Computers & Operations Research 159 (2023) 106337

Contents lists available at ScienceDirect

Computers and Operations Research

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

Disruption recovery for the pickup and delivery problem with time windows—A scenario-based approach for online food delivery

Yuzhen Hu [a,*], Pu Zhang [a], Kang Zhao [b], Song Zhang [a], Bo Fan [c]

[a] School of Economics and Management, Harbin Engineering University, China
[b] Department of Business Analytics, The University of Iowa, Iowa City, IA 52242, USA
[c] School of International and Public Affairs, Shanghai Jiaotong University, Shanghai 200030, China

ARTICLE INFO

Keywords:
Disruption management
PDPTW
Scenario-based approach
Adaptive large neighborhood search heuristic

ABSTRACT

Recently, the pickup and delivery problem has attracted increasing attention due to its applications in online food delivery services. At the same time, unforeseen events, such as vehicle breakdowns, traffic jam, and service time changes, often lead to the delay of delivery services. This paper defines the disruption recovery problem for the pickup and delivery problem with time windows (DR-PDPTW) and presents a scenario-based approach to solve the problem in a computationally efficient way. The approach starts with pre-disruption preparation that generates disruption scenarios and corresponding candidate recovery solutions. Once a disruption occurs, the approach will first check if recovery efforts are necessary based on the magnitude of delays caused by the disruption. If such delays are not acceptable, the approach will move to the post-disruption response stage to match the disruption with an already generated disruption scenario. Then corresponding candidate recovery solutions will be adjusted based on the actual disruption and executed. Computation experiments demonstrate the quality and efficiency of the proposed approach. This study can help to provide real-time decision support for disruption management in online food delivery services.

1. Introduction

The pickup and delivery problem with time windows (PDPTW) aims at finding routes to transport goods from their origins to destinations within a time window and minimize the travel cost. Many studies have proposed models and methods to obtain the optimal or near optimal solutions for this type of problem with static or real-time demand (Berbeglia et al. 2007; Berbeglia et al., 2010; Koç et al. 2020). Recently, the problem has drawn increased attention with the rising popularity of online food delivery services (e.g., ELEME and Meituan in China, and DoorDash and Grubhub in the U.S.), especially during the COVID-19 pandemic. When an order that requests food is placed, the request should be delivered by a single vehicle from the restaurant (i.e., the pickup node) to the customer (i.e., the delivery node). In addition, a vehicle may pick up multiple orders at the same restaurant, especially during peak hours (e.g., lunch or dinner time). As a result, the vehicle may drop off one order on its way to deliver another order.

However, during the process, unforeseen disruptions could happen, such as breakdown of the delivery vehicle, traffic jams, service time change at the pickup node (e.g., postponed food preparation), or the customer's delay in receiving the order at the delivery node. When a disruption occurs, the originally planned route may no longer be optimal or even becomes infeasible. Thus, the delivery vehicle should be rerouted as quickly as possible to minimize the negative impact to customer experience, which may hurt the image and profit of the company. Such rerouting after disruptions is referred to as Disruption Recovery for PDPTW (DR-PDPTW).

After a disruption occurs, one possible solution is to re-optimize all routes to minimize the total traveling cost of the delivery vehicle fleet using dynamic PDPTW. However, such an approach ignores three important characteristics of online food delivery services.

First, besides the overall travelling cost, a delivery platform needs to consider the benefits of its delivery drivers as well. Because the income for a driver depends heavily on the number of completed deliveries, a driver is usually unwilling to give up an order she is already carrying unless the vehicle cannot resume normal operations any time soon. In fact, due to recent public debates in China about how delivery drivers have been abused and exploited by online food delivery platforms (Lai,

* Corresponding author.
E-mail addresses: yuzhenhu@hrbeu.edu.cn (Y. Hu), darer@hrbeu.edu.cn (P. Zhang), kang-zhao@uiowa.edu (K. Zhao), zhangsong@hrbeu.edu.cn (S. Zhang), fanbo@sjtu.edu.cn (B. Fan).

https://doi.org/10.1016/j.cor.2023.106337
Received 13 October 2022; Received in revised form 22 June 2023; Accepted 22 June 2023
Available online 26 June 2023

2020), it becomes more urgent for platforms to consider drivers' satisfaction for public relation and driver retention purposes.

Second, a dynamic PDPTW approach can potentially cause problems related to logistics and customer satisfaction in food delivery. On one hand, transferring food from one driver to another halfway requires clear communication and careful coordination between drivers. These are challenging in an everchanging urban environment, especially when a driver carries multiple orders that need to be transferred to different drivers. On the other hand, minimizing the total travelling cost for all existing order may lead to increased wait time for certain customers, who already have been given expected arrival time soon after they place orders. Extended delays in such cases could significantly hurt customer satisfaction.

Third, a dynamic PDPTW approach to re-optimize all routes will incur higher computation and longer time, while finding recovery solutions needs a shorter response time soon after the disruption occurrence.

Therefore, we believe that DR-PDPTW needs to consider deviations from original plans, and simply applying dynamic PDPTW approaches to this problem is not sufficient. To address this challenge, this paper proposes a novel scenario-based approach for DR-PDPTW. We provide an efficient and accurate decision framework for PDPTW service platforms, such as those for food delivery, to deal with disruptions. Overall, the paper's contributions are threefold:

(1) By defining the novel problem of DR-PDPTW, our work represents the first attempt to handle disruption recovery for the PDPTW problem. In DR-PDPTW, we consider simultaneous occurrence of four different types of disruptions, including the breakdown of delivery vehicles, traffic jams, service time changes at pickup nodes and customers' delays in receiving orders at delivery nodes.
(2) We design a new scenario-based framework to obtain recovery solutions in a computationally efficient way for DR-PDPTW. Different from existing methods that directly identify alternative solutions, the framework prepares a set of disruption scenarios with corresponding recovery solutions in advance. Once a disruption occurs and needs a recovery solution, the framework will choose the most similar one from prepared disruptions, and adjust the chosen solution to better handle the current disruption.
(3) We adopt and design a set of algorithms to expedite the computation for DR-PDPTW solutions. First, we extend the ALNS by outputting multiple recovery solutions, adding a new operator and designing a routing selection algorithm to efficiently generate recovery solutions for prepared disruption scenarios. Then, we propose heuristics based on clustering algorithms to quickly and accurately match prepared scenarios with an actual disruption. Finally, we develop an adjustment algorithm to improve the prepared recovery solution so it can better handle the actual disruption. All these algorithms play important roles in improving the efficiency and efficacy of the proposed framework.

The remainder of this paper is organized as follows. Section 2 reviews related literature and highlights the contributions of the study. The problem description and mathematical formulation are represented in Section 3. The proposed solution methodology, along with necessity analysis of recovery efforts and several algorithms, are introduced in Section 4, and Section 5 reports results from computational experiments. Section 6 concludes the study and discusses possible future research directions.

2. Related work

This paper is related to research in two areas that we will review in this section.

2.1. Disruption management in vehicle routing

One of the major goals of disruption management is to revise the original operation schedule and obtain a reasonable and feasible alternative solution, often in real time (Yu and Qi 2004). Disruption management has important applications in various transportation and logistics systems, such as airlines (Hassan et al., 2021; Hu et al. 2021). Additionally, disruption management has been studied for the scheduling of equipment (Xiong et al. 2018), projects (Burgelman and Vanhoucke 2020) and productions (Sawik 2016), as well as supply chain optimization (Katsaliaki et al. 2022) and vehicle routing (Giménez-Palacios et al. 2022).

DR-PDPTW is a type of disruption management for vehicle routing (Eglese and Zambirinis 2018). While most existing papers mainly focus on vehicle routing disruptions caused by vehicle breakdowns (Li et al. 2009), there are many other types of disruptions, such as road link disruptions (Ngai et al. 2012), supply disruptions (Mu and Eglese 2013), and customer demand disruptions (Spliet et al. 2014). Thus some research on vehicle routing disruptions explored the recovery of disruptions of multiple types. For example, Wang et al. (2012) constructed an optimization model with three objectives for a recovery problem with time windows after disruptions to customers, drivers, and logistics providers. The problem can be solved with the Nested Partition Method. Similarly, Hu et al. (2013) introduced a knowledge-based method to recover from disruptions of multiple types.

Despite the abundance of research on disruptions for vehicle routing, few studies have investigated the DR-PDPTW problem with time windows, mainly due to its unique characteristics that we will outline below.

2.2. Pickup and delivery problem with time windows

The standard PDPTW can be formulated in different ways, such as the three-index formulation (Solomon and Desrosiers 1988, Dumas et al. 1991), the two-index formulation (Furtado et al. 2017). Based on the problem formulations, different algorithms have been developed, such as the branch-and-cut-and-price algorithm (Ropke and Cordeau 2009), the branch-price-and-cut algorithm (Li et al. 2019) and branch and price algorithm (Sun et al. 2018). Nevertheless, for an NP-hard problem like PDPTW, these methods can only handle small-scale problems. As a result, heuristic algorithms have also been adopted, including tabu search (Nanry and Barnes 2000), large neighborhood search (Wolfinger 2021), genetic algorithms (Jun et al. 2021), and adaptive large neighborhood search (ALNS) (Ropke and Pisinger 2006). However, the current state-of-the-art for PDPTW (Christiaens and Vanden Berghe, 2020), a variant of LNS, takes more than 3 min to run when the number of requests is merely 500 on a well-equipped computer workstation (v2 CPU at 2.60 GHz), even after extensive parameter tuning. Such a computational speed is still far from the requirement for DR-PDPTW, especially when there may be tens of thousands of nodes operating at the same time, often with tight time windows, and any delay in generating an alternative solution may lead to a delayed delivery.

Because disruptions happen unexpectedly and delivery vehicles often need to be rerouted thereafter, the DR-PDPTW problem is especially related to online (Jaillet and Wagner 2008, Kai et al. 2004) or dynamic (Karami et al. 2020) PDPTW problems. Existing online or dynamic PDPTW methods mainly focus on requests arriving in real-time and update vehicle routing by minimizing traveling time (Pillac et al. 2012). In online PDPTW, the request information can be predicted by learning. In dynamic PDPTW, the request information is unknown in advance, which is more similar with DR-PDPTW. There are two common approaches to solve dynamic PDPTW: reactive and periodic. Reactive methods re-optimize the routing plan once a delivery is made (Schyns 2015, Vancroonenburg et al. 2016), while periodic methods fully update routes when a certain criterion (e.g., time interval) is met (Karami et al. 2020). It is apparent that reactive methods react faster and start to handle a disruption sooner than periodic methods. Previous research also demonstrated that reactive approaches outperform periodic approaches when service quality is the priority (Vancroonenburg et al. 2016). However, with a full re-optimization of all routes after each delivery (or after each disruption in the case of disruption recovery), reactive methods inevitably incur higher computational cost.

Adding stochasticity into consideration, Bent and Van Hentenryck (2004) presented the multiple scenario algorithm (MSA) strategy for solving dynamic PDPTW with stochastic customer demands\mathord{-} the strategy first generates multiple scenarios of routing plans composed of known and future stochastic information on dynamic customers and then chooses one routing plan scenario at each decision. Nevertheless, previous MSA methods for vehicle routing (Ghiani et al. 2012) and the same day PDPTW (Voccia et al. 2019) only deal with stochastic request demand and are not suitable for DR-PDPTW with multiple unpredictable disruptions. Moreover, the objective of DR-PDPTW is to minimize the effect of the disruption on vehicle routing and customer' demand, whereas PDPTW is only about minimizing traveling time. Thus existing MSA methods cannot be directly applied to DR-PDPTW.

