Neural airport ground handling

Yaoxin WU
Eindhoven University of Technology

Jianan ZHOU
Nanyang Technological University

Yunwen XIA

Xianli ZHANG

Zhiguang CAO
Singapore Management University, zgcao@smu.edu.sg

See next page for additional authors

Citation

WU, Yaoxin; ZHOU, Jianan; XIA, Yunwen; ZHANG, Xianli; CAO, Zhiguang; and ZHANG, Jie. Neural airport ground handling. (2023). *IEEE Transactions on Intelligent Transportation Systems*. 24, (12), 15652-15666.
Available at: https://ink.library.smu.edu.sg/sis_research/8069

Author

Yaoxin WU, Jianan ZHOU, Yunwen XIA, Xianli ZHANG, Zhiguang CAO, and Jie ZHANG

Neural Airport Ground Handling

Yaoxin Wu*, Jianan Zhou*, Yunwen Xia, Xianli Zhang, Zhiguang Cao, Jie Zhang

Abstract—Airport ground handling (AGH) offers necessary operations to flights during their turnarounds and is of great importance to the efficiency of airport management and the economics of aviation. Such a problem involves the interplay among the operations that leads to NP-hard problems with complex constraints. Hence, existing methods for AGH are usually designed with massive domain knowledge but still fail to yield high-quality solutions efficiently. In this paper, we aim to enhance the solution quality and computation efficiency for solving AGH. Particularly, we first model AGH as a multiple-fleet vehicle routing problem (VRP) with miscellaneous constraints including precedence, time windows, and capacity. Then we propose a construction framework that decomposes AGH into sub-problems (i.e., VRPs) in fleets and present a neural method to construct the routing solutions to these sub-problems. In specific, we resort to deep learning and parameterize the construction heuristic policy with an attention-based neural network trained with reinforcement learning, which is shared across all sub-problems. Extensive experiments demonstrate that our method significantly outperforms classic meta-heuristics, construction heuristics and the specialized methods for AGH. Besides, we empirically verify that our neural method generalizes well to instances with large numbers of flights or varying parameters, and can be readily adapted to solve real-time AGH with stochastic flight arrivals. Our code is publicly available at: https://github.com/RoyalSkye/AGH.

Index Terms—Airport Ground Handling, Vehicle Routing Problem, Attention Model, Reinforcement Learning.

I. INTRODUCTION

Efficient turnaround operations for aircraft play an important role in alleviating the flight delays and relevant economic loss at an airport. In *airport ground handling* (AGH), a variety of operations need to be scheduled to serve the flights according to their precedence relation (as shown in Fig. 1), where each type of operation is always performed by a fleet of vehicles. Therefore, central to AGH is usually the vehicle routing problem (VRP). However, different from other conventional VRPs, the one in AGH is much harder given that, 1) apart from the intricate precedence constraint, each (type of) flight has its own requirement (e.g., demand, time windows) for each operation; 2) the problem scale is

Yaoxin Wu is with the Department of Information Systems, Faculty of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, Netherlands. (E-mail: wyxacc@hotmail.com).

Jianan Zhou, Yunwen Xia, Xianli Zhang and Jie Zhang are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore (E-mails: jianan004@e.ntu.edu.sg, yunwen001@e.ntu.edu.sg, xianli001@e.ntu.edu.sg, zhangj@ntu.edu.sg).

Zhiguang Cao is with the School of Computing and Information Systems, Singapore Management University, Singapore. (E-mail: zhiguangcao@outlook.com).
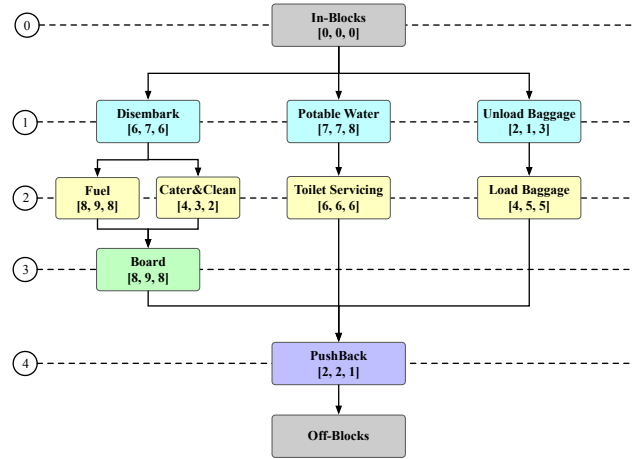

Fig. 1: The exemplary operations in AGH and their precedence relation. The operations at the same level share the same priority, and a smaller level index means the corresponding operation has a higher priority, e.g., fueling, catering and cleaning have the same priority and precede boarding. Moreover, the numbers in the square brackets refers to the corresponding operation duration to three different flight types.

often large in practice especially for busy airports where a considerable number of flights need to be served at each time. Hence, solving the VRP in AGH is nontrivial and challenging (Note: hereafter, we use AGH to represent the VRP in AGH for convenience).

Considering the combinatorial nature of the problem, exact algorithms based on branch and bound (or its variants) have been exploited to solve AGH. They usually model the problem as mixed integer linear programming (MILP) [1], [2], and then solve it with mature solvers, e.g., CPLEX. However, such solvers are inefficient to handle the large and complex solution space, and usually consume prohibitively long computation time. On the other hand, as alternatives to balance the solution quality and computation time, traditional improvement heuristics or meta-heuristics also have been leveraged to solve AGH. In this line of works, a number of attempts design specific heuristics only for a single type of operation, e.g., towing, catering, fueling and baggage loading, respectively [3]–[6]. However, the problem could be much harder when multiple fleets for various operations are practically needed given their complex precedence relation. To handle this challenging setting, some works proposed specialized meta-heuristics to solve AGH with multiple operations [7], [8]. Nevertheless, these methods depend on much trial-and-error and domain knowledge to design hand-crafted rules or operators, which may limit the solving performance and hinder their applications in reality.

Recently, an increasingly large number of deep models have

been proposed to learn construction heuristics for solving VRP variants including traveling salesman problem (TSP) and capacitated vehicle routing problem (CVRP). Most of those deep models follow the encoder-decoder architecture and are trained with reinforcement learning (RL) algorithms [9]–[13]. The learned construction policies infer solutions efficiently. More importantly, those end-to-end neural methods could produce desirable results which sometimes are even comparable to the highly specialized and optimized heuristics. Despite much success achieved, most of the neural methods are only able to tackle the classical VRP variants with much simple constraints, which will become less effective in coping with AGH.

In this paper, we propose a neural method that learns to construct solutions for AGH. Particularly, we first propose a general framework to decompose the global optimization of AGH into sub-problems for each fleet of vehicles based on the precedence of operations, so that a construction heuristic could be applied to solve them in sequence. However, those sub-problems may interplay each other due to the precedence relation (e.g., fueling goes before boarding) at each flight. To address this issue, we present an attention-based policy network with masking schemes for constraints, where we explicitly encode time windows and operations in our neural network to capture the interplay. Moreover, we train the neural network with RL, which is shared across all the sub-problems. The resulting learnt policy is able to focus on long-term return and attain solutions of high-quality. Accordingly, our contributions are summarized as follows,

1) We design a construction framework to decompose the AGH into sub-problems, which allows them to be sequentially solved with the proposed construction heuristic.
2) We parameterize the proposed construction heuristic with an attention-based policy network in conjunction with embeddings of time windows and operations. Further trained with RL and shared across sub-problems, it allows the learnt heuristic to automatically construct solutions for AGH without much hand-crafted rules. To our knowledge, this is one of the early neural methods to tackle VRPs with complex constraints.
3) We conduct extensive experiments with a realistic setting from the CHANGI Airport in Singapore. Results show that the proposed method efficiently produces high-quality solutions and outperforms the state-of-the-art conventional meta-heuristics. Particularly, our method can solve AGH with hundreds of flights in seconds, and is applicable to dynamic scenarios with stochastic arrivals.

The remainder of this paper is organized as follows. Section II reviews existing works related to AGH and neural methods for VRPs. Section III introduces AGH and its MILP formulation. Section IV elaborates the construction framework for AGH and the proposed neural method. Section V presents experimental results and analysis, and we conclude the paper in Section VI.

II. RELATED WORKS

In this section, we first review the existing works pertaining to airport ground handling (AGH), and then provide a view of neural methods for general vehicle routing problems (VRPs).

A. Airport Ground Handling

Most existing works on AGH only tackle a specific (type of) operation, and model it as a vehicle routing problem (VRP) or related variants. Among them, Norin et al. formulated the scheduling of de-icing operation as a vehicle routing problem with time windows (VRPTW), and minimized both the tour length of de-icing vehicles and the total flight delay uisng the greedy randomized adaptive search procedure (GRASP) algorithm [14], [15]. Du et al. modeled the scheduling of towing operation also as a VRPTW and solved it using the column generation method based on the MILP form of the problem [3]. Zhou et al. studied the scheduling of trailers considering both rolling windows and flight delays, which was solved using the genetic algorithm (GA) [16]. Guo et al. considered scheduling vehicles for airport baggage transport, and solved it by leveraging the parameter or rule selection in GA [6]. Han et al. described the scheduling of ferry vehicles as the form of MILP to minimize the number of used vehicles, which was solved using CPLEX [17]. There were also a number of studies for scheduling of fueling operation that were solved with meta-heuristics like ant colony optimization (ACO) and iterated local search (ILS) [5], [18], [19]. On the other hand, only a few works investigated the global optimization for multiple (types of) operations in AGH. Padrón et al. presented a decomposition method to schedule multiple interplayed operations based on constraint programming (CP) and large neighborhood search (LNS) [7]. In particular, they first optimized the time windows for vehicle fleets using CP, then the decomposed VRPTW for each fleet were solved using LNS, which aimed to decrease the waiting time for operations and the total completion time for serving the flights. Moreover, they also ameliorated the method by solving the decomposed VRPTW in different orders, which impacted the performance due to the interplay relation among operations [20]. A recent method leveraged non-dominated sorting genetic algorithm (NSGA-II) to solve the routing problems for multiple types of vehicles (fleets), with respective VRPTW modeled for each fleet [8]. It aimed to minimize the total number of used vehicles as well as time cost of specified vehicles, where the presented method outperformed the multiobjective evolutionary algorithm based on decomposition (MOEAD) and the particle swarm optimization (PSO) algorithm. Other works less related to AGH in airside operation research and on-demand logistics, e.g., gate assignment, manpower scheduling, truck-drone coordinated VRPs, are introduced in [21]–[27].

