, Comput. Oper. Res. 129 (2021) 105205.
[10] S. Bansal, S. Wadhawan, A hybrid of sine cosine and particle swarm optimization (HSPS) for solving heterogeneous fixed fleet vehicle routing problem, Int. J. Appl. Metaheuristic Comput. (IJAMC) 12 (1) (2021) 41–65.
[11] E. Yağmur, S.E. Kesen, Multi-trip heterogeneous vehicle routing problem coordinated with production scheduling: Memetic algorithm and simulated annealing approaches, Comput. Ind. Eng. 161 (2021) 107649.
[12] P. Toth, D. Vigo, The Vehicle Routing Problem, SIAM, 2002.
[13] J.M. Vera, A.G. Abad, Deep reinforcement learning for routing a heterogeneous fleet of vehicles, in: 2019 IEEE Latin American Conference on Computational Intelligence, LA-CCI, IEEE, 2019, pp. 1–6.
[14] J. Li, Y. Ma, R. Gao, Z. Cao, A. Lim, W. Song, J. Zhang, Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem, IEEE Trans. Cybern. (2021).
[15] W. Qin, Z. Zhuang, Z. Huang, H. Huang, A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem, Comput. Ind. Eng. 156 (2021) 107252.
[16] D.S. Lai, O.C. Demirag, J.M. Leung, A tabu search heuristic for the heterogeneous vehicle routing problem on a multigraph, Transp. Res. E 86 (2016) 32–52.
[17] S. Mancini, A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic, Transp. Res. C 70 (2016) 100–112.
[18] J. Wang, A.K.R. Jagannathan, X. Zuo, C.C. Murray, Two-layer simulated annealing and tabu search heuristics for a vehicle routing problem with cross docks and split deliveries, Comput. Ind. Eng. 112 (2017) 84–98.
[19] L. Simeonova, N. Wassan, S. Salhi, G. Nagy, The heterogeneous fleet vehicle routing problem with light loads and overtime: Formulation and population variable neighbourhood search with adaptive memory, Expert Syst. Appl. 114 (2018) 183–195.
[20] Y. Meliani, Y. Hani, S.L. Elhaq, A. El Mhamedi, A tabu search based approach for the heterogeneous fleet vehicle routing problem with three-dimensional loading constraints, Appl. Soft Comput. 126 (2022) 109239.
[21] A. Baniamerian, M. Bashiri, R. Tavakkoli-Moghaddam, Modified variable neighborhood search and genetic algorithm for profitable heterogeneous vehicle routing problem with cross-docking, Appl. Soft Comput. 75 (2019) 441–460.
[22] K. Sethanan, T. Jamrus, Hybrid differential evolution algorithm and genetic operator for multi-trip vehicle routing problem with backhauls and heterogeneous fleet in the beverage logistics industry, Comput. Ind. Eng. 146 (2020) 106571.
[23] S. Voigt, M. Frank, P. Fontaine, H. Kuhn, Hybrid adaptive large neighborhood search for vehicle routing problems with depot location decisions, Comput. Oper. Res. 146 (2022) 105856.
[24] J.C. Molina, J.L. Salmeron, I. Eguia, An ACS-based memetic algorithm for the heterogeneous vehicle routing problem with time windows, Expert Syst. Appl. 157 (2020) 113379.
[25] Gurobi, Gurobi Optimizer. https://www.gurobi.com/products/gurobi-optimizer/.
[26] Google, Google OR-Tools. https://developers.google.com/optimization.
[27] IBM, CPLEX optimizer. https://www.ibm.com/analytics/cplex-optimizer.
[28] N. Mazyavkina, S. Sviridov, S. Ivanov, E. Burnaev, Reinforcement learning for combinatorial optimization: A survey, Comput. Oper. Res. 134 (2021) 105400.
[29] M. Noaeen, A. Naik, L. Goodman, J. Crebo, T. Abrar, Z.S.H. Abad, A.L. Bazzan, B. Far, Reinforcement learning in urban network traffic signal control: A systematic literature review, Expert Syst. Appl. (2022) 116830.
[30] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, in: International Conference on Learning Representations, 2017.
[31] M. Nazari, A. Oroojlooy, M. Takáč, L.V. Snyder, Reinforcement learning for solving the vehicle routing problem, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, 2018, pp. 9861–9871.
[32] W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems!, in: International Conference on Learning Representations, 2018.
[33] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, Adv. Neural Inf. Process. Syst. 28 (2015) 2692–2700.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[35] H. Liao, Q. Dong, X. Dong, W. Zhang, W. Zhang, W. Qi, E. Fallon, L.B. Kara, Attention routing: track-assignment detailed routing using attention-based reinforcement learning, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 84003, American Society of Mechanical Engineers, 2020, V11AT11A002.
[36] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, S. Min, POMO: Policy optimization with multiple optima for reinforcement learning, 2020, arXiv preprint arXiv:2010.16011.
[37] J. Bi, Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun, Y.M. Chee, Learning generalizable models for vehicle routing problems via knowledge distillation, 2022, arXiv preprint arXiv:2210.07686.
[38] A. Hottung, K. Tierney, Neural large neighborhood search for the capacitated vehicle routing problem, in: ECAI 2020, IOS Press, 2020, pp. 443–450.
[39] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[40] M. Kim, J. Park, J. Park, Learning to CROSS exchange to solve min-max vehicle routing problems, in: The Eleventh International Conference on Learning Representations, 2023.
[41] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, C. Zhang, Reinforcement learning with multiple relational attention for solving vehicle routing problems, IEEE Trans. Cybern. (2021).
[42] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[43] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, PMLR, 2015, pp. 448–456.

[44] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014, arXiv preprint arXiv:1412.3555.
[45] H. Huang, S. Yang, X. Li, Z. Hao, An embedded Hamiltonian graph-guided heuristic algorithm for two-echelon vehicle routing problem, IEEE Trans. Cybern. 52 (7) (2021) 5695–5707.
[46] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[47] J. Christiaens, G. Vanden Berghe, Slack induction by string removals for vehicle routing problems, Transp. Sci. 54 (2) (2020) 417–433.
[48] T. Vidal, Hybrid genetic search for the CVRP: Open-source implementation and swap* neighborhood, Comput. Oper. Res. 140 (2022) 105643.
[49] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2) (2012).
[50] T. Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures, Omega 34 (3) (2006) 209–219.