To solve DR-PDPTW more efficiently, we provide a scenario-based solution framework. The framework is able to obtain the optimal solution for DR-PDPTW more than 5 times faster than ALNS. Our framework consists of two stages: the pre-disruption preparation stage and the post-disruption response stage. In the first stage, we prepare a set of disruptions and generates candidate recovery solutions for each prepared disruption scenario. Specifically, we extend the ALNS algorithm (Ropke and Pisinger 2006) to obtain candidate recovery solutions more efficiently by designing a routing selection algorithm, adding pickup-oriented removal operator and keeping multiple recovery solutions. In the second stage, once it is determined that an actual disruption needs a recovery response, we will match the current disruption with prepared disruption scenarios that have been grouped with clustering algorithms. After finding the prepared disruption scenario that is most similar to the current disruption, we also adjust the corresponding candidate recovery solutions and choose the best one to fit the current disruption.

3. Problem description and model formulation for DR-PDPTW

In practice, food requests always arrive in real time and a service platform needs to update the routing schedules for its vehicle fleet accordingly. Such routing schedules can be defined as original planning for DR_PDPTW. When a disruption occurs, the set of unfinished delivery requests R, delivery vehicles K and corresponding routing schedules are deterministic. We can use a directed graph $G=(V, A)$ to describe the delivery requests and routing schedules. In this graph, $V = P \cup D \cup \{0, m+n+1\}$, where set P ($|P|=m$) denotes unserved pickup nodes, and set D ($|D|=n$) denotes unserved delivery nodes for unfinished requests. The starting and ending node of all vehicle routes are assumed to be different for the purpose of simplicity. Each request $r \in R$ is composed of one pickup node $p_r \in P$ and one delivery node $d_r \in D$. The pickup node p_r must be visited before the delivery node d_r for each request $r \in R$. Moreover, for each request r, the pickup node $i = p_r$ has a service time s_{ri} and a time window $[a_{ri}, b_{ri}]$, during which the order needs to be picked up. Similarly, the delivery node $j = d_r$ has a service time s_{rj} and a time window $[a_{rj}, b_{rj}]$, during which the order is expected to be delivered. In G, set A includes arcs between nodes $i, j \in V$. Let $A^+(i)$ be the set of nodes that can be reached from node $i \in V$ via an arc, and $A-(i)$ be the set of nodes that can reach node $i \in V$ via an arc. Each arc $(i,j) \in A$ is associated with travelling cost c_{ij} and travelling time t_{ij}.

As we mentioned earlier, online food delivery has one unique characteristic that is different from previous PDPTW\mathord{-} one pickup node $i \in P$ may have more than one request (e.g., a restaurant receives multiple orders). In other words, two orders $r, r' \in R$ may share the same pickup node (i.e., $p_r = p_{r'}$) but their delivery time windows can be different and even overlapping. Thus we define $R(i)$ as the set of requests related to the node $i \in P \cup D$. For a delivery node $i \in D$, the number of elements in $R(i)$ is mostly 1 (i.e., $|R(i)| = 1$) because a customer usually orders from one restaurant at one time, but for a pickup node $i \in P$, it is common that $|R(i)| > 1$ as a restaurant receives multiple orders.

In this paper, we consider four types of disruptions: vehicle breakdown, traffic jams, and service time changes in pickup nodes and in delivery nodes. Food orders are often delivered via e-bikes in Chinese cities. Minor breakdowns of e-bikes could occur after flat tires, low batteries or traffic accidents. Traffic jams are common as well, especially during rush hours. Service time changes in pickup nodes are often the results of delayed food preparation in restaurants. Service time changes in delivery nodes occur when customers are late in receiving the order. Overall, the length of disruptions studied in this paper is usually within one hour. Additionally, we also assume that the disruption length can be identified when the disruption occurs, a common assumption that has been adopted by previous studies (Yu and Qi 2004, Burgelman and Vanhoucke 2020).

For the purpose of simplicity, after a vehicle breakdown, we assign the position of the affected driver to the vehicle location right before the disruption. After disrupted vehicle k is fixed, the time when the vehicle is ready to operate again can be denoted as τ_k. It is apparent that the vehicle k cannot continue to serve for the requests until the time τ_k. Similarly, traffic jams, service time changes at pickup nodes and customers' delays will corresponds to the values of t_{ij} and s_{ri} respectively. Base on degree of the request completion, service requests in R can be further divided into two sets: $R = R' \cup R''$, $R' \cap R'' = \varnothing$. For each request $r \in R'$, the pickup node has been visited but the delivery node has not been served, i.e., $p_r \notin P$ and $d_r \in D$. For each $r \in R''$, both pickup nodes and delivery nodes have not been served, i.e., $p_r \in P$, and $d_r \in D$. During the disruption recovery process, new arriving requests can be considered as unfinished ones and added into the set of R''.

Fig. 1 illustrates an example with three requests (r_1, r_2, and r_3) and a vehicle's routing schedules for the requests. Requests r_2 and r_3 share the same pickup node p_2. The pickup service time windows for r_2 and r_3 at p_2 are different but partially overlap. There are three delivery nodes (d_1, d_2 and d_3) for the three requests. When a disruption occurs, the vehicle has served pickup node p_1 for r_1, thus the recovery graph only include node p_2, d_1, d_2 and d_3 (in the red box). During the recovery process, the vehicle may pick up food for both r_2 and r_3 from p_2 simultaneously, and then deliver them to d_1, d_2 and d_3 successively. If a new request arrive during the recovery process, it will be added into set R'' and dispatched to drivers after request r_3 is finished. Such a new request will not affect our analysis in this paper.

The recovery solution to DR-PDPTW is a set of feasible vehicle routings by reassigning unfinished requests to vehicles during in a short time period (Yu and Qi 2004). The solution should minimize the total recovery cost, which consists of three parts: the first part is the total travelling cost of the unfinished requests. The other two parts reflect the total deviation cost of the recovery solution from the routing schedules before disruption recovery, and they are the main differences between DR-PDPTW and PDPTW.

For the traveling cost, we define a decision variable y^k_{ij}.$y^k_{ij} = 1$ if arc $(i, j) \in A$ is travelled by vehicle $k \in K$; 0 otherwise. We set traveling cost c_{ij} for each $(i,j) \in A$. It is noted that y^k_{ij} is only associated with uncompleted routes when the disruption occurs, thus we can evaluate the total traveling cost for unfinished requests.

The first type of deviation is the reassignment of the disrupted service request to other vehicles instead of original vehicles. Because the disruption may have made the original assignment non-optimal and even infeasible, disruption management aims at obtaining a feasible reassignment at the lowest deviation, instead of re-optimizing it completely (Yu and Qi 2004). In practice, both platforms and drivers are motivated to decrease the number of requests violating the original vehicle assignment as much as possible. Platforms arrange the drivers to the routes they are familiar with to improve delivery efficiency. Moreover, most drivers want to finish as many service requests already

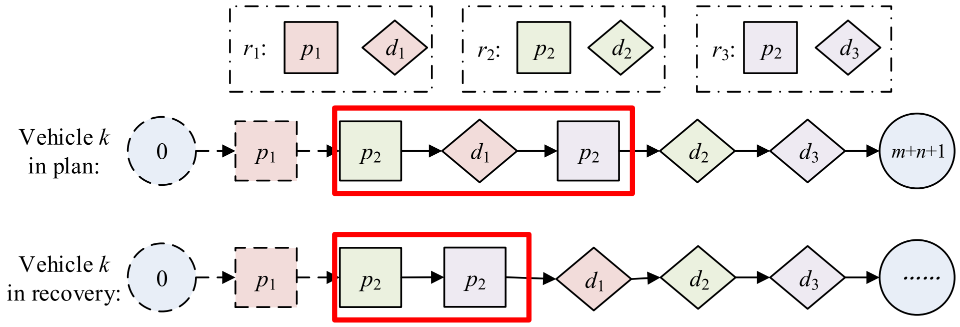

Fig. 1. An example routing of vehicle k when some disruption occurs.

assigned to them as possible, instead of transferring them to other drivers, because finishing order deliveries is their main source of income. Therefore, we set the decision variable x_{rk}. $x_{rk} = 1$ represents that request r is served by the vehicle k in the recovery solution, and $x_{rk} = 0$ otherwise. Moreover, we use e_{rk} to represent the deviation cost of reassigning request r to vehicle k, which violates the original assignment. Note that $e_{rk} = 0$ for the request r which is still served by the original vehicle k_r, i.e. $k = k_r$. Please also note that the deviation cost is zero for the request r that is already picked up by the vehicle $k = k_r$ at the restaurant (pickup node), i.e., $e_{rk} = 0$ for $r \in R'$.

The second type of deviation is the change in service time compared to the original schedule. After a disruption, the service starting time may go beyond the original time windows, and such a violation leads to a penalty. According to Hu et al. (2013), each customer at a delivery node has a maximum tolerance for such time violations. Therefore, we set the time tolerance as δ_j for delivery node $j = d_r$, $r \in R$. Then the time deviation cost can be defined as in (1), where T_{rj} is a decision variable and represents the service beginning time at node j for request r.

$$z_{rj} = \begin{cases} 0 & T_{rj} \leqslant b_{rj} \\ (T_{rj} - b_{rj})/\delta_j & T_{rj} > b_{rj} \end{cases} \forall r \in R, j = d_r \tag{1}$$

Then DR-PDPTW can be formulated as the following optimization problem:

$$\min F(x, y, z) = \sum_{k\in K}\sum_{(i,j)\in A} c_{ij} y_{ij}^k + \sum_{k\in K}\sum_{r\in R''} e_{rk} x_{rk} + \sum_{r\in R}\sum_{j=d_r} z_{rj} \tag{2}$$

Objective function defined in (2) aims to minimize the total traveling cost and total deviation cost from original schedules. Constraints for the optimization problem are listed below.

$$\sum_{i\in A^+(0)} y_{0i}^k = 1 \forall k \in K \tag{3}$$

$$\sum_{i\in A^-(m+n+1)} y_{i,m+n+1}^k = 1 \forall k \in K \tag{4}$$

$$\sum_{j\in A^+(i)} y_{ij}^k = x_{rk} i \in P \cup D, r \in R(i), k \in K \tag{5}$$

$$\sum_{j\in A^-(i)} y_{ji}^k = x_{rk} i \in P \cup D, r \in R(i), k \in K \tag{6}$$

$$\sum_{k\in K} x_{rk} = 1 \forall r \in R \tag{7}$$

$$x_{rk} = 1 r \in R', k = k_r \tag{8}$$

$$T_{r'j} + M(1 - y_{ij}^k) \geqslant T_{ri} + s_{ri} + t_{ij} \forall k \in K, (i,j) \in A, r \in R(i), r' \in R(j) \tag{9}$$

$$T_{rj} \geqslant T_{ri} + s_{ri} + t_{ij} \forall r \in R'', i = p_r, j = d_r \tag{10}$$

$$T_{rj} \geqslant a_{rj}, \forall j \in P \cup D, r \in R(j) \tag{11}$$

$$T_{rj} \geqslant \tau_k x_{rk}, \forall k \in K, j \in P \cup D, r \in R(j) \tag{12}$$

$$z_{rj} \geqslant 0 \forall r \in R, j = d_r \tag{13}$$

$$z_{rj} \geqslant (T_{rj} - b_{rj})/\delta_j \forall r \in R, j = d_r \tag{14}$$

$$x_{rk}, y_{ij}^k \in \{0, 1\} \forall (i,j) \in A, r \in R, k \in K \tag{15}$$

$$T_{ri} \geqslant 0 \forall i \in V, r \in R(i) \tag{16}$$

In this model, constraints in (3) and (4) ensure the flow balance of each vehicle at initial node 0 and at the last node m + n + 1. Constraints in (5) – (8) ensure both the pickup and delivery node are served by the same vehicle. If a request has not been picked up (i.e., the pickup node and delivery node of the request are both unserved) when a disruption occurs, then the assigned vehicle in the recovery solution may not be the same as in the original plan. If a request has been picked up before disruption occurrence but has not been delivered, then the delivery node of the request should be served by the original vehicle. Constraints in (9) ensure the time sequence of visiting two adjacent nodes in one vehicle routing. M is a big enough number to guarantee the linear characteristics of the constraints in (9). Constraints in (10) ensure that the pickup node will be traversed before the delivery node. Constraints in (11) and (12) define the new time τ_k for starting time of the service in the recovery solution, beside the original starting time a_{rj}. Constraints (13) and (14) are equivalent to formulation (1), but the new formulation implies that the recovery model is linear and can be solved by the commercial CPLEX Solver. Constraints (15) and (16) define decision variables.