B. Neural Methods for VRPs

The advances of deep (reinforcement) learning have ignited considerable research on solving combinatorial optimization problems (COPs) with neural networks. The attempts in recent years have shown encouraging results in tackling various classic COPs [28]–[33], where vehicle routing problems (VRPs) are receiving noticeably more attention than others [34]–[39]. Most of the deep models for VRPs follow the encoder-decoder architecture, where the encoder learns representations of problem instances, and the decoder constructs the solution (i.e., tours) sequentially. Among them, Vinyals et al. designed

a RNN based sequence-to-sequence (seq2seq) neural architecture to solve TSP using supervised learning [40]. Bello et al. improved the training algorithm by changing it to reinforcement learning, which circumvents the needs of optimal solutions as the labels [9]. Moreover, they also introduced the masking scheme to guarantee feasible solutions, and attention glimpses to strengthen the network efficiency. Nazari et al. further transformed the encoder in the seq2seq model into the linear projection, and applied it to solve both TSP and CVRP [10].

Rather than RNN, more subsequent works resorted to Transformer to rebuild seq2seq models using attention-based neural networks [41]. Among them, Kool et al. adapted the attention modules (AM) in both encoder and decoder of Transformer to solve TSP, CVRP and other variants [11]. Concurrently, Kaempfer et al. tailored the permutation invariant pooling layer in Transformer to seq2seq models and solved the multiple TSP [42]. However, its performance was inferior to the AM in [11]. On top of AM, Kwon et al. further enhanced the performance on both TSP and CVRP by breaking the symmetry that lies in the essentially same routes [31]. Notably, many other variants of AM were also proposed recently given its desirable framework and convenience for implementation, e.g., the ones consider multiple relational attention [43] and multiple decoders [44]. In addition, a number of endeavours focused on the graph representation for VRP. Khalil et al. described traditional greedy heuristics for TSP as MDP and trained a deep Q-Network (DQN) based policy network which embedded the states in episodes using Graph Neural Network (GNN) [12], [45]. Nowak et al. attempted to predict the solution to the graph based TSP using GNN and supervised learning [46]. Joshi et al. proposed a graph convolutional network (GCN) with different decoding schemes to tackle TSP in both supervised or autoregressive fashion [47]. We would like to note that most of the above works fall into the category of neural *construction* methods, which construct solutions (tours) for VRP sequentially (e.g., node by node). For another category of related works which learn *improvement* heuristics or meta-heuristics, we refer the readers to surveys [48]–[50]

Although the aforementioned neural methods could automatically learn the rules to solve VRPs, most of them only handle problems with simple settings and constraints (e.g. TSP and CVRP). In contrast, this paper turns to solve a complex yet practical VRP with intricate constraints (i.e. AGH with multiple operations), and we propose an effective neural method to solve the problem by learning from data in an end-to-end fashion. To the best of our knowledge, it is a very early attempt to successfully solve the AGH with deep learning based method.

III. PROBLEM STATEMENT

In this section, we describe the AGH with multiple (types of) operations as a multiple-fleet vehicle routing problem with various constraints and formulate it as the form of mixed integer linear programming (MILP).

In practice, each (type of) operation for the flight is normally performed by a fleet of vehicles. Therefore, we model the AGH

with multiple operations as a multiple-fleet VRP, additionally constrained by *capacity*, *time windows* and *precedence*. Specifically, we represent the AGH on an undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with the node set $\mathcal{N} = \{0, 1, \cdots, n, \dot{n}\}$ and edge set $\mathcal{E} = \{(i,j)|i,j \in \mathcal{N}; i \neq j\}$. Both node 0 and $\dot{n}$ denote the depot but are used to differentiate the location which the vehicles start from and return to, so as to avoid temporal conflicts at the depot. We denote flights to be served as $\mathcal{N}^* = \mathcal{N} \setminus \{0, \dot{n}\}$ with demand δ_i^f for each $i \in \mathcal{N}^*$ and operation $f \in \mathcal{F}$ ($\mathcal{F} = \{1, \cdots, F\}$). Naturally, we set $\delta_0^f = \delta_{\dot{n}}^f = 0$. For an operation $f \in \mathcal{F}$, each edge (i,j) is assigned with a cost c_{ij}^f, with $c_{0\dot{n}}^f = +\infty$ to avoid the meaningless travel between node 0 and $\dot{n}$. In this paper, since we aim to minimize the global travel distance of all vehicles in all operations, we set $c_{ij}^1 = \cdots = c_{ij}^F$, all of which equal to the distance between flight i and j. For different operations, the precedence relation is represented as $f_1 \prec f_2$ ($f_1, f_2 \in \mathcal{F}$) if f_1 precedes f_2. We assume that each fleet[1] that corresponds to $f \in \mathcal{F}$ comprises vehicles $\mathcal{V}^f = \{1, \cdots, V^f\}$ with the capacity denoted by Q^f and sufficiently large V^f. In addition, we denote by d_i^f the service time for flight i regarding operation f, which means how much time it takes to complete operation f at flight i, with $d_0^f = d_{\dot{n}}^f = 0$; we denote by t_{ij}^f the travel time from flight i to j by the vehicle for operation f, with $t_{0\dot{n}}^f = 0$; we denote by $[a_i^f, b_i^f]$ the time window to serve flight i, which means the start time of operation f for serving flight i should be between a_i^f and b_i^f. Accordingly, the AGH is formulated as follows,

$$\text{min.} \quad \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}^f} \sum_{(i,j) \in \mathcal{E}} c_{ij}^f x_{ijv}^f \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{N}} \sum_{v \in \mathcal{V}^f} x_{ijv}^f = 1, \forall j \in \mathcal{N}^*, f \in \mathcal{F} \tag{2}$$

$$\sum_{i \in \mathcal{N} \setminus \{\dot{n}\}} x_{iuv}^f = \sum_{j \in \mathcal{N} \setminus \{0\}} x_{ujv}^f, \forall u \in \mathcal{N}^*, v \in \mathcal{V}^f, f \in \mathcal{F} \tag{3}$$

$$\sum_{j \in \mathcal{N}^*} \sum_{v \in \mathcal{V}^f} x_{0jv}^f \leq V^f, \forall f \in \mathcal{F} \tag{4}$$

$$\sum_{j \in \mathcal{N}^*} \sum_{v \in \mathcal{V}^f} x_{0jv}^f = \sum_{i \in \mathcal{N}^*} \sum_{v \in \mathcal{V}^f} x_{i\dot{n}v}^f, \forall f \in \mathcal{F} \tag{5}$$

$$\sum_{i \in \mathcal{N} \setminus \{0\}} \sum_{v \in \mathcal{V}^f} x_{i0v}^f = \sum_{j \in \mathcal{N} \setminus \{\dot{n}\}} \sum_{v \in \mathcal{V}^f} x_{\dot{n}jv}^f = 0, \forall f \in \mathcal{F} \tag{6}$$

$$\sum_{i \in \mathcal{N}^*} \delta_i^f \sum_{j \in \mathcal{N}} x_{ijv}^f \leq Q^f, \forall v \in \mathcal{V}^f, f \in \mathcal{F} \tag{7}$$

$$x_{ijv}^f (T_{iv}^f + d_i^f + t_{ij}^f - T_{jv}^f) \leq 0, \forall v \in \mathcal{V}^f, f \in \mathcal{F} \tag{8}$$

$$a_i^f \leq T_{iv}^f \leq b_i^f, \forall v \in \mathcal{V}^f, f \in \mathcal{F} \tag{9}$$

$$T_{iv}^{f_1} + d_i^{f_1} \leq T_{iv}^{f_2}, \forall f_1, f_2 \in \mathcal{F}, f_1 \prec f_2 \tag{10}$$

$$x_{ijv}^f \in \{0, 1\}, \forall (i,j) \in \mathcal{E}, v \in \mathcal{V}^f, f \in \mathcal{F} \tag{11}$$

$$T_{iv}^f \geq 0, \forall i \in \mathcal{N}^*, v \in \mathcal{V}^f, f \in \mathcal{F} \tag{12}$$

[1]Note that fleets naturally share the same indices as operations since they are performed by the corresponding fleets. We also assume that the vehicles are homogeneous.

where the decision variable x_{ijv}^f determines whether a vehicle v for operation f serves flight j after i, and T_{iv}^f determines when to start serving flight i by vehicle v for operation f. Regarding the objective in Eq. (1), minimizing the global travel distance as the cost is considered in this paper. Regarding the constraints, Eq. (2) ensures that each flight is only served by one vehicle for a given operation; Eq. (3) ensures that a vehicle goes to serve a flight for an operation and will exactly leave the same fight; Eq. (4) ensures that the number of vehicles to serve flights for an operation is not larger than the maximum number of vehicles in the fleet; Eq. (5) ensures that the vehicles in a fleet which leave the depot will return to it; Eq. (6) ensures that all routes start from and end at the depot; Eq. (7) ensures that the demands fulfilled by a vehicle is not larger than its capacity; Eq. (8) ensures the right temporal logic when the vehicle v for operation f serves two continuous flights i and j. Specifically, the start time of its service at flight j should not be earlier than the time point when the vehicle completes the service at flight i and then arrives at flight j. Eq. (9) ensures that the start time of operation f at flight i should be within the time window defined above. Eq. (8) and (9) together ensure that operations always start within the predefined time windows, which also obey the temporal sequence; Eq. (10) ensures that the operations are performed to a flight following their precedence relation, where the exemplary precedence is depicted in Fig. 1. For example, the operation f_2 can only start at flight i after that the operation f_1 is completed at the same flight, if f_1 precedes f_2. We further linearize Eq. (8) as below,

$$T_{iv}^f + d_i^f + t_{ij}^f - T_{jv}^f \leq C(1 - x_{ijv}^f), \forall (i,j) \in E, f \in \mathcal{F}, \quad (13)$$

where C is a large constant (i.e. 10^6), so that the studied AGH could be transformed into a mixed integer linear programming (MILP) problem.

Please note that this paper targets a more complex yet practical version of AGH in comparison to most existing literature, where multiple operations, hundreds of aircraft, and various relations (e.g. precedence and temporal relations) between operations and aircraft are considered. In the following section, we propose a learning based method to solve the AGH which we have modeled as a multiple-fleet VRP with diverse constraints. With the power of deep learning, our method can learn more effective heuristics and cost less domain knowledge and hand-engineering than the existing methods for AGH [7], [8], [20]. As we know, this is the first time that deep learning is applied to solve this practical VRP, rather than simple and standard VRPs in the literature such as TSP and CVRP.

