A hierarchical optimization approach for dynamic pickup and delivery problem with LIFO constraints

Jianhui Du [b], Zhiqin Zhang [a], Xu Wang [b], Hoong Chuin Lau [a*]

a Singapore Management University, School of Computing and Information Systems, 80 Stamford Road, Singapore 178902, Singapore

b Chongqing University, College of Mechanical Engineering, 174 Shazheng Street, Shapingba District, Chongqing, China

Published in Transportation Research Part E: Logistics and Transportation Review (2023) 175, 103131. DOI: 10.1016/j.tre.2023.103131

Abstract: We consider a dynamic pickup and delivery problem (DPDP) where loading and unloading operations must follow a last in first out (LIFO) sequence. A fleet of vehicles will pick up orders in pickup points and deliver them to destinations. The objective is to minimize the total over-time (that is the amount of time that exceeds the committed delivery time) and total travel distance. Given the dynamics of orders and vehicles, this paper proposes a hierarchical optimization approach based on multiple intuitive yet often-neglected strategies, namely what we term as the urgent strategy, hitchhike strategy and packing-bags strategy. These multiple strategies can dynamically adapt to dispatch orders to vehicles according to the status of orders and by considering the travel distance and overtime. To account for the LIFO constraints, block-based operators are designed to schedule the delivery routes, thereby enhancing the search efficiency. The result on real-world instances shows that our proposed hierarchical optimization approach outperforms the current practice and the winning approach in an international competition. Finally, the insights gained from generated instances shows the hierarchical optimization approach has broader applicability.

Keywords: Dynamic pickup and delivery problem, Last in first out, Order dispatching strategy, Neighbourhood search

1. Introduction

In urban logistics systems, dynamic dispatching of orders and route planning of vehicles are vital elements in ensuring timely customer service. In practical distribution applications, logistics companies will face different challenges due to differences in their product characteristics and service guidelines.

The classical vehicle routing problem (VRP) is concerned with known orders and one pickup point (Christofides, 1976, Laporte, 1992), which is the fundamental problem for extensive studies given a central depot. Based on different individual characteristics, many variants have been proposed and applied to optimize the last mile delivery, such as open VRP (Ruiz et al., 2019), VRP with backhaul (Toth and Vigo, 2002), VRP with time windows (Lau et al., 2003), green VRP (Erdoğan and Miller-Hooks, 2012) and multi-echelon VRP (Chen, 2000, Hamdan and Diabat, 2019). In an urban logistics setting, there are often multiple locations, such as material supply points, factories, and warehouses, where the orders are picked up. Thus, the pickup and delivery problem (PDP) has been developed and applied in various contexts such as urban courier services, less-than-truckload transportation, and door-to-door transportation services for the elderly and the disabled (Ropke and Cordeau, 2009). It is evident that PDP requires the vehicles to pick up the orders at origins and then deliver them to destinations. For instance, auto original equipment manufacturers (OEMs) need to pick up goods from different parts warehouses and deliver them to destinations to support production.

In the PDP described above, these orders are all known in one depot before planning, and the decision-making is only applicable to single cycle problems. With the emergence of e-Commerce and on-demand deliveries, order requirements are unknown prior to planning and occur over time as the plan is being executed. Hence, dynamic optimization problems recently become a challenging topic for academics and logistics practitioners (Pureza and Laporte, 2008; Gholami-Zanjani et al., 2019). Due to new orders arriving dynamically, the dynamic pickup and delivery problem (DPDP) needs to dispatch orders and design routes for each vehicle in each timeslot or stage. Contrary to the static delivery problem, vehicles usually make several trips to pick up goods from one or more origins and deliver them to customers per day. Thus, DPDP needs to consider not only the primary objectives in static PDP, but also the viability and flexibility of coping with unknown orders in future (Pureza and Laporte, 2008).

In addition to designing the delivery routes, loading and unloading operations are also critical in the distribution process. Iori and Martello (2010) combined the traveling salesman problem and loading constraints to obtain a better solution for the corresponding logistics targets. When the vehicle has a single access point, the last-in-first-out (LIFO) policy need be observed for a feasible distribution solution, i.e., the last pickup item that has been loaded has to be delivered first (Benavent et al., 2015). If the orders are large, bulky, or fragile items, the load rearrangement on the vehicle may consume much time and increase handling costs. To the best of our knowledge, the DPDP with LIFO constraint has received little attention. It has been recently studied by Li et al. (2021) as well as Xu and Wei (2023).

The main objective of this paper is to propose a very simple to implement hierarchical optimization approach based on multiple strategies and a local search algorithm to solve the DPDP with LIFO constraints. The problem setting is as followings. Each day is divided into equal-duration intervals or epochs (i.e., each interval has a duration of say 10 min), as in Li et al. (2021). In our dynamic environment, new orders are updated at each time interval and decisions need to be made on how the existing plan is to be updated. Under the Markovian assumption that the next state of a dynamic system is related to the current state only, we need to make decisions sequentially over time based on the current state of the system in each time slot (decision epoch). We have developed a Markov decision process (MDP) formulation to describe this DPDP with LIFO constraints, where the constraints ensure that routes respect the vehicle capacity and the LIFO policy. We propose multiple distribution strategies for dynamically dispatching the orders to vehicles and then apply a local search to find a set of routes with minimum total distance and overtime penalty that services all the requests. To validate our model, we perform experiments on the standard PDP dataset and well as instances based on a detailed actual data taken from the International Conference on Automated Planning and Scheduling (ICAPS) 2021 DPDP competition organized by Huawei (Hao et al., 2022, see also https://icaps21.icaps-conference.org/Competitions/). Experimental results show that our proposed approach can often obtain better results under different distribution conditions than those produced by the winning approach for the competition.

The contributions of this paper are as follows:

- We formulate the dynamic pickup and delivery problem with last-in-first-out constraints as a Markov decision process.
- An intuitive and easily implementable hierarchical optimization approach is proposed based on multiple strategies for order dispatching, namely, urgent strategy, hitchhike strategy and packing-bags strategy.
- A computationally efficient local search method is developed based on block-path for route optimization.
- Experimental studies demonstrate the broad applicability and stability of our hierarchical optimization approach when executed on large-scale real data sets.

The remainder of this paper is organized as follows. Section 2 summarizes the literature on dynamic pickup and delivery problem and order dispatching strategies. Section 3 presents the formulation of DPDP with LIFO constraints. In Section 4, we present our hierarchical optimization approach, including the dispatching strategies and local search. The case study and results are shown in Section 5, followed by conclusions in Section 6.

2. Literature review

This section surveys previous works on DPDP and related problems.

According to Eksioglu et al. (2009) the VRP literature has been growing exponentially at a rate of 6% each year (Braekers et al., 2016). Considering real-life requirements and characteristics, several variants of the VRP are proposed, such as multi-depot, multi-echelon, open VRP, and VRP with diverse constraints. The open vehicle routing problem was first addressed by Schrage (1981), which is suitable for meal delivery. The VRP with time windows indicates that the customers must be served within the specified time windows (Desrochers et al., 1992; Spliet et al., 2018). Different constraints enable the VRP model to solve problems in different scenarios. Braekers et al. (2016) conducted a classified review of the VRP literature published between 2009 and June 2015 and analyzed the corresponding trends. Ritzinger et al. (2016) summarized the recent literature in dynamic and stochastic vehicle routing problems. Although VRP is a classical optimization problem, there are many recent studies conducted on it different variants (such as sustainable VRP, EVRP) as well as new solution methods (Nazari et al., 2018; Vidal et al., 2020; Mojtahedi et al., 2021; Sadati et al., 2022). Nazari et al. (2018) developed a framework using reinforcement learning and a parameterized stochastic policy to solve the VRP model. Vidal et al. (2020) summarized the exiting VRP variant and pinpointed current shortcoming and emerging challenges. The VRP can solve the problem if all orders are picked up at the same point and these orders are known beforehand. Due to the diversity of constraints in practical applications, general VRP research is the theoretical background and fundamental that makes it challenging to solve practical problems. Many solutions that did not capture all relevant performance criteria turn out to be impossible to apply in practice (Vidal et al., 2020).

Pickup and delivery problem (PDP) is one of VRP variants, in which couriers have to collect orders in different origins and deliver them to destinations, compared to VRP (Savelsbergh and Sol, 1995). Berbeglia et al. (2007) surveyed the rich PDP literature until 2006 and divided PDPs into 3 classes according to structure: one-to-one, many-to-many, and one-to-many-to-one. In early works, Dumas et al. (1991) presented an exact algorithm based on column generation scheme for PDP and investigated the computational performance using eight problems with 19 to 55 requests; while Lau and Liang (2001) proposed a method to generate good problem instances and benchmarking solutions for the PDP with Time Windows (PDPTW) and a tabu search approach to solve large-scale instances. More recently, Harbaoui Dridi et al. (2020) proposed the use of particle swarm optimization to deal with a challenging variant of PDP. Alyasiry et al. (2019) proposed an exact novel approach based on fragments for PDP with LIFO constraints, taking less time on calculation compared to Cherkesly et al. (2015). However, there is a limit to the problem size that can be solved by the exact algorithm. Dayarian and Savelsbergh (2020) introduced a form of crowdsourcing to increase the availability of resources to address the uncertainty of the DPDP and designed an effective Tabu search heuristic to tackle this problem. Farazi et al. (2022) proposes a deep reinforcement learning (DRL)-based approach to the dynamic on-demand crowdshipping problem. DRL is a method that relies on historical data and cannot complete the solution when there is insufficient data.