4. The proposed scenario-based solution approach

As we discussed before, DR-PDPTW is a dynamic decision process—once a route is disrupted, the recovery solution should be obtained in real time. Note that a recovery solution can be disrupted again and needs another round of recovery efforts, which further complicates the issue. Thus this paper proposes a scenario-based method to provide real-time recovery solutions, which cannot be efficiently obtained via traditional disruption recovery methods based on optimization models and algorithms.

Fig. 2 illustrates the framework of the proposed scenario-based solution method, which consists of a pre-disruption stage and a post-disruption stage. Before any disruption occurs, based on an optimal planning solution of PDPTW, disruption scenarios are generated, and corresponding recovery solutions are obtained by an adaptive large neighborhood search heuristic (ALNS). These solutions are only used as candidate solutions for an actual disruption in the future. When a disruption actually occurs, the framework first checks if recovery efforts are necessary. If so, the matching disruption scenario from pre-generated scenarios and corresponding candidate solutions are chosen and then adjusted to better fit the actual disruption.

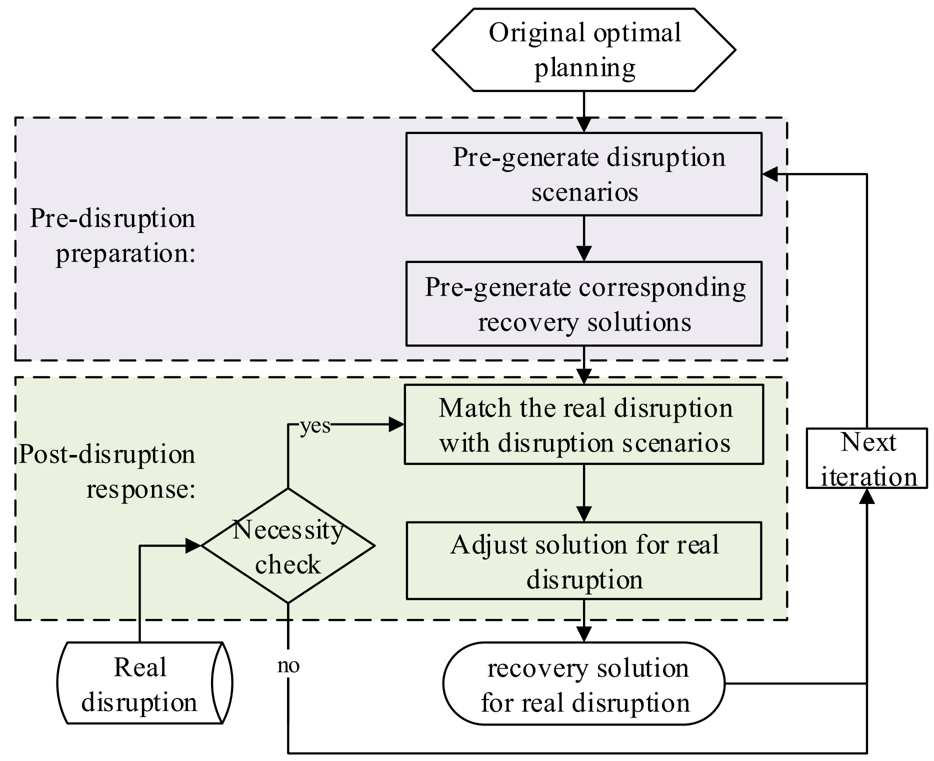
Original optimal planning
Pre-disruption preparation:
Pre-generate disruption scenarios
Pre-generate corresponding recovery solutions
Post-disruption response:
yes
Match the real disruption with disruption scenarios
Next iteration
Necessity check
Adjust solution for real disruption
Real disruption
no
recovery solution for real disruption

Fig. 2. Framework of the proposed scenario-based solution method.

4.1. *Checking the necessity of recovery efforts*

Intuitively, recovery efforts are not necessary if a disruption does not lead to significant violations of service time windows for subsequent services. Thus Voccia et al. (2019) defined the maximal disruption delay time as the longest delay that will not affect the original route's objective function defined in (2). The maximum delay is given to a vehicle before it leaves the depot of a given route. However, in online food delivery services, disruptions often occur after leaving the depot and during the travelling process. In addition, the time of disruption occurrence on a given route also affects the maximal disruption delay. Thus we need to determine if recovery effort is necessary immediately after a disruption.

An original route is defined as an ordered list of nodes: 0,1,…,N, $N+1$, where 0 and $N+1$ represent the depots; 1,…, N represent pickup or delivery nodes in the order of visiting sequences; $[a_i, b_i]$ represents the service time window in the node i; s_i represents the service time for node i, and $t_{i,i+1}$ represents the travelling time from node i to node $i+1$. Also, D_i denotes the vehicle departure time from the node i; A_i denotes the vehicle arrival time at the node i; W_i denotes the waiting time the vehicle spends at the node for order preparation; CW_{ij} denotes the cumulative value of the vehicle waiting time through the subtour $i,…,j$; S_i denotes the slack time of the vehicle at the node i without violating the service time b_i. These notations can be formulated as follows.

$$D_i = \max\{A_i, a_i\} + s_i, 1 \leqslant i \leqslant N \tag{17}$$

$$A_i = D_{i-1} + t_{i-1,i}, 1 \leqslant i \leqslant N \tag{18}$$

$$W_i = \max\{a_i - A_i, 0\}, 1 \leqslant i \leqslant N \tag{19}$$

$$CW_{ij} = \sum_{k=i}^{j} W_k, 1 \leqslant i \leqslant j \leqslant N \tag{20}$$

$$S_i = b_i - A_i, 1 \leqslant i \leqslant N \tag{21}$$

Proposition 1. *With an original route, when a disruption occurs before visiting node i (e.g., due to vehicle breakdown or traffic congestion when travelling from i − 1 to i, or service delay at node i − 1), the disruption time is denoted as DT_i. If DT_i meets the inequality defined in (22), then all requests in this route can be served without violating service time windows.*

$$DT_i \leqslant \min\{S_i, \min_{i<j<N}\{S_j + CW_{j-1}\}\} \tag{22}$$

The proof of Proposition 1 is shown in Appendix A. When one disruption occurs, we can first evaluate its effects using Eq. (22). Recovery efforts are only necessary when the equation does not hold. Such a check can avoid the waste of resources and improve the recovery efficiency when recoveries are indeed necessary. In addition, Eq. (22) can also help to design route selection heuristics and improve the efficiency of generating candidate recovery solutions in Section 4.2.3.

4.2. *The pre-disruption preparation*

After the route of the original PDPTW is identified but before any disruption actually occurs, we generate the possible disruption scenarios uniformly with a certain probability for disruptions of single or multiple types. The principle of generating possible disruptions is based on the multiple scenario approach proposed in Bent and Van Hentenryck (2004). The key idea is providing the possibilities and scales of disruptions on nodes, vehicles, and link edges between two adjacent nodes respectively. Once a disruption occurs on a node, all the routings related to the node after the disruption occurrence may be affected. After a disruption occurs on a vehicle, the corresponding vehicle routing may be affected. Similarly, for a disruption on a link edge, all the routings via the link after the disruption occurrence may be affected as well. In practice, an operator can choose the distributions for types and scales of disruptions based on real-world datasets.

For each generated disruption scenario, we generate the corresponding recovery solutions with the adaptive large neighborhood search (ALNS) heuristic. Note that these recovery solutions are only candidate solutions for an actual disruption. They will not be directly applied after a disruption. As an extension of the large neighborhood search heuristic, ALNS was also used in PDPTW (Ropke and Pisinger 2006) and its variants (Ghilas et al. 2016, Sun et al. 2020). To adopt ALNS for DR-PDPTW, we extend ALNS for PDPTW in three ways. First, multiple recovery solutions will be generated simultaneously for each disruption scenario by ALNS, so that we can create a diverse pool of candidate solutions. Second, a new pickup-oriented removal operator is added to ALNS to consider the phenomenon that a vehicle may pick up multiple orders at the same restaurant in reality, especially at peak hours. Third, a routing selection algorithm is designed in order to minimize the impact on other routes during the recovery process (Yu and Qi 2004) and obtain the recovery solution within seconds. We introduce the three extensions in detail below.

4.2.1. *The extended framework of ALNS heuristic*

Algorithm 1 provides the pseudocode of ALNS to generate candidate recovery solutions for a given disruption scenario. The initial feasible solution can be derived from the original routing schedules after successive delays. This is because disruption management aims at recovering the routings as soon as possible and minimizing the violations from the original schedules. The output is a best solution set S^*, in which there are multiple solutions for the given disruption scenario. This way aims at increasing the diversity of candidate recovery solutions.

The heuristic needs to select q requests to be removed from and then inserted into the routing. The value of q can be generated randomly as long as it is larger than 20% of the total number of requests that need to be served. These q requests are removed from the solution using a removal operator sr, which is selected from SR (the set of removal operators) using the roulette wheel principle with probability $w_{sr}/\sum_{i\in SR} w_i$. The q removed requests are then inserted into the solution for deliveries using an insert operator si, which is selected based on a similar probability $w_{si}/\sum_{i\in SI} w_i$ from SI (the set of insertion operators). The selection probability of operator i (sr or si) can be the weight of each operator, and is initially set to be the same. The weight is then updated based on the score of operator i during every μ iterations. The update equation is formulated as follows.

$$w_{i,j+1} = \begin{cases} w_{ij} & n_{ij} = 0 \\ (1-\beta)w_{ij} + \beta\theta_{ij}/n_{ij} & n_{ij} \neq 0 \end{cases} \tag{23}$$

Each segment includes μ iterations. $w_{i,j}$ represents the weight of operator i in the jth segment. θ_{ij} refers to the sum of total scores of operator i in the jth segment. It can be calculated by Step 4 (in Algorithm 1) according to the performance of the new solution S' compared to the best one from set S^* and the current one S. In other words, if the new solution S' is better than any solution in set S^*, then S' will be added into S^*, the worst one solution s^W will be deleted from S^* and the score of the operators sr and si will be updated by adding $\Delta\theta_1$; if the new solution S' is only better than S, then the current solution will be replaced by S', and the score of the operators sr and si will be updated by adding $\Delta\theta_2$; otherwise, the new solution will be accepted with probability $\alpha = e\hat{}[(F(S)\text{-}F(S'))/T]$, and the score of the operators sr and si will be updated by adding $\Delta\theta_3$, where $\Delta\theta_1 > \Delta\theta_2 > \Delta\theta_3$. The weight then is reset to zero at the beginning of the next segment. $n_{i,j}$ represents the number of times operator i is selected in the jth segment of iterations. β is the update speed of the weight from the jth segment to the next one.