IV. METHODOLOGY

In this section, we introduce the proposed method to solve AGH. Firstly, we present a framework that is able to work elegantly with construction heuristics. It decomposes the global optimization of AGH into VRPs for each fleet based on the precedence relation, and sequentially constructs solutions (routes) to each resulting VRP which will also be used to update the time window of flights for the succeeding level of operations. Then we formulate the solution construction

for each sub-problem as a Markov decision process (MDP), and parameterize the policy with an attention-based neural network, which is shared across all the sub-problems. Finally, we present the RL algorithm to train the policy network. The overview of our proposed method is illustrated in Fig. 2.

A. Construction Framework for AGH

The proposed framework for AGH is generic to construction heuristic such as Clarke and Wright's Savings (CWS), insertion based heuristics or learning based policies, which are used to solve the VRP for each fleet by selecting next flight to serve progressively. In specific, we decompose the AGH problem into sub-problems for each fleet. The sub-problems are essentially VRPs with capacity and time window constraints, which can be further grouped based on their precedence level (priorities). In doing so, we can solve the AGH by optimizing sub-problems following the precedence relation, which reduces the complexity and ensures the feasibility of solutions to the original problem. The *decomposition strategy* and the *overview of construction framework* is briefly summarized as follows (see also Fig. 2):

1) Firstly, we decompose all fleets into several groups based on their precedence level. The fleets in each group should have the same precedence level, which means the time windows of the served flights for the current fleet will not be affected by other fleets in the same group. For example, the disembarkation, portable water supply and baggage unloading in Fig. 1 are in the same group, which can be processed independently. In this way, the global AGH problem is decomposed into several groups, with each group comprising several sub-problems (fleets).

2) Then we start with the group who has the highest precedence level. All sub-problems in the current group are solved by a solver or construction heuristic in the framework, sequentially or in parallel (i.e., the order doesn't matter since they can be processed independently). We solve the sub-problems in each group sequentially.

3) Based on the constructed solutions for the current group, we update the time windows of flights for fleets in the succeeding group whose precedence level is the most nearest to (and lower than) the current one. For example, the completion time of fueling, catering and cleaning will be used to update the time window of boarding in Fig. 1. In other words, the time windows for operations at flights are gradually determined by solving the sub-problems following the precedence relation.

4) After solving all the groups, we obtain the global solution by simply collecting the solution to each sub-problem. It includes the derived routes for vehicles in each fleet, and the start time of operations can be easily computed based on the routes and operation duration. The global solution is feasible since all constraints are satisfied in the solving process. For example, the precedence constraint is satisfied since we solve sub-problems following the precedence level, and update the time windows accordingly; the capacity and time windows are satisfied when solving sub-problems by masking scheme as shown in Eqs. (27) and (28) in Section IV-C.
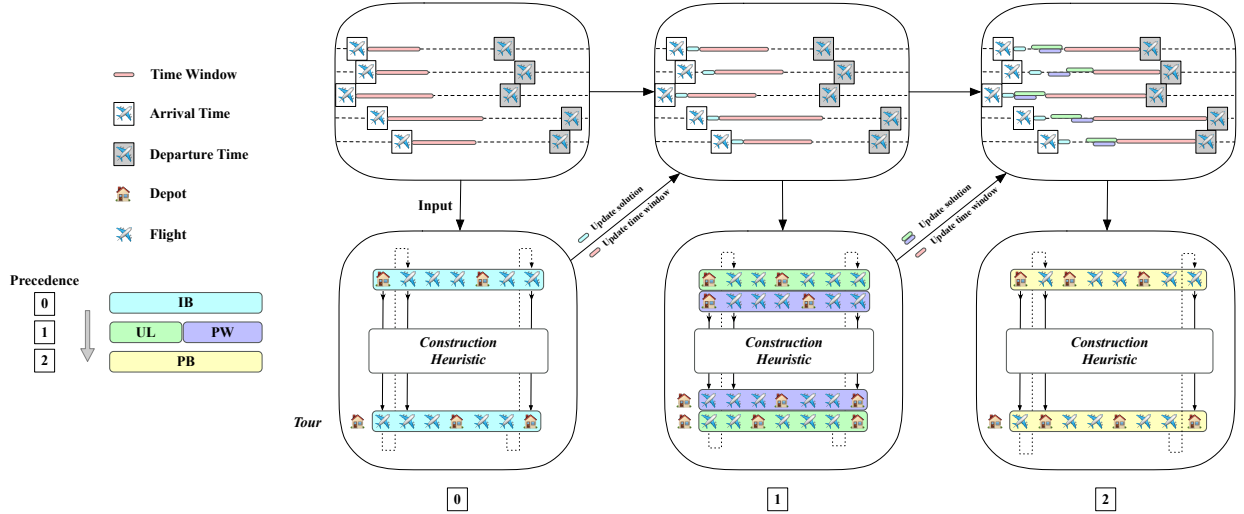
Fig. 2: An illustration of the proposed construction framework for AGH. In this exemplary instance, five flights need to be served by four fleets of vehicles for corresponding operations with three precedence levels (priorities). Based on the precedence relation, we decompose this AGH into four sub-problems (or three groups). For each sub-problem, a local time window (red) of each flight is maintained, which constrains the start and completion of the current operation. The construction heuristic produces solutions for sub-problems, which are also used to update the time windows for VRPs in the next precedence level.

Next, we formally introduce the details of the proposed construction framework for AGH. For a set of operations $s(p)$ with precedence levels $p \in [0, ..., \mathcal{P} - 1]$, we maintain a time window $tw_j^p = [a_j^p, b_j^p]$ for each flight j, where a_j^p and b_j^p represent the start and end of the time window, respectively. The operations in the precedence level p must be launched and completed within the assigned time window. Note that the time window of each flight is the same even for different operations if they have the same precedence level, such as fueling and catering in Fig. 1. Formally, the time window for operation f at flight j is constrained with the precedence level p (i.e., $f \in s(p)$) and updated as follows,

$$a_j^p = \begin{cases} t_a^j, & p = 0 \\ \max_{f \in s(p-1)} T_j^f, & p \neq 0 \end{cases} \tag{14}$$

$$b_j^p = t_d^j - \sum_{p \prec p'} \max_{f \in s(p')} d_j^f, \tag{15}$$

where t_a^j and t_d^j are the planned arrival and departure time of flight j; T_j^f is the completion time of operation f at flight j, which is calculated based on the constructed solutions for operations of the preceding precedence level; d_j^f is the service duration of operation f at flight j. Intuitively, the start of time window at a flight is initialized with the planned arrival time and updated based on the completion time of all operations with higher priorities. On the other hand, the end of time window is initialized with the departure time and progressively tighten based on the least reserved time required by the set of operations with lower priorities. The max operator in Eq. (15) results from the Cannikin Law[2], which means the least reserved time required for set $s(p')$ is determined by the

---

[2] Also known as *Wooden Bucket Theory*, where the capacity of a bucket is determined by the shortest stave.

operation with the longest duration, since it guarantees that all operations in $s(p')$ could be completed within $\max_{f \in s(p')} d_j^f$ in the optimal case.

As illustrated in Fig. 2, we deal with the precedence constraints and assign time windows to each flight following the aforementioned process in Eqs. (14) and (15). The other constraints in VRPs for each fleet are handled by the construction heuristic. The derived solution for each fleet by the construction heuristic will also be used to update time windows of flights for operations in the succeeding precedence level. In this way, AGH is solved by constructing VRP solutions for fleets sequentially which will satisfy all constraints. Note that the operations with the same precedence level (e.g., fueling and catering in Fig. 1) can be performed for the flight at the same time, and their solutions can be constructed in parallel with shared inputs (i.e., the same assigned time windows) to the construction heuristic.

B. MDP Formulation

According to the construction framework in Section IV-A, the AGH problem x can be decomposed into sub-problems x^f, where f refers to an operation (or fleet). These sub-problems can be sequentially solved by any construction heuristic following the precedence relation as mentioned in Section IV-A. Instead of classic construction heuristics, this paper aims to use DRL to learn construction policies to solve the sub-problems. To this end, for each sub-problem x^f, we formulate the solution construction as a Markov decision process (MDP), i.e., $(\mathcal{S}, \mathcal{N}, \mathcal{P}_a, \mathcal{R}_a)$. Specifically, the *agent* repeatedly selects the next flight to serve with the learnt policy, which results in a finite trajectory/episode. In particular, at the tth step,

- *State:* $s_t^f \in S$ is represented by the static (i.e., the graph embedding) and dynamic information (i.e., the context

embedding and previously selected flights).

- *Action:* $a_t^f \in \mathcal{N}$ is to select the next flight to serve w.r.t. the current operation according to the policy $\pi(a_t^f|s_t^f)$, where $\mathcal{N}$ is the node set $\{0, 1, \dots, n\}$ with 0 denoting the depot and others for flight indices.

- *Transition:* $\mathcal{P}_a = P(s_{t+1}^f|s_t^f, a_t^f)$ refers to the transition from state s_t^f to s_{t+1}^f, which is resulted from the action a_t^f. In our framework, the next state is realized deterministically w.r.t. the action, i.e., $\mathcal{P}_a = 1$.

- *Reward:* After all flights are served within T steps, the episode in MDP for operation f ends, and a complete solution (tour) $a^f = (a_1^f, \dots, a_T^f)$ is attained. The total reward for the episode is represented as the negative length of the tour, i.e., $\mathcal{R}_a = -L(a^f)$.

C. Policy Parameterization

We concretize the policy as an attention-based encoder-decoder model inspired by the attention model (AM) in [11], where the encoder learns representations of problem instances, and the decoder constructs solutions by learning which flight to select next for the current operation. Note that the learnt policy is shared across all the operations (sub-problems). Given a sub-problem x^f for instance x, which is a VRP w.r.t. operation f, the probability of a solution $\pi_\theta(a^f|x^f)$ is parameterized by θ and factorized as follows,

$$\pi_\theta(a^f|x^f) = \prod_{t=1}^{T} \pi_\theta(a_t^f|x^f, a_{1:t-1}^f), \qquad (16)$$

where $a_{1:t-1}^f$ represents the current partial solution (route) that has been constructed before the tth iteration. At the tth iteration, we fix the previously constructed route $a_{1:t-1}^f$, and select an unvisited yet feasible flight (or depot) according to the policy π_θ, which will be added into the current route. However, different from AM, our model handles much more complex VRP variants than the ones in [11]. In particular, the architecture of our model is illustrated in Fig. 3, which primarily encompasses the encoder and decoder,

1) Encoder: The encoder first computes the initial embedding $\mathbf{h}_j^{(0)}$ for each node (i.e., flight and depot) with dimension $d_h = 128$. On the one hand, we consider the location information of nodes. Different from the existing works where the coordinates of nodes in TSP or CVRP are randomly sampled according to a specific distribution, the flight nodes in AGH are located at some gates that are fixed in an airport. In other words, each flight is assigned to a specific gate for operations, and all possible locations of flights could be enumerated. Therefore, we leverage a learnable lookup table (i.e., a list of embeddings) to represent the location of each node. On the other hand, for each flight, we embed their demands and time windows by learning a linear projection. Then, the initial embeddings are attained by adding them together. Formally, the initial embeddings for depot and flight nodes are expressed as follows,

$$\mathbf{h}_j^{(0)} = \begin{cases} \mathbf{e}_0, & j = 0 \\ \mathbf{e}_{g_j} + \mathbf{W}[\delta_j^f, a_j^p, b_j^p] + \mathbf{b}, & j = 1, \dots, n \end{cases} \qquad (17)$$

where $\mathbf{e}_{g_j}$ is the embedding of the location g_j for flight j; δ_j^f is the demand of flight j for operation f; a_j^p and b_j^p are the start and end of time window of operation $f \in s(p)$ on flight j, respectively; $\mathbf{W} \in \mathbb{R}^{d_h \times 3}$ and $\mathbf{b} \in \mathbb{R}^{d_h \times 1}$ are trainable parameters; $[\cdot, \cdot]$ is the horizontal concatenation operator. Note that since the time windows for the current operation are updated according to the solutions to the preceding operations, it is natural to leverage recurrent neural network (RNN) or its variants to specifically embed a_j^p and b_j^p. However, it did not bring significant improvements over the linear projection w.r.t. the final global solutions. Hence, we stick to the usage of linear projection in Eq. (17) for better parallelism and higher computation efficiency (we discuss such a comparison in Section V-B).