In static optimization problems, the complete information about orders and locations is known a priori. In a dynamic optimization problem or DPDP, part or all of the information is revealed dynamically over time (Pillac et al., 2013). For example, on the meal delivery or dial-a-ride (DAR) platform, the customers are unknown until an order or requirement is submitted via mobile application or internet website. Thus, in DPDP, decisions must be made dynamically over time as new information becomes available (Dayarian and Savelsbergh, 2020). That is, at each decision epoch, the platforms need to decide whether to dispatch orders or not and design the delivery routes for each vehicle. Various order distribution strategies have been proposed to explore the DPDP's distinctive features in recent years. The waiting strategy (Mitrović-Minić and Laporte, 2004; Branke et al., 2005) was one of the most researched strategies for dynamic order dispatching problems. The idea behind this strategy was delaying the response of the dispatching new orders and making decision together with the subsequent orders generated in the near future. This strategy was proposed and widely accepted as an effective dynamic distribution strategy, because it implies more real data is being collected between consecutive decisions. In particular, Mitrović-Minić and Laporte (2004) defined two simple waiting strategy (drive-first and waiting-first), then generated two hybrid strategies (dynamic waiting and advanced dynamic) that are combinations of two simple strategies. Pureza and Laporte (2008) applied a waiting strategy and a buffering strategy to vehicles and orders, respectively. However, if we continuously adopt the same strategy in DPDP will not be able to cope with different scenarios because fluctuations in order distribution can affect the effectiveness and efficiency of distribution strategies. In the prior works, the primary available effective strategy for DPDP is the waiting strategy. This paper designs multiple strategies to adapt to changes in DPDP scenarios.

Last-in-first-out policy appeared in inventory management and other applications. In some industries, this policy is necessary for vehicles with only one access door for loading and unloading goods. If the goods transported by the vehicle are hazardous, weighty and/or fragile, the LIFO constraints of pick-up and delivery can reduce handling costs and protect these goods and enhance the safety of the workers (Cordeau et al., 2010; Prasanti et al., 2018). Bombelli and Fazi (2022) research the LIFO constraints as a key component of pickup and delivery problem. LIFO policy dramatically increases the difficulty of solving DPDP, because it directly affects the order of item delivery. Recently, Xu and Wei (2023) developed a metaheuristic approach to solve a re-optimization problem in a DPDP with LIFO constraints. However, the solution methods above cannot be applied to large-scale industrial dynamic problems (like the problem in this paper) where the refresh rate of dynamic orders is short (e.g. every 10 min). For solving such dynamic problems at scale, Ma et al. (2021) and Li et al. (2021) designed deep reinforcement learning approaches. However, their methods require an enormous amount of historical data to train the policy. Besides, the trained policy can only solve problems with similar demand distribution.

Table 1 provides a comparison of various PDP papers published more recently, with Dumas et al. (1991) as the baseline. To our best knowledge, there is a lack of the relevant literature about large-scale dynamic pickup and delivery with LIFO constraints. Hence, the research in this problem is beneficial for the real industrial dynamic logistics problems like supply chain in manufacturing industry, furniture industry and other bulky goods delivery companies. Our work focuses on solving large-scale DPDP with LIFO constraints by designing order dispatching strategies that are intuitive, yet often overlooked. To improve the quality of solution we apply a local search for route optimization based on the concept of blocking.

3. Problem description and model formulation

3.1. Problem description

In this problem, a homogeneous fleet of vehicles serve dynamic orders which are generated throughout the planning horizon. Each order has a pickup point (or node) and delivery point. Vehicle need to pick up orders in pickup points and deliver them to destinations following the LIFO sequence. There is a committed delivery time associated with each order. If the vehicle delivers later than this time, there will be a overtime penalty cost. Apart form this cost, we also consider the total travel distance of the entire fleet in this problem.

Here, we formally introduce the notations. We use $h \in H$ to represent the decision epoch (namely the time slot). In a dynamic planning problem, the time slot can vary from a few seconds to a few minutes depending on the problem characteristics and requirements, which directly affects the timeliness of response and the frequency of re-optimization. The vertex set is partitioned as $N = P \cup D$ comprising all pick-up nodes $P = \{1, 2, \dots, n\}$ and delivery nodes $D = \{n+1, n+2, \dots, 2n\}$. O is the set for all the orders.

Table 1
Literature review of PDP papers.

Paper	Multi depots	Time windows	LIFO	Dynamics orders	Large-scale (Orders>1000)	Cold start
Dumas et al. (1991)		✓			✓	
Cordeau et al. (2010)			✓			✓
Häll et al. (2015)	✓	✓		✓		✓
Zhu et al. (2016)	✓			✓		✓
Alyasiry et al. (2019)	✓	✓	✓			✓
Harbaoui Dridi et al. (2020)	✓	✓				✓
Dayarian and Savelsbergh (2020)		✓		✓		✓
Li et al. (2021)	✓	✓	✓	✓	✓	
Ma et al. (2021)	✓	✓	✓	✓	✓	
Farazi et al. (2022)	✓	✓		✓	✓	✓
Bombelli and Fazi (2022)		✓	✓			✓
Xu and Wei (2023)	✓	✓	✓	✓		✓
This paper	✓	✓	✓	✓	✓	✓

Table 2
Notations.

Notation	Description
Indices	
h	Index for decision epochs
i, j	Index for nodes
k	Index for vehicles
Sets	
H	Set of decision epochs
P	Set of pick-up nodes, $\|P\| = n$
D	Set of delivery nodes, $\|D\| = \|P\| = n$
N	Set of all nodes
O	Set of all orders, $\|O\| = \|P\| = n$
OU_h	Set of all unassigned orders at decision epoch h
OA_h	Set of all assigned orders at decision epoch h
K	Set of all vehicles
Parameters	
QC_k	Capacity of vehicle k
t_i^e	Creation time for order i
t_i^l	Commitment time for order i
q_i	Demand for order i
d_{ij}	Distance from node i to node j
Variables	
a_{ihk}	Binary variable. 1, if the order i is assigned to vehicle k at decision epoch h; 0,otherwise.
x_{ijk}	Binary variable. 1, if vehicle k travels along arc (i, j); 0, otherwise
t_i^d	Integer variable for the delivery time of order i
Q_{ik}	Integer or Continuous variable for the load of vehicle k when it leaves node i

At the beginning of each decision epoch h, we have a set of assigned orders to vehicles OA_h and a set of unassigned orders OU_h. These sets of orders are updated dynamically.

To simplify the model, we use index i to represent both orders and their corresponding pickup nodes. For example, $i = 1$ is the first order and its pickup node. Hence, the number of orders is the same as the number of pickup and delivery nodes, $|O| = |P| = |D|$. For the same order, the demand in pick-up nodes and delivery nodes are negative to each other, i.e., $q_{i+n} = -q_i, \forall i \in P$.

Let $K = \{1, 2, \cdots, k\}$ be the set of vehicles located at a depot in the first time slot. For each vehicle k, QC_k is the capacity, and each vehicle k begins its service at an initial position $F_k^I \in P$. Other notations can be seen in Table 2.

Each order $i \in O$ has a creation time t_i^e and a commitment delivery time t_i^l, which is a soft deadline. The goal is to assign newly generated and previous unassigned orders to vehicles and update the planned route for each vehicle at each decision epoch until all orders are delivered to their destinations. Within each route, the retrieval of orders should comply with the LIFO policy.

3.2. Model formulation

Dynamic problems can be modeled as sequential decision process with stochastic information (Powell, 2011). Here, we formulate this DPDP problem as a Markov Decision Problem (MDP) model that enables us to generate the decisions of dispatching orders to vehicles as well as planning routes for each vehicle.

3.2.1. Decision epoch

The decision epoch in our model is time-based. Comparing with the event-based decision epoch (considering one order to dispatch in one decision epoch), time-based approach is more suitable for dynamic vehicle routing problems as it can consider multiple orders simultaneously to save travel cost. We discretize the planning horizon (for example, a day) into multiple time periods. Here, we use $h = 1, 2 \dots, H$ to represent each decision point (for example, 144 decision epochs, where each step is 10 min).

3.2.2. State space

The system state contains the information necessary to assign unallocated orders to vehicles and schedule each vehicle's route at each decision epoch $h (h \in H)$. We present the state S_h at decision epoch h by a tuple (t_h, OU_h, K). Let t_h be the time of decision epoch h. OU_h is the set of unassigned orders at epoch h. The set K contains the status of vehicles, which is updated at epoch h. Each order $i \in OU_h$ has a pickup location (origin) p_i and a drop-off location (destination) d_i in set N. Each order is associated with demand q_i, creation time t_i^e, committed delivery time t_i^l. Thus, we represent its attributes by the tuple $(q_i, p_i, d_i, t_i^e, t_i^l)$ and the attributes of vehicles $k (k \in K)$ by the tuple $(F_k^{loc}, F_k^{dest}, t_k^{dest}, O_k^{transit}, \mathfrak{R}_k)$. Let F_k^{loc} and F_k^{dest} respectively denote the current location and destination of vehicle k, and t_k^{dest} is the time of the vehicle will arrival at the destination. Let $O_k^{transit}$ be the orders in transit, being carried by vehicle k when this vehicle leaves node i. Let $\mathfrak{R}$ denote the route plan for all vehicles and $\mathfrak{R}_k$ denote the planned route for vehicle k, which is composed of a sequence of nodes. Finally, combining all information,we get:

$$\begin{aligned} S_h &= (t_h, OU_h, K) \\ &= (t_h, \{(q_i, p_i, d_i, t_i^e, t_i^l) | i \in OU_h\}, \{(F_k^{loc}, F_k^{dest}, t_k^{dest}, O_k^{transit}, \mathfrak{R}_k) | k \in K\}) \end{aligned}$$

3.2.3. Action

The action space is made up of a set of feasible actions A_h at each decision epoch h of selecting and assigning orders $i \in OU_h$ to vehicles $k \in K$. A specific action set A_h consists of $a_{ihk} (i \in OA_h, k \in K)$ that represents the order i is assigned to vehicle k at decision epoch h. Apart from the order dispatching, the route plan $\mathfrak{R}$ for all vehicles is also part of the action. Thus, the action in this MDP is $A_h = (a_{ihk}, \mathfrak{R})$.