Step 6 and Step 7 update temperature T iteratively, as in simulated annealing. This not only avoids local optimum in the search process, but also accelerates the convergence. If T is below ϵ, then it is reheated to its original temperature. Finally, Step 8 decides whether the search should stop.

Algorithm 1. Pseudocode for the *ALNS* heuristic

```
Step 0.  Input: initial feasible solution $S$; initial temperature $T_0$; reduction rate $\gamma$; set
         of removal operators $SR$ with a score $\theta=0$ for each element; set of insert
```

(continued on next column)

(continued)

```
         operators SI with a score $\theta=0$ for each element; Iteration limitation
         $max\_index$
Step 1.  Let $S^*=\{S\}$, $T=T_0$, $index=0$
Step 2.  Let $S'=S$, $q$=a random number larger than $0.2|R|$, $sr$=an operator from $SR$
         by roulette wheel selection principle with the corresponding weight $w$,
         $si$=an operator from $SI$ by roulette wheel selection principle with the
         corresponding weight $w$
Step 3.  Remove $q$ requests from $S'$ by $sr$ and insert the q requests into $S'$ by $si$
Step 4.  If $F(S')<F(s^*)\ \forall s^*\in S^*$, then
           $S^*= S^*\cup\{S'\}/\{s^W\}$, $S=S'$, and $\theta_{sr}=\theta_{sr}+\triangle\theta_1$, $\theta_{si}=\theta_{si}+\triangle\theta_1$
         Else if $F(S')<F(S)$, then
           $S=S'$, and $\theta_{sr}=\theta_{sr}+\triangle\theta_2$, $\theta_{si}=\theta_{si}+\triangle\theta_2$
         Else then
           Let $S=S'$ with the probability $\alpha$, and $\theta_{sr}=\theta_{sr}+\triangle\theta_3$, $\theta_{si}=\theta_{si}+\triangle\theta_3$
Step 5.  if $index\%\mu = 0$ then
           Update weights $w$ and reset $\theta_i=0$ for each operator $i$
Step 6.  $T=\gamma T$, $index=index+1$
Step 7.  If $T<\epsilon$ then $T=T_0$
Step 8.  If $index< max\_index$ then
           To Step 2
         Else then
           To Step 9
Step 9.  Stop the algorithm and output $S^*$
```

4.2.2. *Removal and insertion operators in ALNS*

The removal operators set SR consists of four operators: *random removal operator*, *worst removal operator*, *Shaw removal operator*, and *pickup-oriented operator*. A *random removal operator* randomly removes q requests from the current solution and is the fastest removal operator. A *worst removal operator* removes requests that will lead to the largest cost saving (according to the objective function in Equation (2) when removed from the solution. The goal is to insert removed requests to other routes with lower cost.

A *Shaw removal operator* is (Shaw 1997, Shaw 1998) removes q similar requests, where $sim(r,r')$, the similarity between requests r and r', includes three terms: distance D between pickup nodes of the two requests (p_r and $p_{r'}$) and between delivery nodes of the two requests (d_r, $d_{r'}$), the delay time violation between the two requests (z_r-$z_{r'}$), and service beginning time violations between pickup nodes ($T(p_r)$-$T(p_{r'})$) and between delivery nodes ($T(d_r)$-$T(d_{r'})$). They are weighted by φ, η, and σ, respectively in Equation (35).

$$\begin{aligned} sim(r,r') = {} & \varphi(D(p_r,p_{r'}) + D(d_r,d_{r'})) + \eta|z_r - z_{r'}| \\ & +\sigma(|T(p_r) - T(p_{r'})| + |T(d_r) - T(d_{r'})|) \end{aligned} \tag{24}$$

A *pickup-oriented removal operator* (Algorithm B.1 in Appendix B) will remove requests from the same pickup node. The intuition is that serving requests from the same pickup node with the same vehicle leads to lower traveling cost. As described in Algorithm B.1, the model first randomly selects and removes one request. It then removes other related requests that share the same pickup nodes as the already removed one. If the number of related requests is lower than q, then another request with a different pickup node and its related requests are selected. This process repeats until it is up to q.

After removing requests from the current routing plan, we use insertion operators to assign these requests to another vehicle's route and obtain a feasible solution. This procedure is listed in Algorithm B.2 of Appendix B. Given a request r to be inserted and a vehicle route k, we will first try to insert the pickup node into route k at a position with the minimum cost. Once that is done, we will attempt to insert the corresponding delivery node following the same logic. If either insertion is not possible, we will stop and test the next request or route. Finally, the minimum-cost position of the pickup node and the delivery node will be output. Moreover, the change of objective value ΔF_{rk} can also be calculated after inserting request r into vehicle route k. Note that the way to pick the minimum-cost position depends on the insertion operator. i.

The set insertion operators *SI* consists of *basic greedy insertion* and *regret_m insertion operators* (with various values for *m*) (Ropke and Pisinger 2006). A *basic greedy insertion operator* will determine the minimum-cost position according to the following objective.

$$\Delta F^* = \min_{r \in R^*} \min_{k \in K} \{\Delta F_{rk}\} \tag{25}$$

With request *r*, a *regret_m insertion operator* will sort the route *k* to ensure $\Delta F_{rk} \leq \Delta F_{rk'}$ for $k < k'$, k, $k' \in \{1,2,\ldots\}$. If $\Delta F_{rk} = \infty$ for some requests $k < m$, then we will select the request that can be inserted into the route with the fewest number of nodes.

$$\Delta F^* = \max_{r \in R^*} \{\sum_{k=1}^{m} (\Delta F_{rk} - \Delta F_{r1})\} \tag{26}$$

Following Ropke and Pisinger (2006), we add noise to the change of insertion cost as $\Delta F^* = \max\{0, \Delta F^* + noise\}$ to avoid local maxima. The noise can be randomly selected from the range $[-\varepsilon, \varepsilon]$, where ε is closely related with the maximal distance between any two nodes. Following the framework proposed by Ropke (2009), we make use of parallel computing to speed up the ALNS, i.e. the multi-core structure in modern CPUs is used to implement removal and insertion operators on different cores in parallel.

4.2.3. *The routing selection heuristic*

According to empirical rules commonly adopted in practice, only a subset of vehicles are used to handle disruptions, even in the case of large disruptions. This is because for a minor disruption, it is not necessary to involve many vehicles. Also, as each driver has his/her own delivery task, large-scale exchange or transfer of requests may lead to logistic challenges in routing schedules. Thus we propose a routing selection heuristic (Algorithm B.3 in Appendix B), in which different selections of vehicles are iteratively considered. This procedure ensures that the recovery solutions can be obtained within seconds.

The heuristic is based on Definition 1 below, which is derived from Proposition 1. We assume that a route with larger slack time *ST* will have more time to absorb possible disruptions. Because one pickup node may serve multiple requests in online food delivery, we first find all disrupted pickup nodes based on originally planned routes. All routes passing through such nodes within the disruption recovery periods will be defined as candidate recovery vehicles. We then order candidate recovery vehicles in the descending order of the *ST* of their routes, so that vehicles with higher slack time are more likely to be used for disruption recovery. The remaining routes are added into the candidate vehicle route set C_{use} used for recovery. Through this procedure, all vehicles will be considered once, but those considered earlier are more likely to contribute to a good recovery solution. Different sets of $M^*|K_{disrupt}|$ vehicle routings in C_{use} are considered consequently, together with the disrupted vehicle route set $K_{disrupt}$, as the input of ALNS. We will update the previous recovery solution if a better solution is obtained by the ALNS heuristic iteratively. Finally, then heuristic will stop when all subsets of C_{use} are checked.

Definition 1. *Given an original route, the slack time of the route (denoted by ST) can be defined as follows.*

$$ST = \max_{1 \leqslant i \leqslant N} MDT_i \tag{27}$$

$$MDT_i = \min\{S_i, \min_{i<j<N} \{S_j + CW_{j-1}\}\} \tag{28}$$

4.3. *Post-disruption response*

When a disruption occurs, the post-disruption response process will start by (1) matching generated scenarios with the actual disruption and (2) adjusting a candidate recovery solution based on the real one.

4.3.1. *Scenario matching*

We assume that two disruptions have a higher level of match if their locations, types and severity are more similar. Given the large number of generated disruption scenarios, it will take a long time to match each scenario with the actual disruption in a sequential order. To accelerate the process, we design a clustering-based matching method, which groups generated disruption scenarios first, and then matches the real disruption scenario with scenarios from one of the clusters. Specifically, we adopt the classic K-means clustering algorithm. Note that disruption locations, types, and severity are measured in different ways, so we need to conduct dimensionless processing before clustering.

In addition, we also extended the unsupervised clustering-based method to design a two-stage clustering-based method, which incorporates more background knowledge on requests and service nodes. The first stage clusters nodes involved in the original routing schedules, while the second stage clusters generated disruption scenarios based on clustering results of the first stage. Both stages use the COP-Kmeans algorithm (Wagstaff et al. 2001). Specifically, the first stage clusters nodes based on both internode distance and must-link constraints. Such constraints are based on requests: if the pickup nodes (or delivery nodes) of two requests are the same, there will be must-link constraints among all pickup and delivery nodes of the two requests. All must-link constraints based on the transitive closure property of the must-link constraints (Bishop, 2006) and the closure of all points are needed before we run the first-stage clustering outlined in Algorithm 2.

Algorithm 2. The first stage COP-Kmeans algorithm for node clustering

```
Step 0.  Input: the set of pickup nodes and delivery nodes $P^0 \cup D^0$; the number of clusters $k$; must-link constraints set $ML$
Step 1.  Let $C_1, \ldots, C_k$ be the initial cluster set
Step 2.  $s$=a node randomly selected from ($P^0 \cup D^0$) without replacement, evaluate the distance ($d_{s1}, \ldots, d_{sk}$) from $s$ to the centers of $C_1, \ldots, C_k$ respectively and sort them in ascending order (let $d_{s1} \leq \ldots \leq d_{sk}$), $i$=1
Step 3.  While $i \leq k$ do
           If $\exists(s,s') \in ML$ meets $s' \in C_i$ then
             break
           else $i \leftarrow i+1$ end if
         End while
Step 4.  If $i <= k$ then set $C_i = C_i \cup \{s\}$, else set $C_1 = C_1 \cup \{s\}$
Step 5.  Update the centers of $C_1, \ldots, C_k$, If the centers are not changed, then to Step 6, otherwise to Step 2
Step 6.  Stop the algorithm and output $C_1, \ldots, C_k$
```

During the second stage clustering that focuses on generated disruption scenario (Algorithm 3), we also add the cannot-link constraints among scenarios to aim clustering. If nodes related to two disruption scenarios are not in the same cluster derived from the first stage clustering, there will be a cannot-link constraint between the two disruption scenarios. With node clusters obtained from the first stage, we tease out the cannot-link constraint set *CL* among generated disruption scenarios before using the COP-Kmeans algorithm to cluster generated disruption scenarios.