Given the yielded initial embeddings, the encoder updates them for $N=3$ times with the multi-head attention (MHA) ($M=8$ heads) layer and feed-forward (FF) layer, which further process their outputs using skip-connection and batch normalization (BN). Specifically, the update of embeddings at the the lth layer (Eq. (18) and (19)), and the forward pass of a MHA layer and FF layer (Eq. (20)-(22)) are expressed as follows,

$$\hat{\mathbf{h}}_j^{(l)} = \text{BN}^{(l)}(\mathbf{h}_j^{(l-1)} + \text{MHA}^{(l)}(\mathbf{h}_j^{(l-1)}; \mathbf{h}_0^{(l-1)}, \dots, \mathbf{h}_n^{(l-1)})), \qquad (18)$$

$$\mathbf{h}_j^{(l)} = \text{BN}^{(l)}(\hat{\mathbf{h}}_j^{(l)} + \text{FF}^{(l)}(\hat{\mathbf{h}}_j^{(l)})), \qquad (19)$$

$$u_{jk}^m = \frac{(W_Q^m \mathbf{h}_j)^T (W_K^m \mathbf{h}_k)}{\sqrt{d_h/M}}, \qquad (20)$$

$$\hat{\mathbf{h}}_j = \text{MHA}(\mathbf{h}_j; \mathbf{h}_0, \dots, \mathbf{h}_n) = \sum_{m=1}^{M} W_O^m [\sum_{k=0}^{n} \frac{e^{u_{jk}}}{\sum_{k'} e^{u_{jk'}}} W_V^m \mathbf{h}_k], \qquad (21)$$

$$\text{FF}(\hat{\mathbf{h}}_j) = W_1^F \cdot \text{ReLu}(W_0^F \hat{\mathbf{h}}_j + b_0^F) + b_1^F, \qquad (22)$$

where $l \in [1, \dots, N]$ is the layer index and $\mathbf{h}_j^{(l)}$ is the embedding for flight j at the lth layer; BN refers to the batch normalization; $(\mathbf{h}_0, \dots, \mathbf{h}_n)$ are embeddings related to the depot and flight nodes; W_Q^m, W_K^m and W_V^m are trainable parameters to attain the query, key and value vectors in MHA; W_O^m is the trainable parameters for linear projection in the mth head; W_0^F, W_1^F and b_0^F, b_1^F are trainable parameters for linear projection and bias in FF layer. The resulting advanced embeddings $\mathbf{h}_j^{(N)}$ for nodes are then passed to the decoder to help decide the next flight to serve.

2) Decoder: Given the embeddings from the encoder and the current partial solution $a_{1:t-1}^f$, the decoder sequentially selects a flight a_t^f at time step t to construct the VRP solution regarding each operation. To well represent the state s_t^f, we incorporate the context embedding $\mathbf{h}_c^{(N)}$ into decoder to capture the dynamics at each step. Formally, we define the context embedding at step $t \in [1, \dots, T]$ as follows,

$$\mathbf{h}_c^{(N)} = [\bar{\mathbf{h}}^{(N)}, \mathbf{e}_f', \mathbf{h}_{a_{t-1}^f}^{(N)}, Q_t, FT_t], \qquad (23)$$
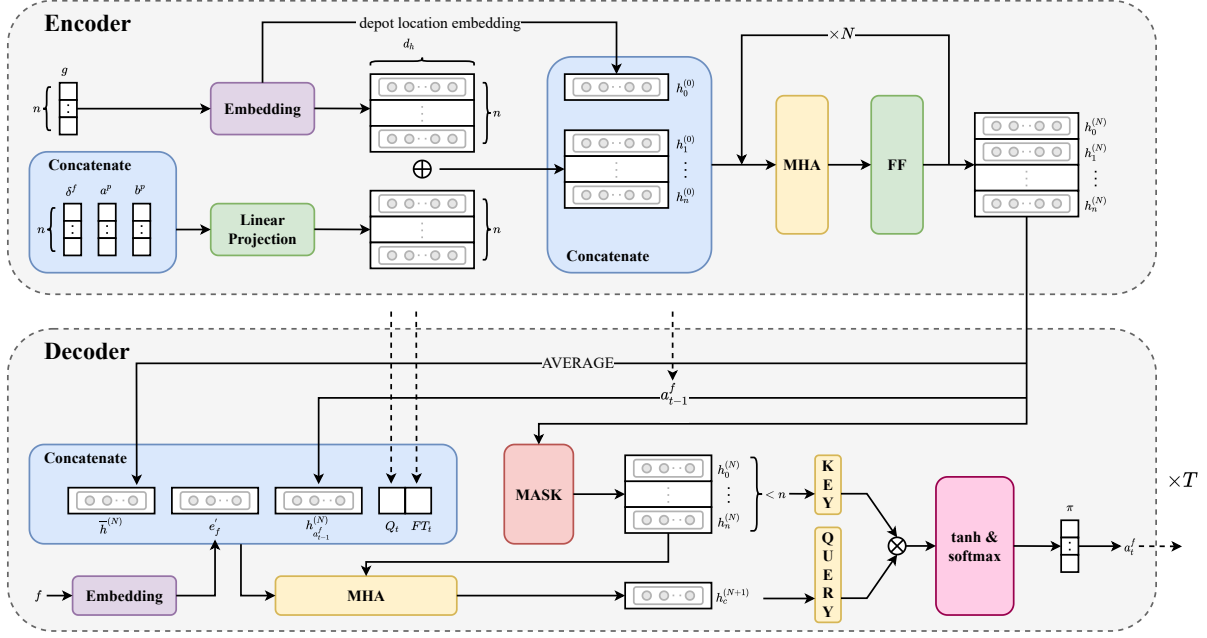
Fig. 3: The architecture of the policy network to construct VRP solutions for respective operations in AGH.

where $\overline{\mathbf{h}}^{(N)} = \frac{1}{n+1} \sum_{j=0}^{n} \mathbf{h}_j^{(N)}$ is the graph embedding, defined as the mean pooling of the advanced node embeddings from the encoder; $\mathbf{e}_f'$ is the fleet embedding used to distinguish the current operation f from others (similar to the location embedding, we also use a learnable lookup table to represent it); $\mathbf{h}_{a_{t-1}}^{(N)}$ is the embedding of the last (selected) node; Q_t represents the remaining capacity of the current vehicle and FT_t is the completion time on the last node by the current vehicle. Among them, while the graph embedding $\overline{\mathbf{h}}^{(N)}$ and fleet embedding $\mathbf{e}_f'$ are fixed, others vary at each step during the solution construction for sub-problem x^f. We define $a_0^f = 0$ and start all tours of vehicles from the depot node, where each tour corresponds to a vehicle. We initialize Q_t and FT_t as $Q_1 = 1$ and $FT_1 = 0$ and update them according to Eqs. (24) and (25), respectively,

$$Q_{t+1} = \begin{cases} 1, & j_t = 0 \\ Q_t - \delta_{j_t}^f, & j_t \neq 0 \end{cases} \qquad (24)$$

$$FT_{t+1} = \begin{cases} 0, & j_t = 0 \\ \max(FT_t + t_{j_{t-1}j_t}^f, a_{j_t}^p) + d_{j_t}^f, & j_t \neq 0 \end{cases} \qquad (25)$$

where we define the node selected at current step as $j_t = a_t^f$, and the one at last step as $j_{t-1} = a_{t-1}^f$; $t_{j_{t-1}j_t}^f$ is the travel time from j_{t-1} to j_t; and $\max(FT_t + t_{j_{t-1}j_t}^f, a_{j_t}^p)$ refers to the start time of an operation on flight j_t, which is the larger value between the time when a vehicle arrives at flight j_t and the start of the time window assigned to the flight. For example, if the latter is larger, a vehicle cannot serve the flight until $a_{j_t}^p$ (in Eq. (14)), i.e., the arrival time of the flight or the latest completion time of operations in the preceding precedence level. Otherwise, the former is larger and the

selected flight is not to be served until the arrival of the current vehicle. Note that both Q_t and FT_t will be reinitialized if the vehicle visits the depot, which practically indicates that another vehicle starts from the depot to serve unvisited flights since the vehicles are homogeneous. However, this procedure is only used to showcase the tour construction. In practice, the vehicles in the same fleet could depart at the same time according to their respective routes to ensure that the earliest available flight could be served timely, while others could wait at the corresponding flights if needed. Moreover, our method also has favorable potential to handle heterogeneous vehicles as long as the relevant properties are properly embedded.