At each decision epoch h, all actions are subject to PDP constraints. To describe the feasible action space, we first define some notations. The variable a_{ihk} can be replaced by a_{ik} at this epoch h. Q_{ik} is the total demand of the orders on vehicle k when it departs from node i. In this problem, a feasible action must satisfy the following constraints:

$$\sum_{k \in K} a_{ik} = 1, \forall i \in OA_h \tag{1}$$

$$Q_{j-1,k} + \sum_{i \in OA_h} q_i a_{ik} \le QC_k, \forall j \in \mathfrak{R}_k, \forall k \in K \tag{2}$$

$$Q_{jk} = \sum_{i \in O_k^{transit}} q_i, \forall j \in \mathfrak{R}_k, \forall k \in K \tag{3}$$

$$Q_{i+n,k} = Q_{i,k} - q_i, \forall i \in P \tag{4}$$

$$Q_{jk} \ge Q_{ik} + q_j - QC_k(1 - x_{ijk}), \forall i, j \in N \text{ and } i \ne j, \forall k \in K \tag{5}$$

Constraints (1) state that an order can only be assigned to one vehicle. Constraints (2) require that each vehicle always satisfies the capacity constraint at each point j in route $\mathfrak{R}_k$. Constraints (3) calculate the weight when vehicle k leave the node j. Constraints (4) and (5) ensure the LIFO policy.

Although these constraints can limit the action space, the action space is still be combinatorial. Thus, to address the curse of dimensionality, we design a heuristic approach. We discuss our proposed order dispatching strategies in Section 4.1. Details of operators for local search can be found in Section 4.2.

When an action has been taken, the state will transit to the appropriate next state which captures the updated information.

3.2.4. Transition

The transition function maps state S_h to the next state S_{h+1}. The transition of this model comprises two components, including stochastic demand (new orders denoted by ω_{h+1}) in the next decision epoch and deterministic update for the current demand and supply (unassigned orders OU_h and vehicles K_h) given an action A_h described above. After this update, the vehicles serve a subset OU_h^- of the OU_h based on the $\mathfrak{R}$ given by the action A_h. Thus, the $OU_{h+1} = (OU_h - OU_h^-) \cup \omega_{h+1}$.

3.2.5. Objective function

In this section, we describe the objective function of the MDP model, which is taken directly from Hao et al. (2022). The solution for this MDP model is a decision policy π which maps each state S_h to a routes planning $\mathfrak{R}$ for every vehicles. The objective is find a policy π^* to minimize the total cost which is made up of a weighted average of two components, namely travel distance and total overtime of a given solution denoted by route plan $\mathfrak{R}$. The objective function is defined in Eq. (6):

$$min F(\mathfrak{R}) = \lambda f_1(\mathfrak{R}) + f_2(\mathfrak{R}) \tag{6}$$

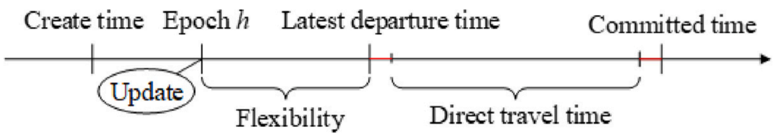

Fig. 1. Timeline of each order.

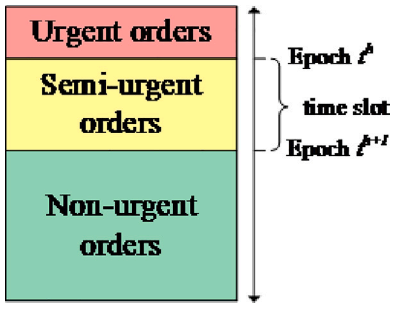

Fig. 2. Order pools.

$$f_1(\mathfrak{R}) = \sum_{i=1}^{|O|} max(0, t_i^d - t_i^l) \tag{7}$$

$$f_2(\mathfrak{R}) = \frac{1}{|K|} \sum_{k=1}^{|K|} \sum_{i=1}^{|N|} \sum_{j=1}^{|N|} d_{ij} x_{ijk} \tag{8}$$

where $f_1(\mathfrak{R})$ is the overtime cost for orders and $f_2(\mathfrak{R})$ is the average distance cost of a vehicle, associated with a route plan $\mathfrak{R}$ as shown in Eqs. (7) and (8). λ is a weight (set to a large positive constant). In this paper, it is set to 10,000 (same as the value used in the ICAPS 2021 competition).

4. Solution approach

In this section, we present a hierarchical optimization approach to solve DPDP, based on multiple order dispatching strategies and local search. The reader may refer Section 4.3 that contains a picture of the overview of our approach.

Specifically, the general idea is to first assign orders to appropriate vehicles using our order dispatching strategies, which also inserts the assigned orders into suitable positions on the current routes of vehicles. Then we further apply a local search procedure to improve these delivery routes.

4.1. Order dispatching strategies

This section presents strategies applied to the hierarchical optimization approach for the DPDP, including urgent strategy, hitchhiker strategy, and packing-bags strategy.

First, we define some notations used in the order dispatching strategies.

Latest departure time: Each order needs to depart before the latest departure time; otherwise, the order will incur an overtime cost. This quantity can be calculated by Eq. (9) and illustrated by Fig. 1. The latest departure time is an essential parameter for identifying the status of the orders.

$$t_i^* = t_i^l - t_i^p - t_i^d - t(arc(p_i, d_i)) \tag{9}$$

Earliest finish time: For each vehicle, we can calculate the vehicle's earliest finish time based on the loaded orders and planned routes.

Buffering pool: A buffering pool is used to postpone the assignment of some non-urgent new requests. We borrow the idea from the work of Pureza and Laporte (2008). This buffering pool is designed to deal with the critical issue of this dynamic pickup and delivery problem namely, how to assign orders which are dynamically generated during the planning horizon. Intuitively, we maintain a buffering pool that contains orders of different types and place them in different sub-pools depending on the *latest departure time* described below.

4.1.1. Generate order buffering pool

All unallocated orders are divided into three order pools according to the latest departure time t_i of each order, as shown in Fig. 2. A detailed description of the algorithm for generating the order buffering pool is given in Figure 1, where t_h is the time at decision epoch h and the t_{h+1} is the time at the next decision epoch $h+1$.

There are three categories of orders:

(1) *Urgent order*: if the *latest departure time* $t_i^* < t_h$, order i is placed in the urgent order pool.

(2) *Semi-urgent order*: if the *latest departure time* $t_h < t_i^* < t_{h+1}$, order i is placed in the semi-urgent order pool.

(3) *Non-urgent order*: if the *latest departure time* $t_{h+1} < t_i^*$, order i is placed in the non-urgent order pool. These three order statuses are closely related to executing of multiple order dispatching strategies below.

Algorithm 1 Generate order buffering pool

```
    Input: $unallocated\_orders$, $epoch\ h$
    Output: $latest\_departure\_time\_list$, $buffering\ pool$
 1: check each $order$ and create $latest\_departure\_time\_list$
 2: sort $unallocated\_orders$ by latest departure time from new to old
 3: for each order in $unallocated\_orders$ do
 4:     $t_i^* = t_i^l - t_i^p - t_i^d - t(arc(p_i, d_i))$
 5:     $T_i = t_i^*$
 6:     if $t_i^* > t_{h+1}$ then
 7:         assign order $O_i$ to $O_{non-urgent}$
 8:     else if $t_h \le t_i^* \le t_{h+1}$ then
 9:         assign order $O_i$ to $O_{semi}$
10:     else if $t_i^* < t_h$ then
11:         assign order $O_i$ to $O_{urgent}$
12:     end if
13: end for
```

4.1.2. Urgent strategy

At the end of Algorithm 1, the urgent order pool can be obtained, which includes orders that need to be allocated immediately. The intuition is that for urgent orders, the sooner they are processed, the less overtime penalty will be incurred. It is inevitable that urgent orders will continue to increase the cost in over time. Thus, the goal of the urgent strategy is to guarantee the priority of processing those urgent orders when there are available vehicles. A high-level overview of the urgent strategy is represented below, and the detailed pseudo code is given in Algorithm 2.

Step 1: Select an urgent order in the urgent order pool sorted by the *latest departure time*.

Step 2: Check the remaining capacity to obtain an available vehicle set V^*.

Step 3: Traverse each vehicle in set V^* and calculate the cost of inserting the urgent order.

Step 4: Assign urgent order to a vehicle with minimum cost.

Step 5: If all urgent orders have been traversed, go to step 6; else go to step 1.

Step 6: Update the planned routes of vehicles.

In Algorithm 2, the *urgent_orders* contains the properties $t_i^*, p_i, d_i,\ q_i$ of each urgent order. and the information of current fleet and orders. F_k^{dest} and Q_k^{remain} indicate the destination and the remaining capacity of vehicle k, which are stored in the vehicles. The functions *get_dest*() and *get_remain*() return the destination and remaining capacity of vehicles, respectively. The input of function *get_insert_overtime_pen* is the vehicle k and order i and its output is the difference of overtime penalty after inserting order i. Similarly, *get_insert_dist()* returns the detour distance. The value C_{ik}^{match}, computed as the evaluation criteria, represents the cost when we insert order i to vehicle k.

The computational complexity of the urgent strategy algorithm (Algorithm 2) is O(n^2) in the worst case. For each urgent order, O(n) nodes are examined (across all vehicles) in the functions in lines 7 and 8, with constant time each. Appending an order to a vehicle takes O(n) time. Since the total number of orders (including urgent orders) is n, the computational complexity is the number of orders n times all the nodes existing on the routes (which is at most n).