Algorithm 3. The second stage COP-Kmeans algorithm for disruption scenario clustering.

```
Step 0.  Input: generated disruption scenarios set $DS$; the number of clusters $k$; cannot-link constraints set $CL$
Step 1.  Let $DS_1, \ldots, DS_k$ be the initial cluster set
```

(*continued on next page*)

(*continued*)

```
Step 2.  $s$=an element randomly selected from $DS$ without replacement, evaluate
         the distance ($d_{s1}$, …, $d_{sk}$) from $s$ to the centers of $DS_1$, …, $DS_k$ respectively
         and sort them in ascending order (let $d_{s1} \leq … \leq d_{sk}$), $i$=1
Step 3.  While $i \leq k$ do
           If $\exists(s,s')\in CL$ meet $s'\in DS_i$ then
             $i \leftarrow i+1$
           else break   end ifend while
Step 4.  If $i \leq k$ then set $DS_i = DS_i \cup \{s\}$, else stop the algorithm and output clustering
         failure
Step 5.  Update the center of $DS_1$, …, $DS_k$, If the center is not changed, then to Step
         6, otherwise to Step 2
Step 6.  Stop the algorithm and output $DS_1$, …, $DS_k$
```

4.3.2. *Adjusting and improvement strategies for the matching scenario*

Although candidate recovery solutions for the matching scenario are as close as we get from prepared solutions before disruptions, they may be far from being optimal or even feasible for the actual disruption. Therefore, we will adjust and improve such candidate recovery solutions of the matching scenario, and then select the best one as the final solution to the actual disruption.

The key idea of such adjustment is that a candidate recovery solution serves as an input into the recovery process to warm-start the new solution, and all requests that have been served before the disruption will be deleted from the new solution. Vehicle routing may be the same according to the candidate solution, but the arrival time and service time may be totally different, due to differences between the actual disruption and historical disruption scenarios. Additionally, newly arrived requests should be added into the set of unfinished requests as well. Therefore, the related traveling cost and the deviation cost from original schedules will be adjusted for the actual disruption.

The improvement strategy is mainly based on a local search heuristic with two steps that are executed iteratively in the following order.

(1) Reposition of all disrupted requests in the descending order of disruption severity. We first try to remove the most serious request from the original route and evaluate the outcome if it is inserted into all other routes. The reposition with the greatest decrease in recovery cost among all possible insertions will be accepted for the disrupted request. The process will be repeated for all disrupted requests until no improvement of objectives can be obtained.
(2) Cross insertion between one disrupted request and another request from different routes. Similar to the first strategy, we first sort disrupted requests in the descending order of disruption severity and select the most serious request. Then we pick another randomly selected request from other routes and evaluate the cost if we insert each request into the other's route. If the cost is lower than the current solution from the first step, then we will adopt it. The iterations are repeated until no improvement can be obtained.

5. Computational experiments

This section describes our computational experiments in detail, including the generation of instances, parameter tuning, results and a case study. All computations are performed on an workstation with Intel i5-7200u CPU and 16 GB of RAM (Windows OS). Our proposed method and the benchmark method are both coded in C++.

5.1. *The generation of instances*

The instances are obtained for the city of Barcelona according to Sartori and Buriol (2020), where pickup and delivery nodes belong to different node sets respectively. However, to reflect characteristics of online food deliveries, we allow some requests to start from the same pickup location (i.e., the same restaurant) in generated instances for our study. Thus, based on the first 100-node instance (i.e. 50 requests), the first 200-node instance (i.e. 100 requests), the first 600-node instance (only used the first 250 requests) and the first 1000-node instance (i.e. 500 requests) in Sartori and Buriol (2020), we modify the selected instances by combining the pickup locations of some requests. The new instances that we generated can be downloaded from https://github.com/team-huyuzhen/instances/tree/master/dr-pdptw-instances.

Our instances contain 50, 100, 250, 500 requests. Each pickup node corresponds to no more than 5 requests. Disruption types in our experiments include vehicle breakdown (denoted by V), disruption of the link edge between two adjacent nodes (denoted by E), and the disruption of service time in pickup node or delivery node (denoted by N). Disruption severity depends on disruption types. For vehicle breakdowns, the length of disruption (i.e., the time it takes to repair a vehicle) can be 10 (low) or 60 min (high). The severity of link disruptions is represented by how much the planned travelling time between the two adjacent nodes has increased. For example, a severity of 2 means the travelling time has doubled between two nodes due to link disruption. Severity for node disruptions can be 5 (low), or 30 min (high). We combine the disruption type, severity and the total number of requests in our notation for a disruption instance. For example, instance N5_50 has 50 requests and service time disruption with a 5-min delay on some nodes. In the following experiments, we use 24 instances, 20 with single-type disruptions and 4 with multiple types of disruptions.

It is worth noting that the traveling time matrix between any two nodes in DR-PDPTW may violate the triangle inequality property, due to vehicle breakdowns or link edge disruptions. However, whether a node i is traversed strictly corresponds to the task of pickup or delivery for a request $\mathbf{r}$. In other words, it is not allowed to visit a node when there is no request related to the node. Moreover, for the pickup or delivery node of request r, the service time window $[a_{ri}, b_{ri}]$ of node i is still strictly enforced. Thus if a vehicle arrives at the node i earlier than a_{ri}, it has to wait until a_{ri} to start the service (e.g., a vehicle has to wait for the food to get ready for pickup). This ensures that DR-PDPTW still has a reasonable model.

5.2. *The pre-generation simulation of disruption scenarios*

As we discussed above, basic instances are derived from the literature and it is difficult to obtain detailed historical disruption information. Therefore, we pre-generate disruption scenarios by using simulations. Based on the original planning, our simulations will generate a series of disruptions in advance. How advanced such pre-generation should take place will be analyzed in Section 5.4.3.

We will randomly generate disruptions for 8% of nodes and vehicles. In terms of edge disruptions, as connections between request nodes form a fully connected network, there will be much more edges than request nodes. The rate of edge disruptions is set to 5% of the total edges, which is lower than other disruption types. All disruption types are randomly generated in each disruption scenario. Disruption severity is consistent with the instances in the mass and randomly distributed in the disruption scenarios. For example, node disruptions with 5 min and 30 min occur randomly with the same possibilities in one scenario.

5.3. *Model tuning for extended ALNS*

To tune parameters in the extended ALNS heuristic for the pre-disruption phase, we set initial parameter values following an early study by Ropke and Pisinger (2006). Then we vary the value of one parameter while keeping others unchanged. Each setting of parameters setting is evaluated with the ALNS heuristic on a group of instances and its performance is measured by average gap between the setting and the best-performing setting.

For the number of iterations, we compare the performance and running time of the instance with 50 and 500 requests under 25,000, 50,000, 200,000, and 400,000 iterations in Table 1. Among them, the solution derived from ALNS with 400,000 iterations has the best-known performance. The "*Gap(%)*" column is the relative difference of objective function values between a solution and the best-known solution, while the "*CT*(sec)" column lists running time in seconds. From Table 1, we find that 100,000 iterations provide a good balance between performance and running time—on average, it has 8% decrease in performance but uses only 1/5 of the running time when compared to the best-known solution.

As for the number of requests removed from one solution at each iteration, Table 2 suggests that the configuration with the value of min (50,0.2|*R*|) leads to the best performance.

We also test the 24 instances to obtain the best parameters of the ALNS heuristic for the disruption recovery problem. The DR-PDPTW problem is different from scheduling optimization, especially in the weights for evaluating similarity of two requests in the Shaw remove operator. Thus we test various values for the three weights φ, η, and σ in formula (24). Results in Table 3 suggest that the gaps between objective values obtained by different weights are very small. When the three weights are 6, 5 (6 or 7), and 9, respectively, the average performance of the solution is the best.

To assess whether to use all operators in the ALNS heuristic, we run the heuristic with all but one operator at a time and report results in Table 4. Comparing rows 2 to 5 with the first row, we see that each removal operator can help to reduce the gap. Thus, we choose to include all removal operators in the proposed ALNS heuristic. In addition, the comparison between the first and third rows clearly demonstrates the contribution of the "pickup-oriented removal operator". Rows 6 to 10 show a similar pattern—all insertion operators, especially *regret_4*, contribute to the decrease in gap, except the regret_*m* insertion operator, which is consequently excluded from the proposed ALNS heuristic.

5.4. Computational results for DR-PDPTW

This section aims to verify the performance of the proposed algorithms for DR-PDPTW. We first demonstrate that ALNS is better than the MIP Solver at obtaining candidate recovery solutions. Then we compare the performance of our scenario-based approach with the direct adjusting method when dealing with real time disruptions. To investigate the effects of different components insides our approach, we then analyze the performance with various matching strategies during the post-disruption response stage and with various numbers of candidate recovery solutions. We also show the necessity of routing selection algorithms during the pre-disruption preparation stage. Last, we apply the proposed approach to real-world data to further demonstrate its efficacy.

5.4.1. Performance of ALNS for generation of candidate solution

In this section, we verify the performance of ALNS in generating candidate solutions for DR-PDPTW. Table 5 shows the computational results solved by the ALNS heuristic and the Gurobi Solver respectively. The stop criteria are 3600 s of running time for the Gurobi Solver and 100,000 iterations for the ALNS heuristic. We set the solution derived from ALNS with 400,000 iterations as the best-known solution. "*AvgGap*" reports the average difference in the objective function values between them and ALNS heuristic with 400,000 iterations. Similarly, "*BestGap*" and "*WorstGap*" show the best and the worst difference between them. "*AvgCT*", "*minCT*" and "*maxCT*" represent the average, minimum and maximum running time of solution method respectively. As seen from Table 5, ALNS heuristic can obtain the optimal solutions within less than 10 s for most instances, and the gap is almost zero. However, with the running time limitation of one hour, Gurobi can only obtain feasible solutions for more than 15 requests, and the worst gap is even higher than 110%. Also note that the scale of instances in Table 5 is much smaller than what a real-world online food delivery service would face. This further highlights that the ALNS heuristic, instead of the Gurobi Solver, is more suitable for generating candidate solutions for DR-PDPTW, especially for the disruption requiring quick recovery.

Table 1
Computational results of ALNS with different numbers of iterations.

#iteration	50		500		average	
	Gap(%)	CT(sec)	Gap(%)	CT(sec)	Gap(%)	CT(sec)
25,000	20.7	12.9	13.5	427.5	17.1	220.2
50,000	13.9	24.5	11.6	851.2	12.8	437.8
100,000	8.9	50.3	7.3	1432.7	8.1	741.5
200,000	5.6	90.1	4.0	3638.7	4.8	1864.4
400,000	0.0	194.1	0.0	6937.4	0.0	3565.8

Table 2
The average gap for various number of requests removed.

q	Gap (%)
min(50,0.2|R|)	0.75
min(75,0.3|R|)	0.95
min(100,0.4|R|)	1.03
min(125,0.5|R|)	0.94
min(150,0.6|R|)	1.27
min(175,0.7|R|)	1.46

Table 3
Parameters in Shaw removal operators for disruption recovery.

φ		η		σ	
Values	Gap (%)	Values	Gap (%)	Values	Gap (%)
1	0.80	1	1.28	1	0.66
2	0.51	2	1.28	2	1.29
3	0.61	3	1.28	3	1.16
4	0.59	4	1.28	4	0.88
5	0.66	5	0.00	5	1.18
6	0.41	6	0.00	6	0.86
7	0.66	7	0.00	7	1.07
8	0.43	8	0.19	8	0.70
9	0.60	9	0.32	9	0.24
10	0.65	10	0.32	10	0.56

5.4.2. Performance of the scenario-based approach

The average results of the 24 instances solved with the proposed approach under various disruptions and various instance size are summarized in Table 6 and Table 7 respectively, with more detailed results in Appendix C. In the two tables, "*AvgObj*" gives the average objective values of the optimal recovery solutions and "*AvgCT*" reports the average response time in seconds, (i.e., the average time required to obtain the recovery solution after a real disruption occurs). "*Direct adjusting*" refers to the disruption recovery result obtained by using directly ALNS as a solution method when the disruption occurs. "*Scenario-based approach*" refers to the solution obtained by the proposed scenario-based approach. "*AvgGap*" reports the average gap of the solutions by the different approaches.