Given the context embedding $\mathbf{h}_c^{(N)}$, we update it to $\mathbf{h}_c^{(N+1)}$ through a MHA layer with message passing only from the flight nodes to the context node, which is similar to the *glimpse* function in [9] as follows,

$$\mathbf{h}_c^{(N+1)} = \text{MHA}(\mathbf{h}_c^{(N)}; \mathbf{h}_0^{(N)}, \dots, \mathbf{h}_n^{(N)}), \qquad (26)$$

where we ignore FF layer, skip-connection and batch normalization (those used in encoder) for the sake of higher efficiency. On the other hand, the flight nodes should be masked during the update of the context embedding if they violate related constraints. For example, if flight j has already been served in the previous step, then $\mathbf{h}_j^{(N)}$ is not passed to the MHA layer in Eq. (26). Formally, the masking of the flight m_t^j and depot m_t^0 at step t are updated sequentially as follows,

$$m_t^j = \begin{cases} \text{FALSE}, & j \notin a_{1:t-1}^f \text{ and } \delta_j^f \leq Q_t \text{ and} \\ & \quad , \max(FT_t + t_{j_{t-1}j}^f, a_j^p) + d_j^f \leq b_j^p \\ \text{TRUE}, & \textit{otherwise} \end{cases} \qquad (27)$$

$$m_t^0 = \begin{cases} \text{TRUE}, & j_{t-1} = 0 \text{ and } \exists j \in \mathcal{N} - \{0\}, m_t^j = \text{FALSE} \\ \text{FALSE}, & \textit{otherwise} \end{cases} \qquad (28)$$

where *TRUE* means the node is masked and thus cannot be selected at the current step. Specifically, the flight $j \in \mathcal{N} - \{0\}$ is masked if it has been already served by a vehicle regarding the current operation $f \in s(p)$, or its demand exceeds the remaining capacity of the vehicle, or the current vehicle cannot finish the service before the end of time window b_j^p. For the depot, we do not allow it to be immediately visited again if it has already been visited at last step while there are still unserved flights that can be visited without violating any constraint at the current step.

The score of selecting each action is realized through a single-head attention mechanism as described in Eq. (29). To satisfy constraints in AGH, the score of node j needs to be masked as $v_t(j) = -\infty$ if m_t^j = TRUE. Accordingly, the final probability distribution over candidate actions is calculated with the Softmax function as shown in Eq. (30).

$$v_t(j) = \begin{cases} C \cdot \tanh(\frac{(\mathbf{W}_Q \mathbf{h}_c^{(N+1)})^T (\mathbf{W}_K \mathbf{h}_j^{(N)})}{\sqrt{d_h}}), & m_t^j = \text{FALSE} \\ -\infty, & m_t^j = \text{TRUE} \end{cases} \quad (29)$$

$$\pi_\theta(a_t^f | x^f, a_{1:t-1}^f) = \frac{e^{v_t(a_t^f)}}{\sum_{j=0}^n e^{v_t(j)}}. \quad (30)$$

Accordingly, our encoder-decoder structured neural network π_θ expresses a stochastic policy to generate a solution (tour) a^f for the fleet (operation) f given the instance x^f, that is,

$$\pi_\theta(a^f | x^f) = \prod_{t=1}^{T} \pi_\theta(a_t^f | x^f, a_{1:t-1}^f), \quad (31)$$

where T means the number of steps used to construct a feasible solution (tour) which serves all aircraft.

D. Policy Training

Given the stochastic policy parameterized by the neural network in Section IV-C, with its solution expressed in Eq. (31), we train the policy network by exploiting the REINFORCE algorithm [51] with a rollout baseline [52]. The pseudocode of training procedure is displayed in Algorithm 1, where L_i^f and $\mathcal{B}_i^f$ mean the cost (length) of the constructed route (tour) for fleet f in the i-th instance by the current trained model and the best saved one, respectively. Correspondingly, $\{\{L_i^f\}_{i=1}^B\}_{f=1}^F$ and $\{\{\mathcal{B}_i^f\}_{i=1}^B\}_{f=1}^F$ mean the list of costs in all operations and instances by the two models, respectively. After each epoch, we use the best checkpoint (i.e. the best saved model so far) to greedily solve instances in the validation set X^{val},[3] and the current model is also used to solve the set. We substitute the current one for the best checkpoint if it can achieve significantly better performance according to a paired t-test [11], as shown in Line 14~17 in Algorithm 1. For each fleet w.r.t. operation f, the loss is defined as the expected solution cost as follows,

$$\mathcal{L}(\theta | x^f) = \mathbb{E}_{\pi_\theta(a^f | x^f)}[L(a^f)], \quad (32)$$

where $L(a^f)$ means a function to compute the length of the constructed route (tour) a^f. Note that since we explicitly

---

[3] We randomly generate 1000 instances after each epoch with the same size and random seed as those used in training to evaluate the current model.

Algorithm 1 Policy Optimization with REINFORCE

1: **Input**: number of epochs E; number of iterations per epoch I; number of operations F; batch size B; significance α.
2: Init θ, $\theta^* \leftarrow \theta$
3: **for** $epoch = 1, ..., E$ **do**
4:     **for** $iter = 1, ..., I$ **do**
5:         generate random instances $\{x_i\}_{i=1}^B$
6:         $\{\{\mathcal{B}_i^f\}_{i=1}^B\}_{f=1}^F \leftarrow$ Greedy($\{x_i\}_{i=1}^B, \theta^*$)
7:         **for** $f = 1, ..., F$ **do**
8:           $\{L_i^f\}_{i=1}^B \leftarrow$ Sample($\{x_i^f\}_{i=1}^B, \theta$)
9:           update time windows as Eqs. (14) and (15)
10:         **end for**
11:         obtain gradient $\frac{1}{B}\nabla\mathcal{L}$ using Eq. (33)
12:         $\theta \leftarrow$ Adam($\theta, \frac{1}{B}\nabla\mathcal{L}$)
13:     **end for**
14:     $\{\{L_i^f\}_{i=1}^{|X^{val}|}\}_{f=1}^F \leftarrow$ Greedy(X^{val}, θ)
15:     $\{\{\mathcal{B}_i^f\}_{i=1}^{|X^{val}|}\}_{f=1}^F \leftarrow$ Greedy(X^{val}, θ^*)
16:     **if** t-test($\sum_{f=1}^F \{L_i^f\}_{i=1}^{|X^{val}|}, \sum_{f=1}^F \{\mathcal{B}_i^f\}_{i=1}^{|X^{val}|}$) $< \alpha$ **then**
17:         $\theta^* \leftarrow \theta$
18:         generate a new validation set X^{val}
19:     **end if**
20: **end for**

---

embed the vehicle and operation information, it is expected that the learned policy is robust and reliable enough to also tackle VRPs for the heterogeneous fleets (i.e. operations) in AGH. To this end, we perform the gradient back-propagation, with the solution to each fleet a^f separately constructed by the same policy network in Section IV-C. Formally, we optimize the policy network for solving AGH with the following update scheme,

$$\nabla\mathcal{L}(\theta | x) = \sum_{x \in \{x_i\}_{i=1}^B} \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} [(L(a^f) \\ - \mathcal{B}(x^f))\nabla \log \pi_\theta(a^f | x^f)], \quad (33)$$
$$\theta \leftarrow \text{Adam}(\theta, \frac{1}{B}\nabla\mathcal{L}),$$

where $\mathcal{B}(x^f)$ represents the length of the solution to operation f constructed by the baseline (i.e. the best checkpoint) in a greedy manner. We use the above gradient to update the parameters of the policy network, as shown in Line 5~12 in Algorithm 1 Intuitively, the above training diagram does not explicitly handle the interplay between solutions to different sub-problems. Alternatively, we also try to update the policy network with the global solution, which is constructed by using the network to schedule all operations in one go (rather than separate computations for each sub-problem). However, we observe that this alternative paradigm yields similar performance, which in turn increases additional training complexity with a long episode to construct the solution. We compare and analyze those different training paradigms in Section V-B.

In summary, we propose a deep learning based construction framework to solve AGH, with constraints properly handled for solution feasibility, the policy network elegantly structured for informative representation and a novel training algorithm

to learn effective policies. Compared to use cases of RL for solving simplified VRP (e.g. TSP or CVRP) in the literature, this is the first time that RL is designed and applied to the complex yet practical VRP in AGH. Extensive experiments in the next section will show that the proposed method achieves significantly better performance than classic methods.

V. EXPERIMENTS AND ANALYSIS

In this section, we evaluate our neural method on AGH instances with 20, 50, 100 flights of 3 types, and 10 types of operations. We compare our method with classic meta-heuristics, construction heuristics and recent methods specifically designed for AGH to verify the superiority. Besides, we evaluate the generalization of the learned policy on instances with larger problem sizes and different parameters, and conduct ablation studies to assess the effectiveness of key designs in the proposed method. Moreover, we also adapt our method to real-time setting with stochastic arrival of flights to verify its capability to handle dynamics.

A. Experimental Settings

Instance Generation. We take the Changi Airport in Singapore as a use case. It involves 3 terminals and 91 gates, and we assume the depot node is placed at a specified location. To conduct the experiment, we load the airport map[4] into SUMO [53], where the distances among locations could be calculated automatically. On top of that, we generate instances with 20, 50 and 100 flights, which are referred as AGH20, AGH50 and AGH100. In those instances, we assign gates randomly to each flight. We consider 3 types of flights (each flight is assigned to a type randomly) and sample their arrival time according to the statistics[5] of real scenarios in Changi airport. Based on the arrival time, the departure time is computed by adding the duration of a turnaround (i.e., 30 minutes, 34 minutes and 33 minutes to the 3 flight types, respectively), which are sufficiently long in general. Regarding operations, we consider the 10 operations, their service duration and precedence relations in Fig. 1 for all instances. Pertaining to each operation, the demand of flights are uniformly sampled from $[1, \ldots, 9]$. We set the capacity of vehicles in each fleet to 30, 40 and 50 in AGH20, AGH50 and AGH100, respectively. Note that we normalize the demand by the capacity of vehicles before feeding them to neural networks, and hence the initial capacity is 1 as shown in Eq. (24). We set vehicle speeds following [54]. For each problem size (i.e. AGH20, AGH50, AGH100), we train the policy network with 100 epochs and generate 12800 instances in each epoch. We randomly generate 1000 instances with the same size and random seed after each epoch as the validation set to evaluate the current model, in order to keep the best-so-far model. For testing, we generate 1000 instances of AGH20, AGH50 and AGH100 with a random seed that is different from the one for training. Similarly, we generate 1000 instances of AGH200 and AGH300 for evaluating the generalization

[4]https://www.changiairport.com/en/maps.html
[5]https://www.changiairport.com/en/flights.html

across problem sizes. In addition, we also test our method to solve AGH instances with different parameters and real-time settings, with the instance generation introduced in the corresponding section (i.e. Section V-D and V-E).

Baselines. We compare the learned policy with representative meta-heuristics, construction heuristics and recent specialized methods for AGH. All baselines with their implementation details are introduced as follows.