4.1.3. Hitchhike strategy

In dynamic routing problems, it is necessary to consider whether an unallocated order can be inserted into a vehicle's current planned route. In our problem, the LIFO policy needs to be considered carefully, which makes insertion challenging. In ridesharing literature, "hitchike" is a powerful strategy to reduce total traveling cost (Chan and Shaheen, 2012). Similarly, in this paper, we borrow it to our logistics problem. In our case, for each unallocated order, the hitchhike strategy aims to select the most appropriate vehicle and insert the pickup and delivery nodes consecutively of a given order to a suitable position in the planned route of this vehicle. In order to accommodate different ratios of demand and supply, we design different levels of operators for the hitchhike

Algorithm 2 Urgent strategy algorithm

```
    Input: $vehicles$; $urgent\_orders$; $latest\_departure\_time\_list$
 1: for each order $i$ in $urgent\_orders$ do
 2:     Initial a empty list $V^*$
 3:     for each vehicle $k$ in $vehicles$ do
 4:         $F_k^{dest} = get\_dest(k)$
 5:         $Q_k^{remain} = get\_remain(k)$
 6:         if $Q_k^{remain} \geq q_i$ then
 7:             $\Delta f_1 = get\_insert\_overtime\_pen(k, i)$
 8:             $\Delta f_2 = get\_insert\_dist(k, F_k^{dest}, i))$
 9:             $C_{ik}^{match} = \lambda \Delta f_1 + \Delta f_2$
10:             Append $(k, C_{ik}^{match})$ $to$ $\boldsymbol{V}^*$
11:         end if
12:     end for
13:     if $\boldsymbol{V}^*$ is not empty then
14:         Find the vehicle $k$ with min $C_{ik}^{match}$
15:         Assign urgent order $i$ $to$ $k$
16:     end if
17: end for
```

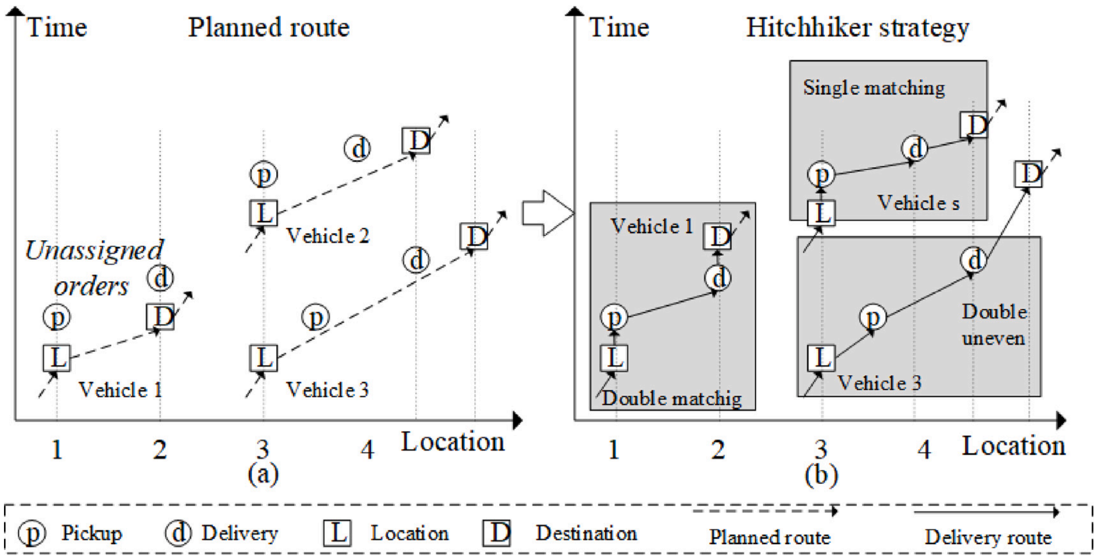

Fig. 3. Schematic of operators for the hitchhike strategy.

strategy, namely double matching, single matching, and double uneven. By these three levels of operators, our hitchhike strategy can dynamically relax the condition of insertion. Thus, the higher this ratio, the less orders have more chance to be inserted as hitchhike orders to avoid detour of vehicles. The definition is as follows, and the schematic is shown in Fig. 3. In this figure, the dotted line is the original planned routes for vehicles, on which the square nodes with ‘L’ and ‘D’ are the nodes of the current location and destination of the vehicles. The circle nodes with ‘p’ and ‘d’ are the pickup and delivery nodes for unassigned orders. After dispatching those unassigned orders, new planning routes are represented by the solid line.

Double matching: Select order whose pickup and delivery nodes are matched with the two upcoming adjacent nodes in the vehicle’s planned route.

Single matching: Select order whose pickup node is the same as the destination of a given vehicle and the distance between this order’s delivery node and the next node in the planned route of this vehicle is acceptably short. Or the delivery node of this order is the same as the next node after the destination of this vehicle and the distance between the order’s pickup node and destination is short enough.

Double uneven: Select orders whose pickup and delivery nodes are close to but not identical to the two coming adjacent points.

We illustrate the different operators of the hitchhiker strategy in Fig. 3. Here the x-axis represents the different locations and y-axis is the time. In this figure, there are three unassigned orders and three vehicles in different positions. Fig. 3(a) denotes the planned routes before the hitchhiker strategy; Fig. 3(b) represents the delivery routes after inserting the hitchhiker orders by using

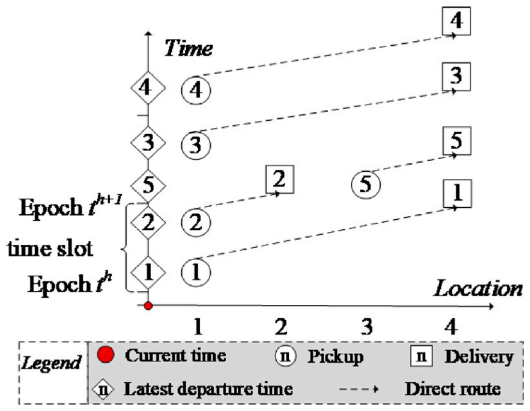

Fig. 4. Unallocated orders.

double matching, single matching, and double uneven operators, respectively. applying double matching operator to vehicle 1, only unassigned orders are inserted into the corresponding locations, if its pickup point and delivery point are same as two adjacent nodes in planned route. Vehicle 2 and vehicle 3 show the single matching and double uneven operators, respectively.

4.1.4. Packing-bags strategy

The intuition of the packing-bags strategy is to pack unassigned orders into bags based on order similarity and related constraints. First, this strategy identifies one unassigned order from the order buffering pool which is sorted by the *latest departure time*. Then, the information of the selected order is set as the base information of this bag. Finally, we compare the similarity of other orders with the base information of the given bag to decide whether to combine them into it. The main consideration related the addition is the available vehicle capacity, location, order demand, and the latest departure time. These orders in one bag will be complied the LIFO policy.

To adapt to different situations, we have designed multiple levels of threshold of packaged operators: same pickup-delivery, same pickup, and same delivery. The definition is as follows, and the schematic is shown in Fig. 4.

Same Pickup-Delivery (SPD): Identify and select orders with the same pickup and delivery points as the bag baseline. This is a high level of threshold for packing-bags strategy leading to fewer bags packed, comparing with the following two levels.

Same Pickup (SP): Identify and select orders with the same pickup point with bag baseline, then decide whether to assign orders to bag based on similarity.

Same Delivery (SD): Identify and select orders with different pickup and delivery points from the bag, but there is an acceptable distance from the bag at both the pickup and delivery points.

Fig. 4 illustrates the pickup and delivery points and the latest departure time of five unallocated orders. In Fig. 5, subgraphs (A), (B), and (C) are the planned routes generated by taking different packing operators for unallocated orders, respectively. Under the Same Pickup-Delivery constraint, only orders 1 and 3 are packed successfully because they have the same pickup and delivery points. The Same Pickup packing operator can assign order 2 and orders 1, 3 together. By contrast, the Same Delivery operator can insert order 5 into the bag of order 1 and order 3.

4.2. Local search

These order dispatching strategies can quickly allocate a large number of orders, however, these strategies do not design the optimal delivery route for each vehicle. In this section, we present the local search operators applied to minimize the travel distance of the planned routes generated by the heuristic rules above. To improve the effectiveness of the local search under LIFO constraints, we propose block-path-based operators.

Block-path: This refers to a sequence of nodes that must include both the origin and the destination of the orders that adhere to the LIFO constraint. This idea of utilizing a block was proposed by Carrabs et al. (2007) for solving the Pickup and Delivery Traveling Salesman Problem with LIFO Loading constraints. In our work, we design three types of block-path, type 1, type 2, and type 3, as shown in Fig. 6. Correspondingly, the block-based operators preserve the feasibility of LIFO constraint, as shown in Carrabs et al. (2007). In the following, we explain our block-based operators in detail.

4.2.1. Insert operator

OnePoint-Insert $(o, k, \mathfrak{R}_k)$: Given an order o, a vehicle k, and planned route $\mathfrak{R}$, the operator returns a new planned route, where the order o is inserted, and pickup and delivery points are adjacent. The OnePoint-Insert operator selects one order and a point in the planned route, and insert the pickup point and delivery point of the selected order into the route. It compares the change in the cost of routes to identify the optimal path, as shown in Fig. 7.

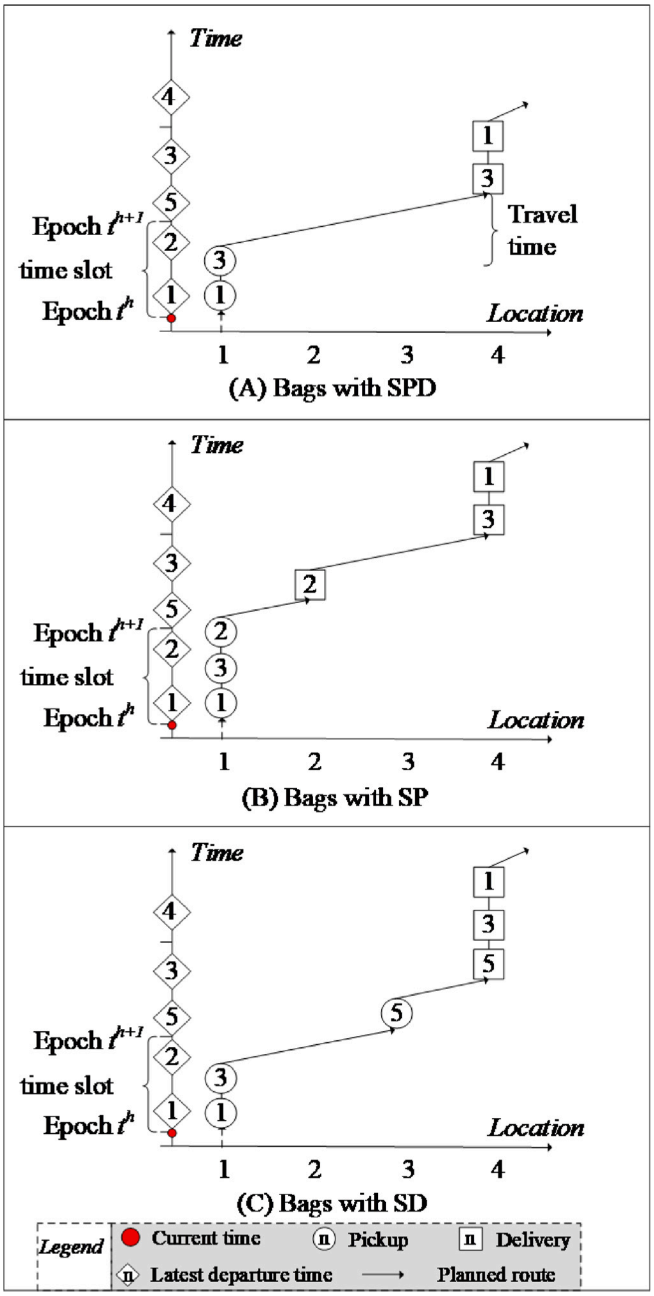

Fig. 5. Schedule options for packing-bags strategy.