Table 6 shows no matter what the disruption is, the average gap between the scenario-based approach and the directly adjusting method are all within −1.5% and 0. Meanwhile, the average response time of the direct adjusting method is almost 5 times of the scenario-based solution approach. The sharp distinction in the average response time highlights the advantages of the proposed scenario-based solution approach in providing real-time responses with almost no compromise in solution quality, even under various types of disruptions, which makes it a good fit for online food delivery services. Note that the average gap of all disruption scenarios is less than 0. It shows that the scenario-based approach has better performance than the direct adjusting method, mainly due to the improvement strategies adopted by the scenario-based

Table 4
Average gaps for various configurations of removal and insertion operators (an empty cell means the corresponding operator is missing from a configuration. Removal operators are in italic and insertion operators are in bold).

Conf.	*Rand*	*Pick*	*Worst*	*Shaw*	**Greedy**	**Reg_2**	**Reg_3**	**Reg_4**	**Reg_m**	gap (%)
1	●	●	●	●	●	●	●	●	●	2.5
2	□	●	●	●	●	●	●	●	●	3.7
3	●	□	●	●	●	●	●	●	●	2.9
4	●	●	□	●	●	●	●	●	●	3.7
5	●	●	●	□	●	●	●	●	●	3.3
6	●	●	●	●	□	●	●	●	●	2.6
7	●	●	●	●	●	□	●	●	●	3.7
8	●	●	●	●	●	●	□	●	●	2.7
9	●	●	●	●	●	●	●	□	●	4.3
10	●	●	●	●	●	●	●	●	□	2.1

Table 5
Computational results from Gurobi Solver and the ALNS heuristic (*Gaps* are % and *CTs* are in seconds).

#request	*Gurobi*						*ALNS*					
	Avg Gap	*Best Gap*	*Worst Gap*	*Avg CT*	*Min CT*	*Max CT*	*Avg Gap*	*Best Gap*	*Worst Gap*	*Avg CT*	*Min CT*	*Max CT*
10	0.0	0.0	0.0	1844	1663	2055	0.2	0.0	0.8	3.3	2.5	4.3
15	0.0	0.0	0.2	3600	3600	3600	0.0	0.0	0.0	4.5	4.0	5.2
20	7.7	4.4	17.6	3600	3600	3600	0.0	0.0	0.0	5.6	5.1	6.9
25	77.8	52.7	112.4	3600	3600	3600	0.0	0.0	0.0	7.0	6.4	7.6

Table 6
Performance of the scenario-based approach under various disruptions.

Disruption scenario	Direct adjusting		Scenario-based approach		
	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgGap (%)	AvgCT (sec)
E2	1,476,681	188	1,462,939	−0.9	37
N5	1,442,355	192	1,437,832	−0.3	32
N30	1,477,013	187	1,469,933	−0.4	25
V10	1,475,637	190	1,454,013	−1.4	29
V60	1,486,084	192	1,465,456	−1.3	40
N5V10E2	1,471,718	191	1,455,156	−1.1	31

Table 7
Performance of the scenario-based approach under various instance sizes.

#request	Direct adjusting		Scenario-based approach		
	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgGap (%)	AvgCT (sec)
50	212,981	13	213,395	0.1	0.14
100	737,823	20	747,462	1.3	0.39
250	1,403,243	178	1,391,329	−0.8	11.97
500	3,532,279	550	3,478,034	−1.5	117.58

approach.

Table 7 shows the performance of the proposed solution approach with various instance sizes denoted as "*#request*". Similar to Table 6, we find little difference in objective values of various solution approaches, but their average response time varies greatly. Note that the average gap is positive with small instance size (50 requests and 100 requests) and negative with larger instance size (250 requests and 500 requests). The reason may be that for smaller instance sizes, the direct adjusting method based on ALNS has almost found global optimal solution. However, the global optimization of directly adjusting method failed to solve the larger instance size within such limited time. This also offers a chance for improvement strategies to continue improving the recovery solutions.

5.4.3. Performance with various matching strategies

The average results of the 24 instances solved with the proposed scenario-based approach with various matching strategies are summarized in Table 8. "*disT*" is the pre-disruption time for the generations of disruption scenarios and corresponding candidate recovery solutions in our simulations. Larger values mean more time to generate more disruption scenarios and candidate recovery solutions. "*match*" means the proposed solution approach with the sequential matching method; "*match_C*" and "*match_2C*" refer to the proposed solution approach with clustering-based and two-stage clustering-based methods, respectively.

From Table 8, we see that all the three matching strategies performs well in the post-response stage. It is also noted that the average solution value with lower pre-disruption generation time is larger than that with larger pre-disruption generation time, because the proposed solution approach needs time before disruptions to generate enough disruption scenarios and corresponding solutions. In particular, the solution value derived from "*match_2C*" is the lowest and the most stable. Moreover, the average response time of "*match_2C*" is the lowest as well.

5.4.4. Performance with various numbers of candidate recovery solutions

After a disruption, once a disruption scenario is matched to the actual disruption, all candidate recovery solutions will be adjusted to fit the actual disruption. Table 9 shows how the performance of the proposed model changes with different numbers of candidate solutions (*#cand_sol*). We can see that with the increasing of "*#cand_sol*" from 1 to 5, objective values of most instances decrease and the response time increases. This reflects a tradeoff\mathord{-} higher diversity of candidate solutions can improve the performance of recovery, but it needs more response time to adjust candidate solutions. Additionally, disruption types have little impact on objective values and response time of the recovery solution, but the size of instances has significant impact on them. Thus we analyze the routing selection heuristic in Section 5.4.5.

5.4.5. Performance of the routing selection heuristic

To demonstrate the contribution of the routing selection heuristic in ALNS, we test four instances, and report results in Table 10, where "*AvgGT*" represents the average running time of generating recovery solutions for each generated disruption scenario. Both the objective value and response time of the proposed solution approach are reduced by using the routing selection heuristic, mainly because candidate recovery solutions are generated by swapping some of the vehicles instead of all vehicles. This leads to faster generation of candidates (See

Table 8
Performance with various matching strategies.

disT (min)	match		match_C		match_2C	
	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgCT (sec)
5 min	1,526,819	40.2	1,528,057	40.2	1,530,257	40
15 min	1,479,922	38.2	1,479,899	37.8	1,478,784	37
30 min	1,365,658	21.2	1,363,709	20.9	1,363,624	20.5

Table 9
Performance of "*match_2C*" with various numbers of candidate recovery solutions.

#cand_sol	V60_100		N5V10E2_100		V60_1000		N5V10E2_1000	
	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgCT (sec)	AvgObj (CNY)	AvgCT (sec)
1	246,103	0.04	227,230	0.05	3,539,723	54	3,559,693	58
3	232,597	0.12	214,850	0.13	3,499,457	170	3,502,318	145
5	222,391	0.20	219,088	0.21	3,513,746	206	3,487,148	322

Table 10
Performance with and without the routing selection heuristic.

Instances	Without routing selection			With routing selection			Gap (%)		
	AvgObj (CNY)	AvgCT (sec)	AvgGT (sec)	AvgObj (CNY)	AvgCT (sec)	AvgGT (sec)	AvgObj	AvgCT	AvgGT
V60_100	212,275	0.2	12.2	207,364	0.2	5.5	−2	0	−55
N5V10E2_100	223,007	0.2	12.2	216,893	0.1	12	−3	−50	−2
V60_100	3,552,865	138.2	453.4	3,545,929	111.9	181.2	0	−19	−60
N5V10E2_100	3,615,404	139.9	453.2	3,571,714	119.5	453.1	−1	−15	0

"*AvgGT*"), which means more disruption scenarios with higher diversity can be generated within a limited amount of time before disruptions. Although the quality of candidate solutions may not be better in the generation process, they can be adjusted after a disruption.

Also, we can see that the gap of "*AvgGT*" is smaller for the instances with higher disruption severity such as "N5V10E2". The reason is that when a disruption is very severe, the number of disrupted vehicles is multiplied by a constant and may become larger than the total number of vehicles in the original planning. Then the routing selection heuristic will stop working.

5.4.6. *Experiments with real-world restaurant locations*

To further demonstrate the efficacy of the proposed approach in practice, we collect 118 restaurant locations from the API of Baidu Maps (a popular online map service in China). All these restaurants are served by one delivery station (denoted as a black star in Fig. 3) from an online food delivery platform. Because we cannot obtain detailed data of requests and customers from Baidu Maps or the platform, we have to simulate customer locations, service time windows and disruption scenarios. Although part of the data used in our experiments is synthesized, these experiments can still help us verify the effectiveness of the

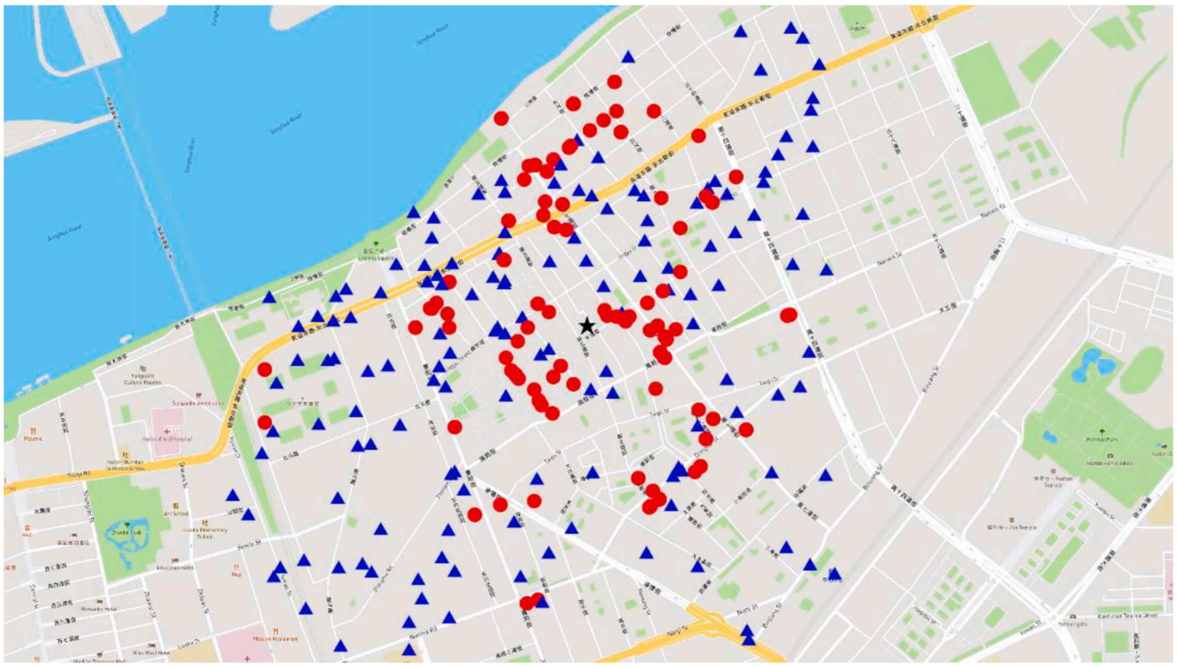

Fig. 3. The case study with 150 customers for a distribution station in China.

scenario-based solution approach on a scale that is close to real-world operations.

We randomly generate 150, 200, and 250 customers respectively as delivery nodes (denoted as blue triangles) within a circle. The center of the circle is the geographic center of all restaurants and the radius of the circle is 1.2 times the distance of the farthest restaurant from the circle's center point. For any request, the pickup and delivery node are randomly matched. Service time windows of pickup nodes are randomly chosen within 3 h from noon. The service at a delivery node begins as the starting time of its pickup window plus the travelling time from the pickup node to the delivery node. The length of the service time window at the delivery node is a random number between 30 and 60 min. We follow the convention of time being proportional to distance like Li and Lim (2001). But the distance between two nodes is calculated based on Haversine distance in this paper, instead of Euclidean distance in Li and Lim (2001).