- CPLEX (v20.1), a mature and highly optimized commercial solver for mixed integer linear programming (MILP) [55].
- Insertions, a type of construction heuristics to solve VRPs. We compare with random, nearest, farthest insertion and nearest neighbor [56].
- Clarke and Wright Savings (CWS), a construction heuristic to solve VRPs [57]. The first flight to be served is selected according to the start of time windows, and the route is constructed progressively according to the distance savings and temporal relations between every two flights.
- Simulated Annealing (SA), a meta-heuristic that mimics the physical annealing process [58]. We use the nearest neighbor heuristic to find the initial solution, and equip it with *swap* operator to explore the solution space (since it is better than *2-opt* for AGH according to our preliminary results). We use Metropolis criterion to accept solutions. Concretely, we run SA for each fleet independently. We set the size of the neighborhood (defined by the swap operator) to 500, the maximum iteration number to 100, the initial temperature to 200, the cooling factor to 0.9, and the time limit to 30m.
- Large Neighborhood Search (LNS), a meta-heuristic that generalizes neighborhood search for optimization by iteratively refining an incumbent solution with local search [59]. Instead of explicitly defining a neighborhood function (e.g., k-opt operation), LNS defines the neighborhood implicitly through a pair of destroy and repair operators. For each instance, we obtain an initial solution with a simple heuristic based on the nearest neighbor heuristic. Following the LNS framework in [60], the destroy operator iteratively selects decision variables in MILP to be reoptimized, and the repair operator (i.e., CPLEX) attains a feasible solution in the reoptimization. The feasible solution replaces the incumbent one to be further improved afterwards if it has a better objective value. Concretely, in each LNS iteration, we randomly select 50 percent of decision variables to be destroyed (and reoptimized) while fixing others. The time limit of each LNS iteration is set to 20s, 1m and 2m for AGH20, AGH50 and AGH100, respectively.
- LNS_SA, a natural combination of LNS and SA where SA is used as the acceptance criterion in LNS framework. Specifically, instead of simply greedily accepting new feasible solutions, we use Metropolis criterion to accept solutions as used in the SA. The initial temperature is 200, and is cooled by 0.95 every 10 LNS iterations. The other settings (e.g., the time limit of each LNS iteration)

TABLE I: Performance regarding ablation studies.

Method	AGH100 (Greedy)			AGH100 (Sampling)		
	Obj.	Gap	Time	Obj.	Gap	Time
$\mathcal{L}_G$	576733.94	139.53%	4.09s	531477.94	136.92%	5.35s
$\mathcal{L}_{M_G}$	363241.50	50.73%	4.10s	316558.09	41.02%	5.28s
$\mathcal{L}_{M_F}$	299984.50	24.47%	4.10s	263335.75	17.29%	5.27s
w/o TW	338958.66	40.72%	4.12s	301930.75	34.53%	5.24s
w. LSTM	245940.63	2.01%	4.22s	227446.69	1.27%	5.34s
Ours	**242930.86**	**0.75%**	4.08s	**225617.61**	**0.45%**	5.13s

Fig. 4: Training curves regarding ablation studies.

remain the same as the ones described above.

- Genetic Algorithm (GA), a recent method specifically designed for AGH [8]. While the original work only considers one type of aircraft, we adapt it to our setting with heterogeneous aircraft. The solution (chromosome) of each fleet is represented as an array, where the first half represents the vehicle index required by each flight, and the last half represents the order in which each vehicle serves the flight. The selection, crossover and mutation operators are the tournament selection, simulated binary crossover and polynomial mutation, respectively. The hyperparameters follow the ones used in [8].
- CP_LNS, which tackles similar AGH to ours by first decomposing it into sub-problems for each fleet according to temporal constraints, and then applying LNS with constraint programming (CP) to solve them [7], [20]. We adapt it to solve exactly the same AGH to ours. The neighborhood structure is defined by two operators: the Random Pivot operator (RPOP), which removes and reinserts individual customers, and the Small Routing (SMART), which relies on arc exchanges. The time limit of each LNS iteration is kept the same as the vanilla LNS described above.

We report the average performance in terms of objective values and relative gaps[6] of all methods on the testing set, where the gaps are calculated against the best solution found among all the methods. For our method, we use two decoding strategies, 1) *Greedy*, which always selects the flight with the highest probability (based on the output of policy network) at each step; 2) *Sampling*, which samples 1000 solutions in parallel based on the output probability distribution of policy network, and selects the best one as the final solution. We also report the average time needed to solve a single instance, the setting of which may vary with different methods. Note that we implement (run) all methods on CPU except that our neural method is on GPU. In specific, we run those experiments on a server equipped with a single GeForce RTX-2080Ti GPU card and Intel i9-10940X CPU @ 3.30GHz.

---

[6]In specific, let $\bar{x}$ and $\bar{x}^*$ be the solution found by the current method and the best solution found among all methods, respectively. For each instance, we define the gap of $\bar{x}$ to the best found solution as: $|c(\bar{x}) - c(\bar{x}^*)|/c(\bar{x}^*)$, where $c(\cdot)$ is the cost of a solution. We report the average gap over all instances in the testing set throughout the experiment.

Hyperparameters. Regarding CPLEX and meta-heuristics, we set time limit as 30 minutes to solve instances of AGH20, AGH50 and AGH100. In addition, for GA, we set the population size to 100, crossover probability to 0.7 and mutation probability to 0.3. For SA, we explore 500 candidate solutions in the neighborhood of current solution at each iteration of the local search. The initial temperature is set to 200 with decay rate of 0.9. For LNS, we set the degree of destruction to 0.5, which means we reoptimize a sub-MILP with half unfixed decision variables (while the other half is fixed) in the original MILP at each iteration. Moreover, we use CPLEX as the repair operator in LNS. For each LNS iteration, the time limit for AGH20, AGH50 and AGH100 is set to 20 seconds, 1 minute and 2 minutes, respectively. For LNS_SA, the initial temperature is set to 200 with decay rate of 0.95 for every 10 LNS iterations. For our neural method, we train the policy network for 100 epochs with instances generated on the fly. In each epoch, 200 batches of 64 instances are processed for training, and 1000 instances are evaluated at the end of each epoch. In the neural structure, we use $N = 3$ layers in the encoder and $M = 8$ heads in each multi-head attention layer. The dimension of all embeddings is set to 128. We use Adam optimizer [61] to update the parameters of neural network with a constant learning rate 10^{-4}. The significance α in t-test is set to 5%.

B. Ablation Study

We first conduct ablation studies to verify the effectiveness of key designs in our neural method. Specifically, we evaluate different loss functions to optimize the policy network for AGH, and different embedding methods for the temporal inputs, where we primarily take AGH100 as the study case.

Loss Functions. While the learnt policy for each sub-problem is shared among different fleets, we concern that loss $\mathcal{L}(\theta|x^f)$ in Eq. (32) only optimizing the tour length for each sub-problem may ignore the interplay between sub-problems, i.e., the solutions to the higher precedence level affects the time windows of the succeeding sub-problems. Thus, we also extend the loss function to consider the global solution or solutions to a series of sub-problems. Three candidate loss functions are evaluated as follows,

TABLE II: Comparison with baselines.

Method	AGH20			AGH50			AGH100		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
CPLEX	**80151.01**	**0.00%**	30m	300801.58	104.09%	30m	795687.22	252.23%	30m
SA	112250.94	40.26%	30m	248161.94	69.82%	30m	416360.91	84.94%	30m
LNS	91722.98	14.37%	30m	183769.36	25.73%	30m	320327.66	42.25%	30m
LNS_SA	94816.34	18.45%	30m	183388.98	25.44%	30m	324700.14	44.16%	30m
GA	143664.25	79.45%	30m	304865.51	108.75%	30m	515242.92	129.10%	30m
CP_LNS	114576.62	43.10%	30m	227609.73	55.88%	30m	303815.31	34.88%	30m
Random Insertion	168649.50	96.75%	1.70s	340960.69	135.49%	0.38m	612480.69	171.80%	2.44m
Nearest Insertion	156261.83	82.53%	2.33s	322091.38	122.49%	0.48m	590135.81	161.90%	3.24m
Farthest Insertion	157519.63	83.97%	2.65s	351715.00	143.04%	0.49m	666756.19	195.92%	4.05m
Nearest Neighbor	147099.97	71.44%	0.11s	271017.28	87.04%	0.25s	425558.38	88.75%	0.51s
CWS	106356.66	23.96%	0.10s	190049.50	31.08%	0.25s	300500.69	33.29%	0.50s
Ours(Greedy)	92121.10	7.12%	2.88s	**157540.69**	**8.58%**	3.35s	**242930.86**	**7.67%**	4.08s
Ours(Sampling)	86380.93	0.45%	2.98s	**145099.80**	**0.02%**	3.68s	**225617.61**	**0.00%**	5.13s

1) $\mathcal{L}_G$: We optimize the policy network with solutions of all fleets. The gradient is computed as in Eq. (34), where $L(a)$ and $\mathcal{B}(x)$ represent costs of the global solutions attained by the current model and baseline model, respectively. This approach optimizes the multi-fleet vehicle routing in AGH as a whole,

$$\nabla\mathcal{L}_G(\theta|x) = (L(a) - \mathcal{B}(x))\nabla\log\prod_{f\in\mathcal{F}}\pi_\theta(a^f|x^f). \quad (34)$$

2) $\mathcal{L}_{M_G}$: As an alternative, we define the gradient as a weighted sum of the solution cost of each sub-problem and that of the global solution ($\alpha = 0.95$),

$$\nabla\mathcal{L}_{M_G}(\theta|x) = \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}[(\alpha(L(a^f) - \mathcal{B}(x^f)) +$$
$$(1-\alpha)(L(a) - \mathcal{B}(x)))\nabla\log\pi_\theta(a^f|x^f)]. \quad (35)$$

3) $\mathcal{L}_{M_F}$: We further refine $\mathcal{L}_{M_G}$ by only considering the solution costs of sub-problems with lower priorities,

$$\nabla\mathcal{L}_{M_F}(\theta|x) = \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}[(\alpha(L(a^f) - \mathcal{B}(x^f)) +$$
$$(1-\alpha)\sum_{p\prec p'}\sum_{f'\in s(p')}(L(a^{f'}) - \mathcal{B}(x^{f'})))\nabla\log\pi_\theta(a^f|x^f)]. \quad (36)$$

Note that, same as Eq. (32), the above loss functions are used to update the policy network after the global solution to AGH is constructed. Then we train the policies with the above loss functions and test them on AGH100 instances. The results are gathered in TABLE I, where we observe that the loss function in Eq. (32) achieves better performance than the above three variants. Meanwhile, we display the validation performance during training in Fig. 4. We observe that the training process with the loss $\mathcal{L}_G$ or $\mathcal{L}_{M_G}$ is difficult to be converged. It might result from the relatively high complexity involving the whole AGH solution for $\mathcal{L}_G$, and the noisy information for $\mathcal{L}_{M_G}$. Here, the noisy information refers to the solution costs from

operations that are not related much, since the solution to an operation may only affect the ones with lower priorities. On the other hand, although $\mathcal{L}_{M_F}$ performs better than $\mathcal{L}_{M_G}$, it is still inferior to $\mathcal{L}(\theta|x^f)$ based on our empirical results. Thus we conclude that the loss $\mathcal{L}(\theta|x^f)$ is sufficiently effective and efficient to train a desirable policy to solve AGH.