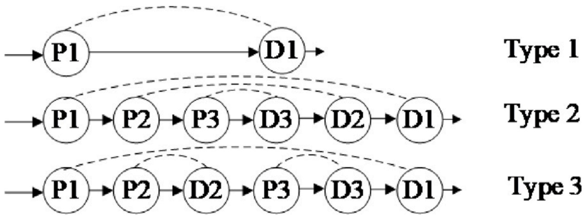

Fig. 6. Schematic diagram of three types of block-path.

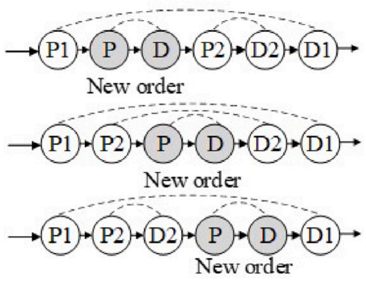

Fig. 7. OnePoint-insert representation.

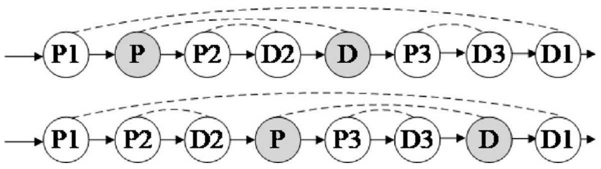

Fig. 8. TwoPoint-insert representation.

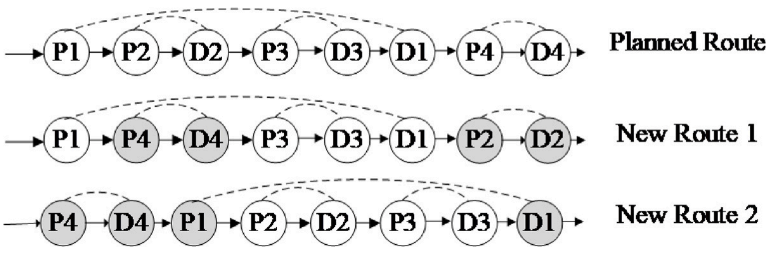

Fig. 9. Exchange operator representation.

TwoPoint-Insert $(o, k, \mathfrak{R}_k)$: Given an order o, a vehicle k, and planned route $\mathfrak{R}_k$, the function can select one block, then insert one order into the planned route. The pickup and delivery points are scattered at both ends of a block, as shown in Fig. 8.

Let $\mathfrak{R}_k$ be a feasible route containing n orders and $2n$ points (including pickup and delivery points). The TwoPoint-Insert operator is described as follows. (1) Select an order (p_i, d_i) from the order pool. (2) Select a block and insert p_i in front of the beginning of the block and insert d_i at its end to produce a new route $\mathfrak{R}_k^*$, as shown in Fig. 8.

The TwoPoint-Insert operator can effectively reduce the computational time complexity compared to the general insert operator that inserts the pickup and delivery points arbitrarily in a standard PDP problem, which is explained below.

Suppose there are n orders in route $\mathfrak{R}_k$, the length of $\mathfrak{R}_k$ is $2n$. When one order is inserted into route $\mathfrak{R}_k$ via the general insert operator, the run time complexity is $2n^2 + n$ because the pickup point has to be inserted before the delivery point. In the case of the TwoPoint-Insert, the run time complexity is reduced to $(n^2 + n)/2$, thereby reducing the run time complexity by $(3n^2 + n)/2$.

4.2.2. Exchange operator

The exchange operator aims to generate a new route by exchanging the location of two block-path that conforms to the LIFO policy because each block is LIFO-compliant. Note that, since the orders that have been picked cannot be moved, we first select the pickup node and then read the corresponding delivery node to get the required block path. For the planned route in Fig. 9, we can obtain the new route 1 by exchanging two block-path (P4D4 and P2D2) or the new route 2 by swapping the location of P4D4 and P1 to D1.

4.3. Hierarchical optimization approach

In summary, the structure of our hierarchical optimization approach is given in Fig. 10. It also shows the interaction between the hierarchical approach and simulator environment, which constantly updates dynamic information, including orders, vehicles, and routes. When a new epoch is triggered or new orders are updated, the algorithm first performs order identification, sorts and

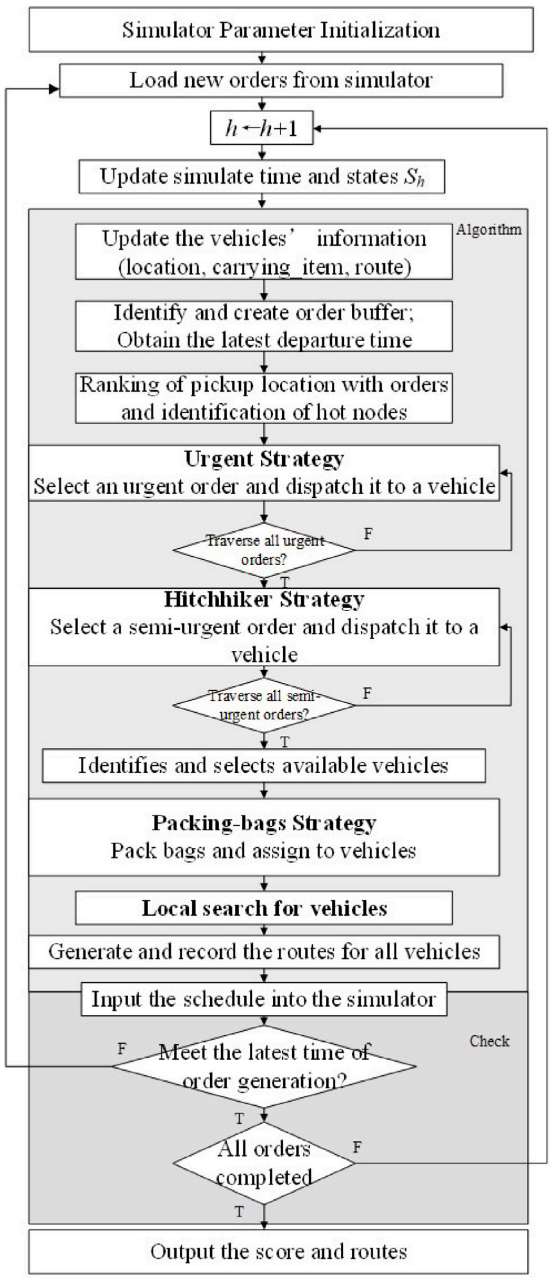
Simulator Parameter Initialization
Load new orders from simulator
$h \leftarrow h+1$
Update simulate time and states S_h
Algorithm
Update the vehicles' information (location, carrying_item, route)
Identify and create order buffer; Obtain the latest departure time
Ranking of pickup location with orders and identification of hot nodes
Urgent Strategy
Select an urgent order and dispatch it to a vehicle
Traverse all urgent orders?
F
T
Hitchhiker Strategy
Select a semi-urgent order and dispatch it to a vehicle
Traverse all semi-urgent orders?
F
T
Identifies and selects available vehicles
Packing-bags Strategy
Pack bags and assign to vehicles
Local search for vehicles
Generate and record the routes for all vehicles
Input the schedule into the simulator
Check
F
Meet the latest time of order generation?
T
All orders completed
F
T
Output the score and routes

Fig. 10. Hierarchical optimization approach for DPDP.

updates the order pool, and then uses each order dispatch strategy in turn to process orders with different states to minimize the overtime quantity. After the order dispatching process, a local search is designed to minimize the total travel distance that does not exceed the overtime quantity. This process is repeated until there is no more unfinished order in the simulator environment.

5. Computational experiments

In this section, we present computational experiments to demonstrate the effectiveness of the proposed solution approach. Section 5.1 describes the test instances obtained from a competition in an international conference, which are realistic problems built around real business scenarios of Huawei Technologies Ltd. To be more general, we randomly generate new instances to compare the applicability of the proposed approach. Section 5.2 details the results of our proposed solution approach for both data

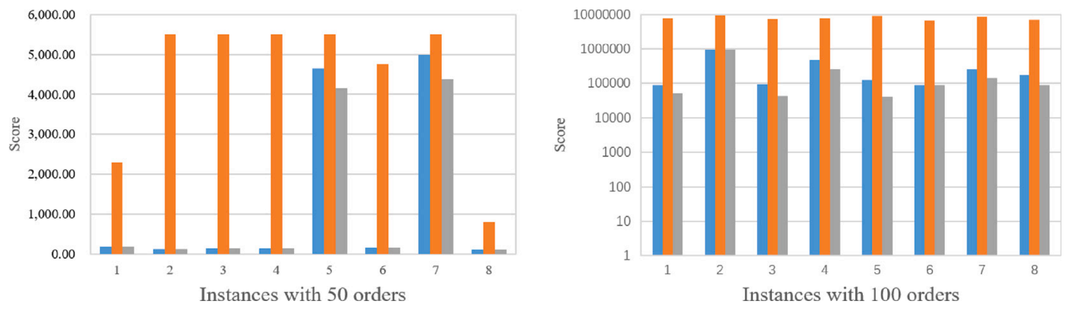

Fig. 11. Results for small scale ICAPS competition instances.

sets, comparing with two benchmarks. Then we conduct the quantitative sensitivity analysis for the performance of our proposed approach in Section 5.3.

5.1. *Test instances*

To our knowledge, there is no public benchmark available in the literature for the DPDP with LIFO constraints. To analyze the performance of the hierarchical optimization approach and local search, we use the instances originated from the competition data sets derived from the ICAPS 2021 DPDP competition (Hao et al., 2022), which contain order and location information. To test the solution approach in more general settings, we also randomly generate new instances to further derive insights of the proposed approach.