In addition, we cannot find data for traffic jams from Baidu Maps. Moreover, the distance between two nodes was calculated based on their latitude and longitude, instead of considering the actual road network between the two nodes. Thus we have to simulate disruption scenarios. We assume that 17 randomly chosen edges between adjacent nodes had travelling time doubled due to edge disruption from traffic jams. The method to generate disruptions is the same as the one described in Section 5.1. The traveling time matrix may violate the triangle inequality property, but it does not affect our analysis. Data of these generated instances can be downloaded from the aforementioned online data archive as well.

Computational results are shown in Fig. 4, where Fig. 4(a) displays the objective values and Fig. 4(b) displays the running time. In the experiment, we give nine instances. The number of delivery nodes are 150, 200, 250 and the traveling time of the 17 link edges is double, triple, and quadruple (i.e. 2, 3, and 4) respectively as the original planning due to traffic jam. The figures demonstrate better performance of the proposed scenario-based approach than the direct adjusting approach based on ALNS. Take the instance (150, 2) as an example, our proposed approach can provide a recovery solution within less than 1 s. By contrast, the direct adjusting approach obtains a similar solution, whose performance is within 0.7% of the proposed solution, but takes 30 s. Moreover, the experiments reflect that the number of delivery nodes has great influence on the recovery performance and running time, but the influence of disruption level is not obvious.

5.5. *Analysis of divergence from original plans*

Previous research on PDPTW mainly focused on minimizing the total travelling cost, while DR-PDPTW also considers the divergence from original routes. Such divergence can be measured in two ways: whether the request is fulfilled by the original vehicle (the second term in the objective function) and the service delay of a request against the original time window (the third term in the objective function). Therefore we show the relative analysis in Section 5.5.1 and Section 5.5.2 respectively. Moreover, we analyze the impact of two disruptions: node service time change and vehicle breakdown, on the proposed recovery solutions in Section 5.5.3 as well.

5.5.1. *Divergence from the original assignment of requests to vehicles*

Fig. 5 shows the recovery results of DR-PDPTW compared with the original plan. Fig. 5(a) displays the number of requests served by the different vehicles against the original assignment, and Fig. 5(b) shows the delay cost in the recovery solution compared with the original plan. Moreover, as a comparison, we also include a benchmark model that skips the second term in the objective function (i.e., it does not consider the deviation cost of reassigning disrupted requests to other vehicles). In Fig. 5(a), the vertical axis on the left hand side indicates the number of requests violating original assignments; the vertical axis on the right indicates the gap of the number of the violating requests between our recovery solution and the benchmark. Compared to our model, the benchmark model has much more requests that violate original vehicle assignments\mathord{-} the average increase is 50% and the maximum increase is 80%. In fact, trying to keep original vehicle assignments also leads to lower total delay cost, as shown in Fig. 5(b). This is likely because changing vehicles for a disrupted request will inevitably lead to delay of the delivery. Both results highlight the value of minimizing the divergence in delivery vehicle assignments.

5.5.2. *Divergence from time window based on customers' tolerance of delays*

In this section we attempt to analyze if customers' tolerance of delay (δ) would affect the delay of recovery solutions. In Fig. 6, the horizontal axis is $1/\delta$, whose value ∞ (i.e., $\delta = 0$) refers to hard time windows with no tolerance in DR-PDPTW. The vertical axes represent the total delay time of requests divergent from original time window in the optimal recovery solution. Fig. 6(a) is based on two instances with size 100 and Fig. 6(b) corresponds to two instances with size 1,000. In Fig. 6(a), as $1/\delta$ increases (i.e., lower customer tolerance to time violation), the total delay of all requests decreases significantly until it becomes 0. Fig. 6(b) shows similar results, although no solution is feasible when customers have zero tolerance. Such results suggest that surveys on customers' tolerance of time violations should be developed, so that recovery solutions can be adjusted accordingly. By properly setting time windows based on different customers' tolerance levels, a platform can reduce service delay of drivers when disruptions occur.

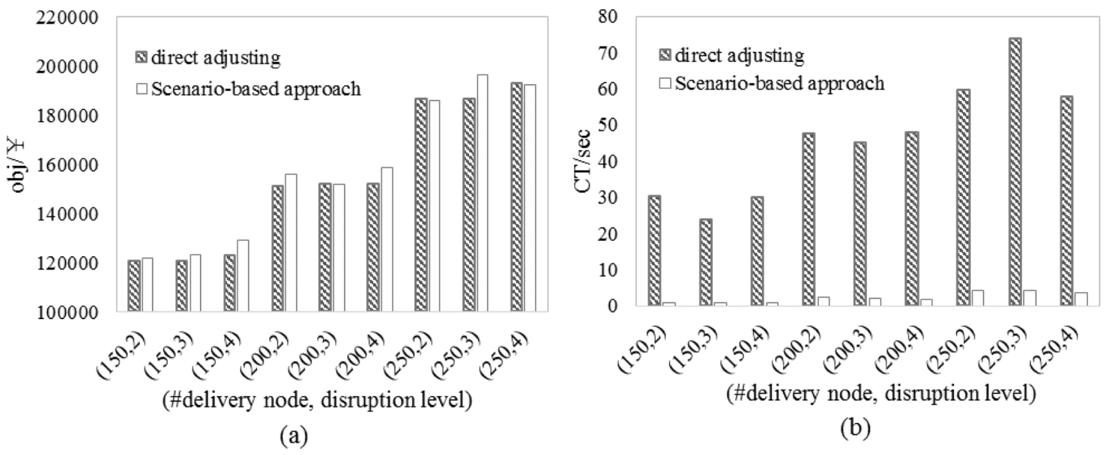

Fig. 4. Performance of the scenario-based approach under various number of delivery nodes and disruption levels.

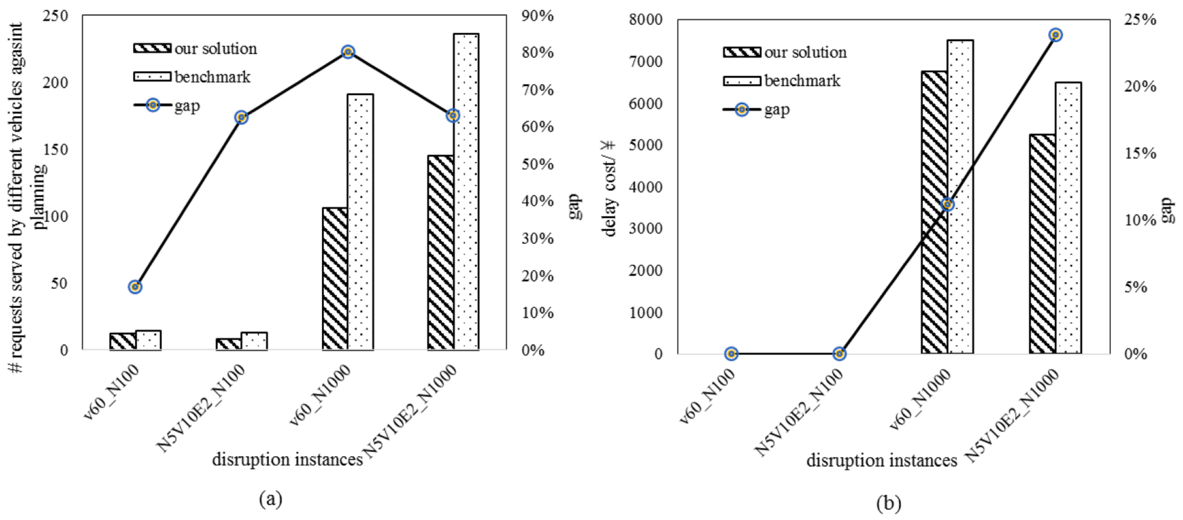

Fig. 5. Performance of solutions with and without considering delivery vehicle divergency.

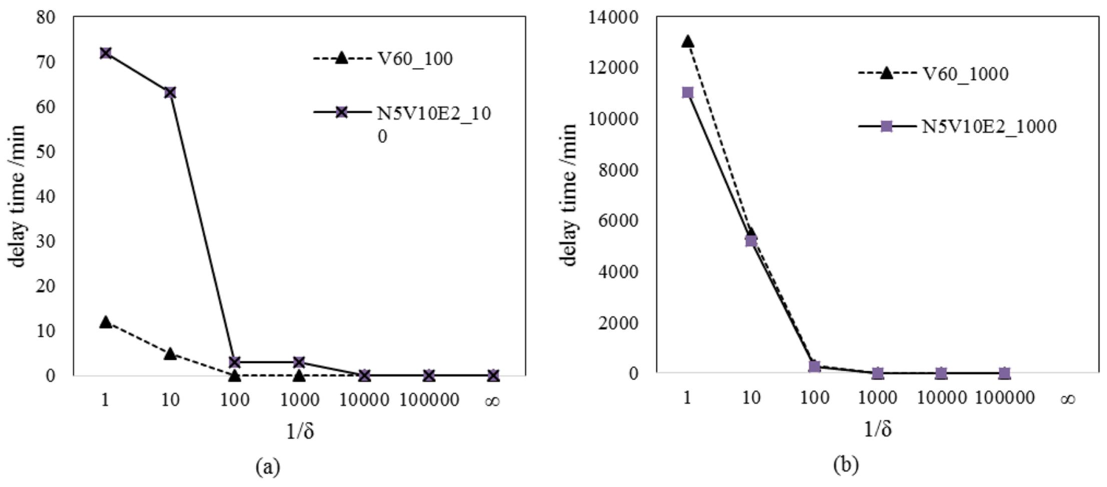

Fig. 6. Recovery of service delay under various time violation tolerance of customers.

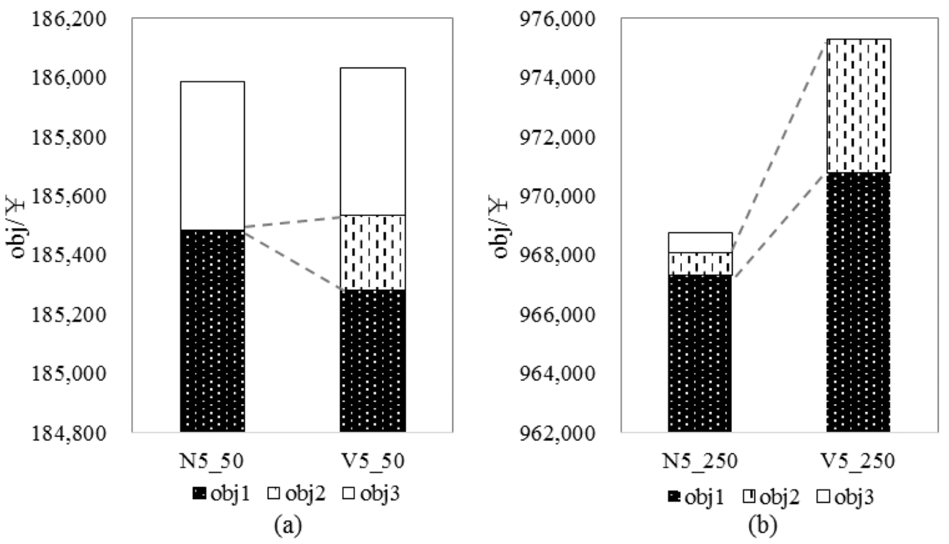

Fig. 7. Performance of recovery solutions against two types of disruptions.