Temporal Embeddings. Since the time windows in each operation are affected by solutions to the operations with a higher priority in AGH, we would like to emphasize the importance of embedding such temporal information, and also investigate the recurrent neural network (e.g., LSTM) for time window embedding. Specifically, the temporal input a_j^p and b_j^p are first processed with a linear projection, and then passed to LSTM accompanied with the hidden and cell states. Those hidden and cell states are initialized with zero vectors and updated as the output of LSTM at the last step, which is regarded as the embedding of temporal feature, and then added to the initial embedding $\mathbf{h}_j^{(0)}$. As shown in TABLE I and Fig. 4, the policy learned without representation of the temporal feature (w/o TW) performs significantly inferior to the one with such representation, both for the training and testing. Also, the results indicate that the recurrent neural network does not bring obvious advantage over ours. It only slightly improves the validation performance at the very beginning of the training. Hence, we stick to the usage of the linear projection in Eq. (17) to embed the temporal input for better parallelism and less computational cost.

C. Comparison Study

We compare our neural method with baselines on AGH20, AGH50 and AGH100 and record the average objective value, primal gap and (needed) runtime. All results are displayed in TABLE II. We observe that CPLEX is able to solve small instances well on AGH20 with the given 30 minutes, achieving the smallest objective value and primal gap among all methods. However, its performance saliently degrades on larger instances, which is generally inferior to the other

TABLE III: Generalization study on problem sizes.

Method	AGH200			AGH300		
	Obj.	Gap	Time	Obj.	Gap	Time
LNS	686158.86	86.18%	1h	1576055.58	223.49%	1h
LNS_SA	685258.40	85.88%	1h	1521977.52	212.27%	1h
CP_LNS	471051.22	27.32%	1h	576481.92	18.80%	1h
Random Insertion	1163235.50	216.39%	0.33h	1684740.15	246.08%	1.27h
Nearest Insertion	1110559.75	202.08%	0.47h	1577794.82	224.13%	1.69h
Farthest Insertion	1297243.25	252.82%	0.51h	1895887.19	289.45%	1.67h
Nearest Neighbor	664187.06	80.58%	0.97s	858120.69	76.23%	1.43s
CWS	491486.56	33.60%	1.01s	638389.44	31.09%	1.52s
Ours(Greedy)	**387361.94**	**5.28%**	6.78s	**504203.06**	**3.53%**	8.55s
Ours(Sampling)	**367934.56**	**0.00%**	9.89s	**487027.66**	**0.00%**	17.49s

TABLE IV: Generalization study on varying parameters.

Method	Gaussian			Poisson		
	Obj.	Gap	Time	Obj.	Gap	Time
LNS	311851.84	37.33%	30m	320533.46	40.44%	30m
LNS_SA	309631.53	36.33%	30m	316322.71	38.89%	30m
CP_LNS	306331.68	34.93%	30m	312833.22	37.75%	30m
Random Insertion	607621.00	166.88%	2.79m	619340.75	172.94%	3.45m
Nearest Insertion	584337.69	156.64%	3.85m	593679.06	161.70%	4.66m
Farthest Insertion	659150.63	189.50%	5.13m	674967.44	197.53%	5.95m
Nearest Neighbor	424645.47	86.38%	0.55s	424585.13	87.00%	0.57s
CWS	304361.41	33.58%	0.57s	311102.69	36.99%	0.52s
Ours(Greedy)	**246531.45**	**8.14%**	5.15s	**245913.59**	**8.23%**	4.89s
Ours(Sampling)	**227973.09**	**0.00%**	6.87s	**227209.89**	**0.00%**	6.84s

methods on AGH50 and AGH100. The reason is that the computational complexity increases exponentially as the size grows due to its NP-hard nature, and CPLEX based on the exact algorithm becomes less effective in searching optimal or high-quality solutions. On the other hand, our neural method can attain the best results on AGH50 and AGH100, and especially when multiple solutions are sampled, the average primal gap is almost 0. It means our method outperforms all baselines on almost all instances. Compared to meta-heuristics, including the two methods (GA and CP_LNS) specialized for AGH, even our method with *Greedy* decoding achieves better results than them on all problem sizes (except for LNS on AGH20). Moreover, our method with *Sampling* decoding further improves the results significantly. Note that our method is more efficient than meta-heuristics, since the policy network constructs the solution to an instance once without additional steps for iterative improvements. Additionally, while GA and CP_LNS resort to domain knowledge and trial-and-error to derive the algorithm, e.g., the design of operators, our method can learn more powerful policies automatically. Meanwhile, it is also clear that the learned construction heuristic by our method is considerably superior to classic insertion heuristics and CWS, which again verifies its favorable capability in solving AGH.

D. Generalization Study

After the policy is trained, we expect that it can generalize well to larger unseen instances. To demonstrate such a capability, we directly apply the policy trained on AGH100 to solve 1000 instances of AGH200 and AGH300, which are generated following almost the same procedure as described in Section V-A. The only difference is that we set the capacity of vehicles to 60 and 70 for AGH200 and AGH300, respectively. For meta-heuristics, we only report the results of LNS, LNS_SA and CP_LNS with the time limit of 1 hour, since we found that they perform better than SA and GA with a similar pattern in Section V-C. We also compare with classic construction heuristics, and all results are summarized in TABLE III. It is revealed that CP_LNS and CWS attain the best results among the meta-heuristics and construction heuristics, respectively. On the other hand, our method with either *Greedy* or *Sampling*

decoding strategy can still yield better solutions than all baselines on both AGH200 and AGH300. Moreover, the needed runtime of the construction heuristics is shorter than that of meta-heuristics, exhibiting a desirable efficiency on the large instances. In summary, our method is able to achieve the best solutions with reasonably short runtime when generalizing to larger AGH instances.

On the other hand, it is also necessary to evaluate the trained model with instances from varying distributions, so as to verify its potential to be used with diverse modes of instance parameters. Although our instance generation mostly follows the real scenarios in Changi airport, we further manifest the power of the model with varying flight demands and arrival time. Specifically, we sample flight demands from Gaussian (with mean 5 and variance 2.5) and Poisson distributions (with the expected number of events 5), respectively. Similarly, we also sample the arrival time of aircraft by assigning probabilities to each hour, which are derived from Gaussian (with mean 0 and variance 1) and Poisson (with the expected number of events 4) distributions. We directly apply the trained model to the instances and record the results in TABLE IV. It is clear that our methods with Sampling can achieve the lowest gaps and objective values under either Gaussian or Poisson distribution. Furthermore, the runtime of our method is relatively short compared to the metaheuristics and some of the construction heuristics. Despite the good generalization to the above two distributions, we will apply the proposed method to solve AGH in different airports in the future, so as to evaluate it with other practical parameter distributions.

E. Real-time AGH

In reality, we not only concern a good solution to AGH that could be computed days earlier, but also hope the model could handle real-time AGH that is characterized with stochastic arrivals of flights. In other words, when the known flights are being scheduled, subsequent (new) flights will arrive at airport at random time which are not known in advance. To evaluate this property, we adapt our method and construction heuristics to the real-time AGH setting (in a re-optimization manner), which mimics the real-world dynamics in an airport. When the new flights come, we fix the scheduling of operations before

TABLE V: Results for real-time AGH

Method	AGH50→100		
	Obj.	Gap	Time
Random Insertion	558672.81	102.72%	2.02m
Nearest Insertion	429569.28	55.97%	2.98m
Farthest Insertion	429871.59	56.11%	2.86m
Nearest Neighbor	515896.97	87.60%	3.25s
CWS	336290.59	21.99%	2.90s
Ours(Greedy)	**305379.37**	**10.55%**	10.99s
Ours(Sampling)	**276043.84**	**0.00%**	14.90s

current timestamp, and continue to schedule the vehicle fleets for *unserved* flights. Following the same instance generation procedure in Section V-A, we first generate 500 instances of AGH50, and gradually add new incoming flights (until 100 flights in each instance) during inference whose arrival time are sampled according to the statistics of CHANGI airport. Note that the arrival time of new flight cannot precede the current timestamp, and hence is revealed dynamically during the execution of operations. From the results in TABLE V, it is exhibited that our method can deal with well the dynamic arrivals of flights and offer shorter route to dispatch all fleets for operations. In addition, our method runs efficiently which meets the requirements of real-time AGH, which well justified its favorable capability of handling dynamics and randomness in practice.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we first propose a neural method to solve AGH. We present a construction framework for AGH with multiple (types of) operations, which decomposes the studied problem into VRPs for each fleet and solves them with a construction heuristic following the precedence relation. Particularly, we concretize the construction heuristic with an attention-based policy network and train it by the RL algorithm, which is shared by each fleet (sub-problem). Results show that our method outperforms classic construction heuristics, meta-heuristics and existing specialized methods for AGH, in terms of solution quality and computational efficiency. Moreover, our method generalizes well to instances with larger scales or different parameters and performs favorably on real-time AGH with stochastic flight arrivals, which yields superior results to all baselines.

In addition, the ablation study provides two pivotal insights in the algorithmic design for solving AGH. First, the time window is an important factor that intertwines sub-problems in AGH. The linear projection is empirically found to be better than LSTM to embed this information into the policy network. Second, optimizing the solutions to sub-problems gains superior performance to the direct optimization of the global solution during training. These findings may serve as a source of inspiration for the use of DRL in addressing other AGH variations and similar practical VRPs.

We would like to note that this work is an early attempt to solve AGH with deep learning, which has substantial practical

significance. First, it benefits the algorithmic development for AGH in the data-driven fashion, without much trial-and-error and domain knowledge to design hand-crafted rules or operators. Also, the neural method can deliver superior solutions to classic methods in a fairly short time, which has a potential to promote the efficiency of airport management and the economics of aviation. Furthermore, the proposed method belongs to an appealing application of DRL for solving practical VRPs with complex constraints, in comparison to most existing neural methods for simple and standard VRPs. In the future, we will, 1) apply our method to much larger AGH instances such as the ones with 1000 flights; 2) adapt it to other stochastic settings, e.g., the uncertain service time for operations; 3) involve heterogeneous vehicles in fleets.

REFERENCES

[1] P. Zhao, W. Gao, X. Han, and W. Luo, "Bi-objective collaborative scheduling optimization of airport ferry vehicle and tractor," *International Journal of Simulation Modelling*, vol. 18, no. 2, pp. 355–365, 2019.