We use **DPDP-i-j-n** to name all instances. DPDP is the problem type, where indicates the order size, indicates the number of vehicles, and denotes the serial number. For example, there are 100 orders and five vehicles in the instance **DPDP-100-5-1**. Likewise, randomly generated instances are named **DPDP-R-i-j-n**. The computational experiment has the following attributes for each initial or generated instance.

Time horizon: T is the planning horizon (24 hours).

Time slot: Δt, is the time between consecutive decision epochs (e.g., every Δt minutes).

Vehicle capacity: since the vehicles are homogeneous, the vehicle capacity is a fixed value for each instance.

Number of vehicles: the total number of available vehicles.

Average number of orders per vehicle: the number of total orders divided by the number of vehicles, which is the ratio of demand and supply. It is abbreviated as *avg.order per vehicle*.

5.2. *Computational results*

This section presents the result of competition instances and randomly generated instances. We compare our proposed hierarchical optimization approach with two benchmark approaches, including (1) the baseline (a greedy approach), (2) the winning approach in the ICAPS 2021 competition (Hao et al., 2022).

Baseline: A greedy algorithm, where each new order is dispatched to the nearest available vehicle, is used to solve the DPDP, and the result is used as the baseline.

Ye and Liang's method (Ye and Liang 2022) (abbreviated Ye's method): This method obtained the best results out of all the competing methods in the ICAPS 2021 competition. It mainly works by waiting for enough orders and then assigning the set of orders to vehicles.

5.2.1. *Results for ICAPS 2021 competition instances*

In this section, we conduct the experiments for three approaches on the ICAPS competition instances. Note that a higher value of the objective function means higher cost in the following results.

(1) Small instances

There are 50 orders and 5 vehicles in the first instance sets that are named **DPDP-50-5-n** (n here is the instance identification number). We perform our experiments on eight small instances, and the results are as shown in Fig. 11 (A). The x-axis shows the instance number **n**, and the y-axis shows value of the objective function (Score).

Instances **DPDP-100-5-n** contain 100 orders and five vehicles. Fig. 11 (B) shows the objective function of the eight small instances solved by three different methods. In Fig. 11 (B), the logarithmic scale provides a better and more precise picture of the advantages and disadvantages of different methods.

In Figs. 11 (A) and 11 (B), the blue bar indicates the greedy approach, the yellow bar indicates the approach in competition, and the gray bar indicates the proposed hierarchical optimization approach. For all the instances with 50 or 100 orders, our approach obtains the best results.

(2) Mid-sized instances

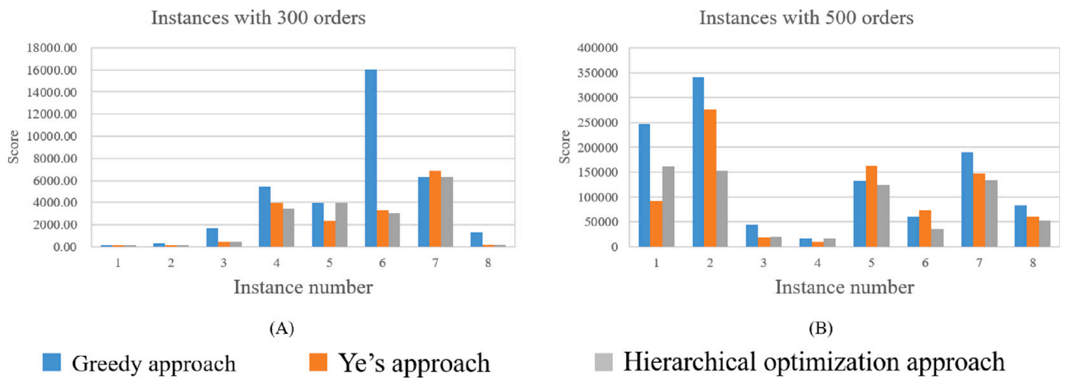

Fig. 12. Results for middle scale ICAPS competition instances.

Table 3
Results for large instances.

Instance	Greedy	Ye's method	Hierarchical	Improvement
DPDP-1000-50-1	3 365 810	30 136	**28 121**	6.69%
DPDP-1000-50-2	3 413 342	**22 252**	24 013	−7.91%
DPDP-1000-50-3	3 639 247	24 261	**23 338**	3.8%
DPDP-2000-50-1	81 679 118	**124 253**	126 292	−1.64%
DPDP-2000-50-2	87 302 720	190 604	**185 288**	2.79%
DPDP-2000-50-3	82 961 747	**181 170**	185 909	−2.62%
DPDP-3000-100-1	919 884 573	1 184 048	**1 082 139**	8.61%
DPDP-3000-100-2	911 166 177	1 629 783	**1 619 514**	0.63%
DPDP-3000-100-3	897 217 929	729 537	**678 125**	7.05%
Average				1.93%

Next, we experiment on two groups of mid-sized instances. One group of instances have 300 orders with 20 vehicles and the other group of instances contain 500 orders with 20 vehicles. For each group, we compare the solution and results, which are shown in Figs. 12 (A) and 12 (B).

In Fig. 12(A), we observe that the proposed hierarchical optimization approach obtains the best results in six out of the eight instances, while Ye's approach achieves the best results in two instances. The greedy approach gets two similar results as ours in fifth and the seventh instances. Nevertheless, the greedy approach is not stable, i.e., the value of the objective function is very large in instance 6. In Fig. 12(B), we observe that the hierarchical optimization approach obtains the best results for five out of eight instances. Ye and Liang's approach obtains the best result in three instances.

(3) Large instances

These are instances with the number of orders equal to or greater than 1000. We experiment on nine large instances with 1000 orders, 2000 orders, and 3000 orders, each having three different instances respectively. Table 3 shows the results of these experiments. In this table, column two to column four are the results for greedy, Ye's method, and our proposed hierarchical method. The last column show the improvement of our results compared with Ye's method, calculated by

$$\frac{result\ of\ Ye's\ method - result\ of\ Hierarchical}{result\ of\ Ye's\ method}$$

We observe that our hierarchical optimization approach is superior to a greedy approach and Ye and Liang's approach.

5.2.2. *Results for randomly generated instances*

Based on the analysis of the ICAPS 2021 competition instances, we found that the distribution of pickup and delivery points of the majority of the orders are concentrated in a small portion of all the locations. This enables specialized algorithms (such as Ye's method) to exploit to produce good solutions. In order to verify the generalizability of the various solution approaches, we generate two groups of random instances. These two groups are small (50 orders) and large (2000 orders) instances with randomly selecting pickup and delivery points for orders among all the points. These instances are named as **DPDP-R-i-j-n** which are available on GitHub here.[2]

We apply three approaches to solve these two groups (small and large) of generated instances. Fig. 13 shows the results for these randomly generated instances. In this figure, the bar charts represents the objective function values. Fig. 13(A) and (B) show that our proposed method has the best performance for all randomly generated instances with no matter 50 orders (small) instances or

[2] https://github.com/123zhangzq/TRE2023DPDPLIFO

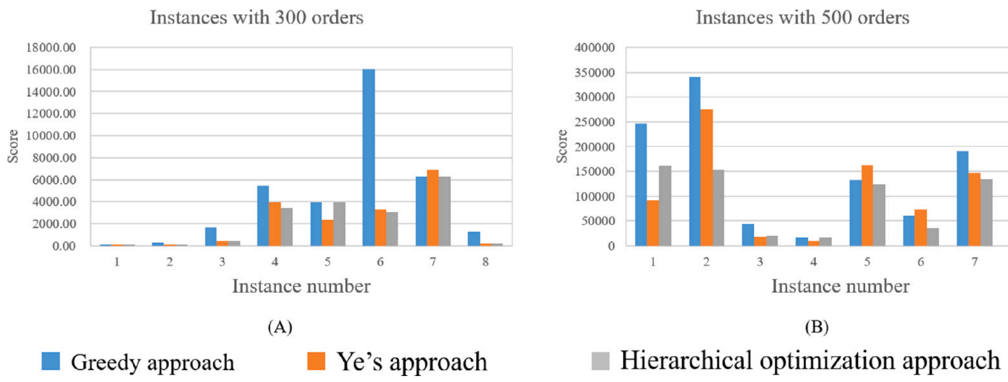

Fig. 13. Results for randomly generated instances.

2000 orders (large) instances. Although The Ye and Liang's approach outperforms the greedy approach for all large instances, which can be seen in Fig. 13(B), their approach is not stable for small instances, as shown in Fig. 13(A).

5.2.3. Discussion

For instances with different sizes, we observe that each approach may have different performance. Our proposed approach can achieve the best performance among the most instances. For small instances, the result of the greedy approach is better than Ye and Liang's approach, while the opposite is true for large instances. Specifically, for small instances, Figs. 11(A) and 11(B) shows the greedy approach and hierarchical approach can obtain a good result. However, Ye and Liang's approach cannot get a good solution due to large overtime penalty. For larger instances, Table 3 shows that Ye and Liang's method is better compared to the greedy approach as their method focusses on optimizing travel distance.

The performance of the proposed hierarchical approach can achieve best results among all random generated instances showed in Fig. 13. Therefore, our proposed approach has broader applicability than methods designed explicitly for ICAPS 2021 competition.

From the above computational results, we see that heuristic solution approaches may be sensitive for the feature of the problem instances, such as scale. When the size of the problem changes, some methods may become ineffective. Next section will show more sensitivity analysis on our proposed algorithm.

5.3. Sensitivity analysis

5.3.1. Run time analysis

In this section, we study the effect of problem size on the computation time of our proposed algorithm. We implement the algorithm in python and run the tests on a Windows personal computer of 64-bit with AMD Radeon R7-4800U.

Fig. 14 shows the average run time of our hierarchical optimization approach at each time slot when solving different sizes of instances. The horizontal and vertical axes represent the scale of the problem and the run time in seconds. It confirms our computational complexity analysis that the run time scales quadratically with the order volume. In absolute terms, we see that our proposed method is able to respond to new orders in 5 s when the number of orders is less than 500. Even though the run time increases with the problem scale, the average run time for one time slot is still less than 30 s, which is much smaller than the time slot itself (10 min). Even for the largest instances, the average run time is still acceptable for the implementation of the algorithm to the real industrial scale problems. The run time of our method can be further improved on more powerful machines.

5.3.2. Impact of time slot interval

In this section, we analyze the influence of the interval length of the time slot on the performance of our proposed solution approach. To quantitatively study this influence, we examine the impact of different time slots on objective function and travel distance.

Fig. 15 shows the trend of the result for different size instances as time slot changes, respectively. In this figure, the *x-axis* represents four different time slots including 2, 5, 10, and 15 min. The left *y-axis* shows the values for the bar chart, which represents the objective function for three different instances. The right *y-axis* is for line graph showing the travel distance cost.

For all these three instances, the bar chart shows that objective function values (including overtime penalty cost and travel distance cost) tend to increase with a longer time slot. However, the travel distance cost tends to decrease as the time slot increases, which can be seen in the line graph.

Managerial insight: The natural expectation is that the larger the time slot (decision epoch), the more information is available for planning, which results in the less total travel distance. On the other hand, a large time slot also means that fewer decisions are made (or late response for the dynamic orders) resulting in larger overtime cost. A small time slot means that we can be more responsive leading to a higher chance to identify matches between orders and vehicles. Therefore, in practice, managers should set the time slot based on the considerations from overtime penalty cost and travel distance cost in their specific problem.

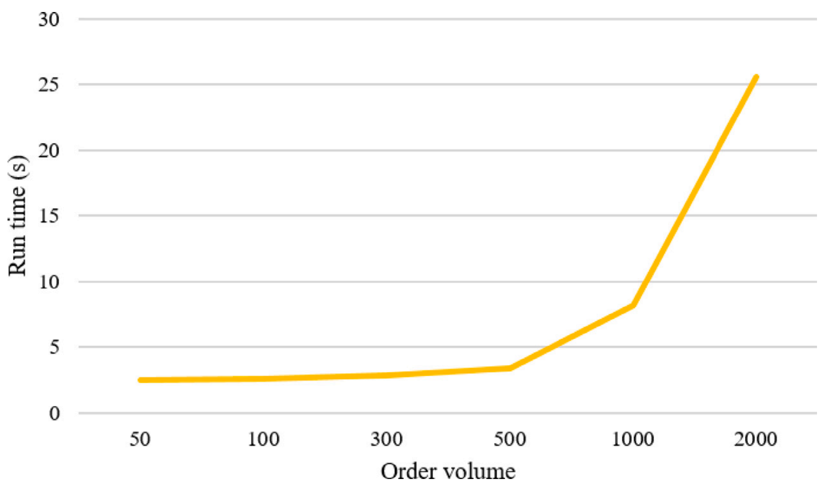

Fig. 14. Average run time per time slot.

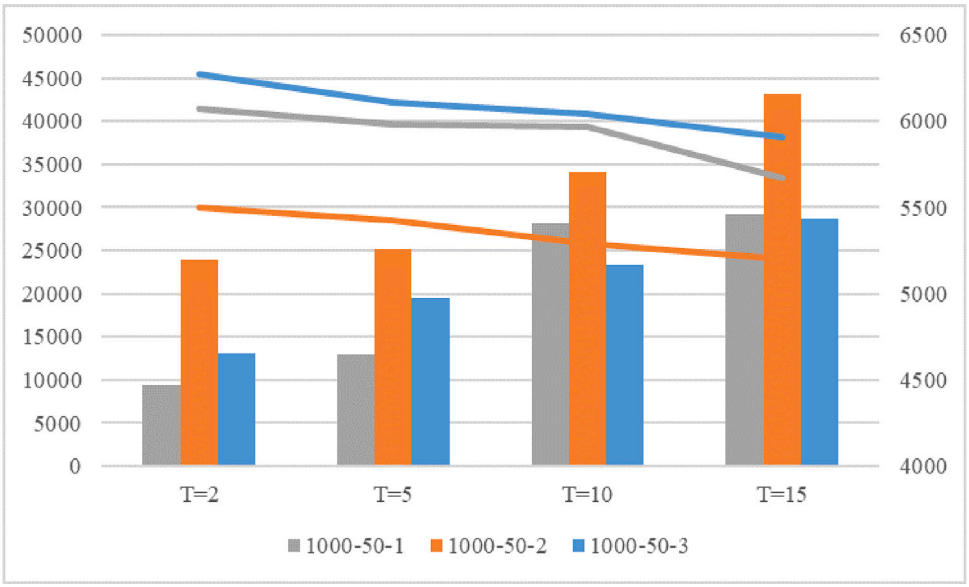

Fig. 15. Proposed algorithm performance under different time slot.

5.3.3. Impact of order distribution

Another impact on the performance of the proposed algorithm is the distribution of the orders in terms of location and time. As we have studied the former in Section 5.2.2, this section will investigate the latter (i.e. percentage of orders generated over time).

Here, we select two 50-order instances (DPDP-50-5-1 and DPDP-50-5-7) and two 300-order instances (DPDP-300-20-1 and DPDP-300-20-7) on which our proposed algorithm has the best and the worst performance in terms of objective value. The distribution of order arrival time is shown in Figs. 16 and 17. In these two figures, the horizontal axes are time horizon (a day), while the vertical axes are the distribution of the orders represented by the ratio of the number of the orders in a given one-hour period to the total number of orders. We compare the time distribution by counting the peak periods where the dynamic orders arrive in a short time. For example, from Fig. 16, we can see that the instance with the worst result has eight peaks with fluctuating distributions, whereas the instance with the best result has five peaks.

Hence, the distribution of orders over time has influence on the number of orders available for assignment in each decision epoch, which in turn affects the quality of solution generated by our solution approach.

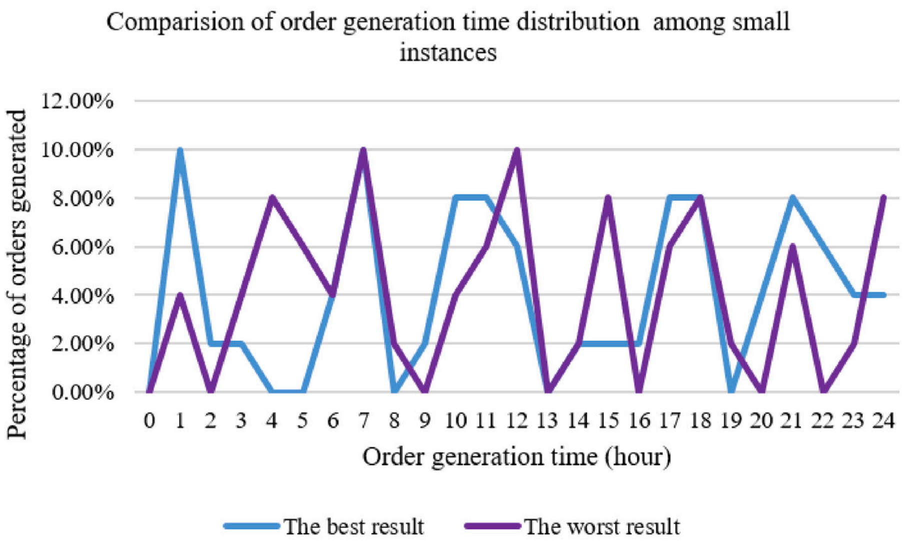

Fig. 16. Order distribution of small instances with the best and worst results.

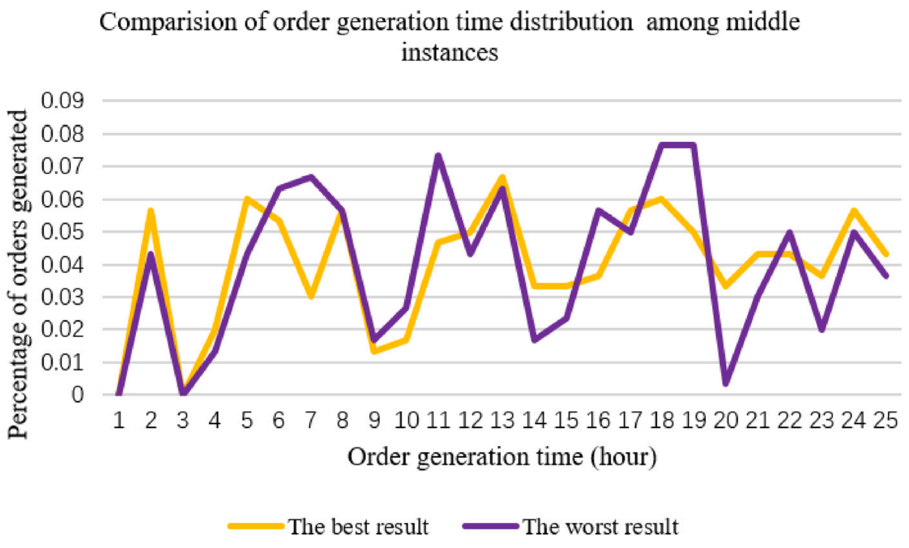

Fig. 17. Order distribution of middle instances with the best and worst results.

6. Conclusion

This paper proposed an effective hierarchical optimization approach for dynamic pickup and delivery problem (DPDP) with last in first out (LIFO) constraints. This work can be considered the first attempt to design hierarchical order dispatching strategies in DPDP. We design and define the urgent strategy, hitchhike strategy, and packing-bags strategy. The local search block-based operators can reduce the search space compared with operators designed for general routing problems. The results show that the proposed hierarchical optimization approach in this paper can effectively solve cases of different sizes, and the results are better than the winning approach on real world instances provided in an international programming competition. Moreover, we verify the wide applicability of the method proposed in this paper with randomly generated instances. Finally, we analyze the effect of the distribution pattern of orders on the results of instances and analyze the impact of time slot interval on DPDP.

There are many possibilities of extension from this work. First, in a real-world logistics system, the objective is more complicated, and involves soft constraints such as driver preferences (which is increasingly prevalent in a gig economy). While our problem can be extended from a bi-objective (purely based on cost) to a multi-objective setting involving other soft constraints, the problem is much more complicated and require a new approach to cope with generating a single solution for the actual assignment amidst the non-dominated solutions on the pareto front. Second, we have not included stochasticity in this problem. While there has been papers that deal with both dynamism and stochasticity in VRP (see for example the survey in Ritzinger et al. (2016)), many do not consider LIFO and other realistic constraints. Furthermore, many papers are model-based, which do not make use of historical data to address both the dynamic and stochastic aspects satisfactorily. We see a lot of potential in developing new (data-driven) models and algorithms to cope with such problems in view of the rising need in logistics planning.