[2] S. Zhu, H. Sun, and X. Guo, "Cooperative scheduling optimization for ground-handling vehicles by considering flights' uncertainty," *Computers & Industrial Engineering*, p. 108092, 2022.

[3] J. Y. Du, J. O. Brunner, and R. Kolisch, "Planning towing processes at airports more efficiently," *Transportation Research Part E: Logistics and Transportation Review*, vol. 70, pp. 293–304, 2014.

[4] S. C. Ho and J. M. Leung, "Solving a manpower scheduling problem for airline catering using metaheuristics," *European Journal of Operational Research*, vol. 202, no. 3, pp. 903–921, 2010.

[5] X. Feng, H. Zuo, and Q. Sun, "Research on collaborative scheduling of aircraft ground service vehicles based on simple temporal network," in *2021 IEEE 3rd International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*. IEEE, 2021, pp. 263–269.

[6] W. Guo, P. Xu, Z. Zhao, L. Wang, L. Zhu, and Q. Wu, "Scheduling for airport baggage transport vehicles based on diversity enhancement genetic algorithm," *Natural Computing*, vol. 19, no. 4, pp. 663–672, 2020.

[7] S. Padrón, D. Guimarans, J. J. Ramos, and S. Fitouri-Trabelsi, "A bi-objective approach for scheduling ground-handling vehicles in airports," *Computers & Operations Research*, vol. 71, pp. 34–53, 2016.

[8] Y. Liu, J. Wu, J. Tang, W. Wang, and X. Wang, "Scheduling optimisation of multi-type special vehicles in an airport," *Transportmetrica B: Transport Dynamics*, pp. 1–17, 2021.

[9] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[10] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[11] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations (ICLR)*, 2018.

[12] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[13] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[14] A. Norin, T. A. Granberg, P. Värbrand, and D. Yuan, "Integrating optimization and simulation to gain more efficient airport logistics," in *Eighth USA/Europe Air Traffic Management Research and Development Seminar*, 2009, p. 10.

[15] A. Norin, D. Yuan, T. A. Granberg, and P. Värbrand, "Scheduling de-icing vehicles within airport logistics: a heuristic algorithm and performance evaluation," *Journal of the Operational Research Society*, vol. 63, no. 8, pp. 1116–1125, 2012.

[16] Z. Zhou, S. Liu, and K. Huang, "Research on airport trailer emergency scheduling model based on genetic simulation annealing algorithm," in *IOP Conference Series: Materials Science and Engineering*, vol. 383, no. 1. IOP Publishing, 2018, p. 012044.

[17] X. Han, P. Zhao, Q. Meng, S. Yin, and D. Wan, "Optimal scheduling of airport ferry vehicles based on capacity network," *Annals of Operations Research*, vol. 295, no. 1, pp. 163–182, 2020.

[18] Y. Du, Q. Zhang, and Q. Chen, "Aco-ih: An improved ant colony optimization algorithm for airport ground service scheduling," in *IEEE International Conference on Industrial Technology*, 2008, pp. 1–6.

[19] K. A. Zampirolli and A. R. S. Amaral, "Simulated annealing and iterated local search approaches to the aircraft refueling problem," in *International Conference on Computational Science and Its Applications*. Springer, 2021, pp. 422–438.

[20] S. Padrón and D. Guimarans, "An improved method for scheduling aircraft ground handling operations from a global perspective," *Asia-Pacific Journal of Operational Research*, vol. 36, no. 04, p. 1950020, 2019.

[21] Y. S. Gök, D. Guimarans, P. J. Stuckey, M. Tomasella, and C. Ozturk, "Robust resource planning for aircraft ground operations," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2020, pp. 222–238.

[22] Y. S. Gök, M. Tomasella, D. Guimarans, and C. Ozturk, "A simheuristic approach for robust scheduling of airport turnaround teams," in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 1336–1347.

[23] G. Andreatta, L. Capanna, L. De Giovanni, M. Monaci, and L. Righi, "Efficiency and robustness in a support platform for intelligent airport ground handling," *Journal of Intelligent Transportation Systems*, vol. 18, no. 1, pp. 121–130, 2014.

[24] K. Ng, C. K. Lee, F. T. Chan, and Y. Lv, "Review on meta-heuristics approaches for airside operation research," *Applied Soft Computing*, vol. 66, pp. 104–133, 2018.

[25] J. Liu, Z. Guo, and B. Yu, "Optimising gate assignment and taxiway path in a discrete time–space network: integrated model and state analysis," *Transportmetrica B: Transport Dynamics*, pp. 1–23, 2022.

[26] Y. Wang, Z. Wang, X. Hu, G. Xue, and X. Guan, "Truck–drone hybrid routing problem with time-dependent road travel time," *Transportation Research Part C: Emerging Technologies*, vol. 144, p. 103901, 2022.

[27] Y.-J. Liang and Z.-X. Luo, "A survey of truck–drone routing problem: Literature review and research prospects," *Journal of the Operations Research Society of China*, vol. 10, no. 2, pp. 343–377, 2022.

[28] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems.." *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[29] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1621–1632.

[30] M. Lee, S. Hosseinalipour, C. G. Brinton, G. Yu, and H. Dai, "A fast graph neural network-based method for winner determination in multi-unit combinatorial auctions," *IEEE Transactions on Cloud Computing*, 2020.

[31] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 21 188–21 198.

[32] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[33] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 6278–6289.

[34] Y. Ma, J. Li, Z. Cao, W. Song, H. Guo, Y. Gong, and Y. M. Chee, "Efficient neural neighborhood search for pickup and delivery problems," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 7 2022, pp. 4776–4784.

[35] J. Li, Y. Ma, R. Gao, Z. Cao, L. Andrew, W. Song, and J. Zhang, "Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem," *IEEE Transactions on Cybernetics*, 2021.

[36] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 11 096–11 107.

[37] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang, "Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2306–2315, 2021.

[38] L. Xin, W. Song, Z. Cao, and J. Zhang, "Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021.

[39] J. Zhou, Y. Wu, Z. Cao, W. Song, J. Zhang, and Z. Chen, "Learning large neighborhood search for vehicle routing in airport ground handling," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[40] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2692–2700.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[42] Y. Kaempfer and L. Wolf, "Learning the multiple traveling salesmen problem with permutation invariant pooling networks," *arXiv preprint arXiv:1803.09621*, 2018.

[43] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, and C. Zhang, "Reinforcement learning with multiple relational attention for solving vehicle routing problems," *IEEE Transactions on Cybernetics*, 2021.

[44] L. Xin, W. Song, Z. Cao, and J. Zhang, "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems," in *Proceedings of 35th AAAI Conference on Artificial Intelligence*, 2021, pp. 12 042–12 049.

[45] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[46] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, "A note on learning algorithms for quadratic assignment with graph neural networks," in *International Conference on Machine Learning (ICML)*, vol. 1050, 2017, p. 22.

[47] C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent, "Learning tsp requires rethinking generalization," in *International Conference on Principles and Practice of Constraint Programming*, 2021.

[48] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[49] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art," *European Journal of Operational Research*, vol. 296, no. 2, pp. 393–422, 2022.

[50] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[51] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[52] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[53] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018, pp. 2575–2582.

[54] S. S. De Vicente, "Ground handling simulation with cast," Ph.D. dissertation, 2010.

[55] IBM(2017), "Ibm ilog cplex 12.10 user manual ibm crop." 2017.

[56] O. Bräysy, W. Dullaert, and M. Gendreau, "Evolutionary algorithms for the vehicle routing problem with time windows," *Journal of Heuristics*, vol. 10, no. 6, pp. 587–611, 2004.

[57] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[58] A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Computers & operations research*, vol. 104, pp. 191–206, 2019.

[59] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook of metaheuristics*. Springer, 2019, pp. 99–127.

[60] J. Song, Y. Yue, B. Dilkina *et al.*, "A general large neighborhood search framework for solving integer linear programs," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 20 012–20 023, 2020.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

Yaoxin Wu received the B.Eng degree in traffic engineering from Wuyi University, Jiangmen, China, in 2015, the M.Eng degree in control engineering from Guangdong University of Technology, Guangzhou, China, in 2018, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in 2023. He was a Research Associate with the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU). He joins the Department of Information Systems, Faculty of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, as an Assistant Professor. His research interests include combinatorial optimization, integer programming and deep learning.

Zhiguang Cao received the Ph.D. degree from Interdisciplinary Graduate School, Nanyang Technological University. He received the B.Eng. degree in Automation from Guangdong University of Technology, Guangzhou, China, and the M.Sc. in Signal Processing from Nanyang Technological University, Singapore, respectively. He was a Research Fellow with the Energy Research Institute @ NTU (ERI@N), a Research Assistant Professor with the Department of Industrial Systems Engineering and Management, National University of Singapore, and a Scientist with the Agency for Science Technology and Research (A*STAR), Singapore. He joins the School of Computing and Information Systems, Singapore Management University, as an Assistant Professor. His research interests focus on learning to optimize (L2Opt).
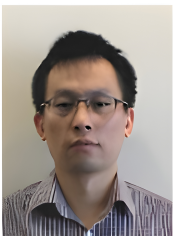
Jianan Zhou received the B.Eng. degree in software engineering from Northeastern University, Shenyang, China, in 2019, and the M.Sc. degree in artificial intelligence from Nanyang Technological University, Singapore, in 2021. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interest includes machine learning with combinatorial optimization problems.

Jie Zhang received the Ph.D. degree from the Cheriton School of Computer Science, University of Waterloo, Canada, in 2009. He is currently a Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. He is also a Professor at the Singapore Institute of Manufacturing Technology. During his Ph.D. study, he held the prestigious NSERC Alexander Graham Bell Canada Graduate Scholarship rewarded for top Ph.D. students across Canada. He was also a recipient of the Alumni Gold Medal at the 2009 Convocation Ceremony. The Gold Medal is awarded once a year to honour the top Ph.D. graduate from the University of Waterloo. His papers have been published by top journals and conferences and received several best paper awards.

Yunwen Xia received the B.Eng. degree in computer science and Technology from Xiamen University, Xiamen, China, in 2019, and the M.Eng degree from Nanyang Technological University, Singapore, in 2022. She was a Project Officier with the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), Nanyang Technological University. Her research interest includes recommendation system and neural combinatorial optimization.

Xianli Zhang received the B.Eng. degree in survey engineering from Wuhan University, Wuhan, China, in 2000, and the M.Eng. degree in electronics and communication engineering from Beihang University, Beijing, China, in 2008. He is currently a Ph.D. candidate with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include scheduling, evolutionary computation, and deep learning.