CRediT authorship contribution statement

Jianhui Du: Methodology, Data curation, Implementation, Writing – original draft, Writing – review & editing. **Zhiqin Zhang:** Methodology, Data curation, Implementation, Writing – original draft, Writing – review & editing. **Xu Wang:** Supervision. **Hoong Chuin Lau:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision.

Acknowledgments

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-100E-2021-089). It was performed by the first author while he was on exchange at the Singapore Management University supported by the China Scholarship Council, China scholarship. The authors also appreciate the editor and the anonymous reviewers. All authors approved the version of the manuscript to be published.

References

Alyasiry, Ali Mehsin, Forbes, Michael, Bulmer, Michael, 2019. An exact algorithm for the pickup and delivery problem with time windows and last-in-first-out loading. Transp. Sci. 53 (6), 1695–1705.

Benavent, Enrique, Landete, Mercedes, Mota, Enrique, Tirado, Gregorio, 2015. The multiple vehicle pickup and delivery problem with LIFO constraints. European J. Oper. Res. 243 (3), 752–762.

Berbeglia, Gerardo, Cordeau, Jean-François, Gribkovskaia, Irina, Laporte, Gilbert, 2007. Static pickup and delivery problems: A classification scheme and survey. Top 15, 1–31.

Bombelli, Alessandro, Fazi, Stefano, 2022. The ground handler dock capacitated pickup and delivery problem with time windows: A collaborative framework for air cargo operations. Transp. Res. Part E: Logist. Transp. Rev. 159, 102603.

Braekers, Kris, Ramaekers, Katrien, Van Nieuwenhuyse, Inneke, 2016. The vehicle routing problem: State of the art classification and review. Comput. Ind. Eng. 99, 300–313.

Branke, Jürgen, Middendorf, Martin, Noeth, Guntram, Dessouky, Maged, 2005. Waiting strategies for dynamic vehicle routing. Transp. Sci. 39 (3), 298–312.

Carrabs, Francesco, Cordeau, Jean-François, Laporte, Gilbert, 2007. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS J. Comput. 19 (4), 618–632.

Chan, Nelson D., Shaheen, Susan A., 2012. Ridesharing in north America: Past, present, and future. Transp. Rev. 32 (1), 93–112.

Chen, Fangruo, 2000. Optimal policies for multi-echelon inventory problems with batch ordering. Oper. Res. 48 (3), 376–389.

Cherkesly, Marilène, Desaulniers, Guy, Laporte, Gilbert, 2015. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. Transp. Sci. 49 (4), 752–766.

Christofides, Nicos, 1976. The vehicle routing problem. Revue Française d'automatique Informatique, Recherche Opérationnelle Recherche Opérationnelle 10 (V1), 55–70.

Cordeau, Jean-François, Iori, Manuel, Laporte, Gilbert, Salazar González, Juan José, 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. Networks 55 (1), 46–59.

Dayarian, Iman, Savelsbergh, Martin, 2020. Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. Prod. Oper. Manage. 29 (9), 2153–2174.

Desrochers, Martin, Desrosiers, Jacques, Solomon, Marius, 1992. A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. 40 (2), 342–354.

Dumas, Yvan, Desrosiers, Jacques, Soumis, Francois, 1991. The pickup and delivery problem with time windows. European J. Oper. Res. 54 (1), 7–22.

Eksioglu, Burak, Vural, Arif Volkan, Reisman, Arnold, 2009. The vehicle routing problem: A taxonomic review. Comput. Ind. Eng. 57 (4), 1472–1483.

Erdoğan, Sevgi, Miller-Hooks, Elise, 2012. A green vehicle routing problem. Transp. Res. Part E: Logist. Transp. Rev. 48 (1), 100–114.

Farazi, Nahid Parvez, Zou, Bo, Tulabandhula, Theja, 2022. Dynamic on-demand crowdshipping using constrained and heuristics-embedded double dueling deep Q-network. Transp. Res. Part E: Logist. Transp. Rev. 166, 102890.

Gholami-Zanjani, Seyed Mohammad, Jafari-Marandi, Ruholla, Pishvaee, Mir Saman, Klibi, Walid, 2019. Dynamic vehicle routing problem with cooperative strategy in disaster relief. Int. J. Ship. Transp. Logist. 11 (6), 455–475.

Häll, Carl H., Lundgren, Jan T., Voss, Stefan, 2015. Evaluating the performance of a dial-a-ride service using simulation. Public Transp. 7, 139–157.

Hamdan, Bayan, Diabat, Ali, 2019. A two-stage multi-echelon stochastic blood supply chain problem. Comput. Oper. Res. 101, 130–143.

Hao, Jianye, Lu, Jiawen, Li, Xijun, Tong, Xialiang, Xiang, Xiang, Yuan, Mingxuan, Zhuo, Hankz Hankui, 2022. Introduction to the dynamic pickup and delivery problem benchmark–ICAPS 2021 competition. arXiv preprint arXiv:2202.01256.

Harbaoui Dridi, Imen, Ben Alaïa, Essia, Borne, Pierre, Bouchriha, Hanen, 2020. Optimisation of the multi-depots pick-up and delivery problems with time windows and multi-vehicles using PSO algorithm. Int. J. Prod. Res. 58 (14), 4201–4214.

Iori, Manuel, Martello, Silvano, 2010. Routing problems with loading constraints. Top 18 (1), 4–27.

Laporte, Gilbert, 1992. The vehicle routing problem: An overview of exact and approximate algorithms. European J. Oper. Res. 59 (3), 345–358.

Lau, Hoong Chuin, Liang, Zhe, 2001. Pickup and delivery with time windows: Algorithms and test case generation. In: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001, pp. 333–340.

Lau, Hoong Chuin, Sim, Melvyn, Teo, Kwong Meng, 2003. Vehicle routing problem with time windows and a limited number of vehicles. European J. Oper. Res. 148 (3), 559–569.

Li, Xijun, Luo, Weilin, Yuan, Mingxuan, Wang, Jun, Lu, Jiawen, Wang, Jie, Lü, Jinhu, Zeng, Jia, 2021. Learning to optimize industry-scale dynamic pickup and delivery problems. In: 2021 IEEE 37th International Conference on Data Engineering. ICDE, IEEE, pp. 2511–2522.

Ma, Yi, Hao, Xiaotian, Hao, Jianye, Lu, Jiawen, Liu, Xing, Xialiang, Tong, Yuan, Mingxuan, Li, Zhigang, Tang, Jie, Meng, Zhaopeng, 2021. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. Adv. Neural Inf. Process. Syst. 34, 23609–23620.

Mitrović-Minić, Snežana, Laporte, Gilbert, 2004. Waiting strategies for the dynamic pickup and delivery problem with time windows. Transp. Res. B 38 (7), 635–655.

Mojtahedi, Mohammad, Fathollahi-Fard, Amir M, Tavakkoli-Moghaddam, Reza, Newton, Sidney, 2021. Sustainable vehicle routing problem for coordinated solid waste management. J. Ind. Inform. Integr. 23, 100220.

Nazari, Mohammadreza, Oroojlooy, Afshin, Snyder, Lawrence, Takác, Martin, 2018. Reinforcement learning for solving the vehicle routing problem. Adv. Neural Inf. Process. Syst. 31.

Pillac, Victor, Gendreau, Michel, Guéret, Christelle, Medaglia, Andrés L, 2013. A review of dynamic vehicle routing problems. European J. Oper. Res. 225 (1), 1–11.

Powell, Warren B., 2011. Approximate Dynamic Programming: Solving the Curses of Dimensionality, Vol. 842. John Wiley & Sons.

Prasanti, Nissa, Sentia, Prima Denny, et al., 2018. Pickup and delivery problem with LIFO, time duration, and limited vehicle number. In: MATEC Web of Conferences, Vol. 204. EDP Sciences, p. 07003.

Pureza, Vitória, Laporte, Gilbert, 2008. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. INFOR: Inform. Syst. Oper. Res. 46 (3), 165–175.

Ritzinger, Ulrike, Puchinger, Jakob, Hartl, Richard F., 2016. A survey on dynamic and stochastic vehicle routing problems. Int. J. Prod. Res. 54 (1), 215–231.

Ropke, Stefan, Cordeau, Jean-François, 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transp. Sci. 43 (3), 267–286.

Ruiz, Efrain, Soto-Mendoza, Valeria, Barbosa, Alvaro Ernesto Ruiz, Reyes, Ricardo, 2019. Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm. Comput. Ind. Eng. 133, 207–219.

Sadati, Mir Ehsan Hesam, Akbari, Vahid, Çatay, Bülent, 2022. Electric vehicle routing problem with flexible deliveries. Int. J. Prod. Res. 60 (13), 4268–4294.

Savelsbergh, Martin W.P., Sol, Marc, 1995. The general pickup and delivery problem. Transp. Sci. 29 (1), 17–29.

Schrage, Linus, 1981. Formulation and structure of more complex/realistic routing and scheduling problems. Networks 11 (2), 229–232.

Spliet, Remy, Dabia, Said, Van Woensel, Tom, 2018. The time window assignment vehicle routing problem with time-dependent travel times. Transp. Sci. 52 (2), 261–276.

Toth, Paolo, Vigo, Daniele, 2002. VRP with backhauls. In: The Vehicle Routing Problem. SIAM, pp. 195–224.

Vidal, Thibaut, Laporte, Gilbert, Matl, Piotr, 2020. A concise guide to existing and emerging vehicle routing problem variants. European J. Oper. Res. 286 (2), 401–416.

Xu, Xiaofeng, Wei, Zhifei, 2023. Dynamic pickup and delivery problem with transshipments and LIFO constraints. Comput. Ind. Eng. 175, 108835.

Zhu, Zexuan, Xiao, Jun, He, Shan, Ji, Zhen, Sun, Yiwen, 2016. A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem. Inform. Sci. 329, 73–89.