5.5.3. Impact analysis of two types of disruptions on the proposed recovery solutions

It is challenging to compare the severity of various types of disruptions, since the temporal length of a disruption is measured differently—for traffic jams, it is the ratio between the new travel time and the original travel time; for node delays and vehicle breakdowns, it is the length of time the disruption lasts.

Despite the difficulty in measuring the severity of a disruption, we still attempt to give some preliminary analysis on how the proposed method performs against some types of disruptions in this section. Fig. 7 compares average total cost of the proposed recovery solutions under two different types of disruptions (5-min node delay N5 vs. 5-min vehicle breakdown V5). The total cost is the sum of three objective terms: the traveling cost (obj1), the deviation cost of request to vehicle assignment (obj2) and the request delay cost (obj3). Fig. 7(a) is for an instance with 50 requests and Fig. 7(b) shows the similar comparison with 250 requests. Both instances feature disruptions of the same scale \mathord{-} 8% of all the nodes (for N5) and vehicles (for V5) respectively.

Both Fig. 7(a and b) show that the total recovery cost for vehicle breakdown (V5) is higher than node service time change (N5). This is apparently due to the higher of deviation cost of requests to vehicle assignment (obj2) for the vehicle breakdown disruptions. In other words, vehicle breakdowns are more challenging to handle for our method than node disruptions\mathord{-} once a vehicle is unable to operate, nodes unserved by the disrupted vehicle will inevitably be affected. To avoid request delays for customers, the driver may have to transfer some requests to other available drivers nearby, which consequently increases the deviation cost of request to vehicle assignments (obj2).

6. Conclusions and future work

This paper presents a mixed-integer programming model and a novel solution framework for the disruption recovery problem for pickup and delivery with time windows (DR-PDPTW) in online food delivery services. In this problem, one pickup node may correspond to multiple requests, as one restaurant can serve multiple customers. Our model considers four types of disruptions that are common for food deliveries. We also add a necessity check module to decide if a disrupted route will lead to an unacceptable delay before providing recovery solutions. The proposed scenario-based solution framework attempts to generate disruption scenarios and corresponding solutions before disruptions. Once a disruption occurs, it will match the disruption with generated scenarios with the help of scenario clustering and then adjust candidate solutions based on the actual disruption. Its performance is validated via comparisons with the ALNS heuristic commonly used for PDPTW. Computational results show that without compromising solution quality, the proposed framework is much more efficient in providing solutions. The average response time is less than 1/5 of that of the ALNS heuristic. Such efficiencies are highly desirable for online food delivery services. Analyses of divergence from original plans also demonstrate the necessity of studying DR-PDPTW as a separate problem in the unique context of online food delivery.

There are also some directions for future research. First, the generation of disruption scenarios and candidate solutions would benefit from using real-world data. Second, since PDPTW can be solved by exact algorithms in certain cases, it would be interesting to combine exact algorithms and scenario-based methods to further improve the performance of recovery solutions while ensuring the computational efficiency. Last, this paper only considers short-term vehicle-related disruptions, such as flat tires or minor accidents, which can be fixed within 60 min or less. In the case of a vehicle complete breakdown, where the vehicle cannot resume normal operations any time soon, transferring orders on the vehicle to another becomes necessary. How to design recovery solutions to accommodate both short and long-term vehicle disruptions would be an interesting direction for future research.

Declaration of Competing Interest

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

Data availability

Relative data has been included in the paper and supplemental material

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (71801061), the Natural Science Foundation of Heilongjiang province (LH2021G003), the Social Science Foundation of Heilongjiang province (18GLC208), National Natural Science Foundation of China (72001055). Moreover, the authors are indebted to the anonymous reviewers for helpful comments and suggestions.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cor.2023.106337.

References

Bent, R., Van Hentenryck, P., 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper. Res. 52 (6), 977–987.

Berbeglia, G., Cordeau, J.F., Gribkovskaia, I., Laporte, G., 2007. Static pickup and delivery problems: a classification scheme and survey. TOP 15 (1), 1–31.

Berbeglia, G., Cordeau, F., Laporte, G., 2010. Dynamic pickup and delivery problems. Eur. J. Oper. Res. 202 (1), 8–15.

Bishop, C., 2006. Pattern recognition and machine learning. Springer Press, Washington.

Burgelman, J., Vanhoucke, M., 2020. Project schedule performance under general mode implementation disruptions. Eur. J. Oper. Res. 280 (1), 295–311.

Christiaens, J., Vanden Berghe, G., 2020. Slack induction by string removals for vehicle routing problems. Transp. Sci. 54 (2), 417–433.

Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pickup and delivery problem with time windows. Eur. J. Oper. Res. 54 (1), 7–22.

Eglese, R., Zambirinis, S., 2018. Disruption management in vehicle routing and scheduling for road freight transport: a review. Top 26 (1), 1–17.

Furtado, M.G.S., Munari, P., Morabito, R., 2017. Pickup and delivery problem with time windows: a new compact two-index formulation. Oper. Res. Lett. 45 (4), 334–341.

Ghiani, G., Manni, E., Thomas, B., 2012. A comparison of anticipatory algorithms for the dynamic and stochastic traveling salesman problem. Transp. Sci. 46 (3), 374–387.

Ghilas, V., Demir, E., Woensel, T.V., 2016. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. Comput. Oper. Res. 72, 12–30.

Giménez-Palacios, I., Parreño, F., Álvarez-Valdés, R., Paquay, C., Oliveira, B.B., Carravilla, M.A., Oliveira, J.F., 2022. First-mile logistics parcel pickup: Vehicle routing with packing constraints under disruption. Transp. Res. Pt. e-Logist. Transp. Rev. 164, 102812.

Hassan, L.K., Santos, B.F., Vink, J., 2021. Airline Disruption Management: A Literature Review and Practical Challenges. Comput. Oper. Res. 127, 105137.

Hu, X., Sun, L., Liu, L., 2013. A PAM approach to handling disruptions in real-time vehicle routing problems. Decis. Support. Syst. 54 (3), 1380–1393.

Hu, Y., Zhang, P.u., Fan, B.o., Zhang, S., Song, J., 2021. Integrated recovery of aircraft and passengers with passengers' willingness under various itinerary disruption situations. Comput. Ind. Eng. 161, 107664.

Jaillet, P., Wagner, M.R., 2008. Online vehicle routing problems: A survey. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), Operations Research/Computer Science InterfacesThe Vehicle Routing Problem: Latest Advances and New Challenges. Springer US, Boston, MA, pp. 221–237.

Jun, S., Lee, S., Yih, Y., 2021. Pickup and delivery problem with recharging for material handling systems utilising autonomous mobile robots. Eur. J. Oper. Res. 289 (3), 1153–1168.

Kai, G., Niklaus, C., Voss, S., 2004. Dispatching of an Electric Monorail System: Applying Metaheuristics to an Online Pickup and Delivery Problem. Transp. Sci. 38 (4), 434–446.

Karami, F., Vancroonenburg, W., Vanden Berghe, G., 2020. A periodic optimization approach to dynamic pickup and delivery problems with time windows. J. Scheduling 23 (6), 711–731.

Katsaliaki, K., Galetsi, P., Kumar, S., 2022. Supply chain disruptions and resilience: a major review and future research agenda. Ann. Oper. Res. 319 (1), 965–1002.

Koç, Ç., Laporte, G., Tükenmez, İ., 2020. A review of vehicle routing with simultaneous pickup and delivery. Comput. Oper. Res. 122, 104987.
Lai, Y., 2020. Delivery drivers trapped in the system. *Ren Wu* 8, 70–91.
Li, H., Lim, A., 2001. A metaheuristic for the pickup and delivery problem with time windows. 13th IEEE Conference of Tools Artificial Intelligence. November 7-9. Dallas, TX.
Li, C., Gong, L., Luo, Z., Lim, A., 2019. A branch-and-price-and-cut algorithm for a pickup and delivery problem in retailing. Omega-Int. J. Manage. S. 89, 71–91.
Li, J.Q., Mirchandani, P.B., Borenstein, D., 2009. Real-time vehicle rerouting problems with time windows. Eur. J. Oper. Res. 194 (3), 711–727.
Mu, Q., Eglese, R., 2013. Disrupted capacitated vehicle routing problem with order release delay. Ann. Oper. Res. 207 (1), 201–216.
Nanry, W.P., Barnes, J.W., 2000. Solving the pickup and delivery problem with time windows using reactive tabu search. Transp. Res. Pt. B-Methodol. 34 (2), 107–121.
Ngai, E.W.T., Leung, T.K.P., Wong, Y.H., Lee, M.C.M., Chai, P.Y.F., Choi, Y.S., 2012. Design and development of a context-aware decision support system for real-time accident handling in logistics. Decis. Support. Syst. 52 (4), 816–827.
Pillac, V., Guéret, C., Medaglia, A.L., 2012. An event-driven optimization framework for dynamic vehicle routing. Decis. Support. Syst. 54 (1), 414–423.
Ropke, S., Cordeau, J.F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transp. Sci. 43 (3), 267–286.
Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.
Ropke S., 2009. Parallel large neighborhood search-a software framework. MIC 2009: The VIII Metaheuristics International Conference. July 13-16. Hamburg, Germany.
Sartori, C.S., Buriol, L.S., 2020. A study on the pickup and delivery problem with time windows: matheuristics and new instances. Comput. Oper. Res. 124 (2), 105065.
Sawik, T., 2016. Integrated supply, production and distribution scheduling under disruption risks. Omega-Int. J. Manage. S. 62, 131–144.
Schyns, M., 2015. An ant colony system for responsive dynamic vehicle routing. Eur. J. Oper. Res. 245 (3), 704–718.
Shaw, P., 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Department of Computer Science. University of Strathclyde, Scotland.
Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. Springer, Berlin, Heidelberg.
Solomon, M., Desrosiers, J., 1988. Time window constrained routing and scheduling problems. Transp. Sci. 22 (1), 1–13.
Spliet, R., Gabor, A.F., Dekker, R., 2014. The vehicle rescheduling problem. Comput. Oper. Res. 43, 129–136.
Sun, P., Veelenturf, L.P., Hewitt, M., Woensel, T.V., 2018. The time-dependent pickup and delivery problem with time windows. Transp. Res. Pt. B-Methodol. 116, 1–24.
Sun, P., Veelenturf, L.P., Hewitt, M., Van Woensel, T., 2020. Adaptive large neighborhood search for the time-dependent profitable pickup and delivery problem with time windows. Transp. Res. Pt. E-Log. 138, 101942.
Vancroonenburg, W., Causmaecker, P.D., Berghe, G.V., 2016. A study of decision support models for online patient-to-room assignment planning. Ann. Oper. Res. 239 (1), 253–271.
Voccia, S.A., Campbell, A.M., Thomas, B.W., 2019. The same-day delivery problem for online purchases. Transp. Sci. 53 (1), 167–184.
Wagstaff, K., Cardue, C., Rogers, S., Schrödl, S., 2001. Constrained k-means clustering with background knowledge. The Eighteenth International Conference on Machine Learning. June 28. Williamstown.
Wang, X., Ruan, J., Shi, Y., 2012. A recovery model for combinational disruptions in logistics delivery: considering the real-world participators. Int. J. Prod. Econ. 140 (1), 508–520.
Wolfinger, D., 2021. A large neighborhood search for the pickup and delivery problem with time windows, split loads and transshipments. Comput. Oper. Res. 126, 105110.
Xiong, X., Wang, D., Cheng, T.C.E., Wu, C., Yin, Y., 2018. Single-machine scheduling and common due date assignment with potential machine disruption. Int. J. Prod. Res. 56 (3), 1345–1360.
Yu, G., Qi, X., 2004. Disruption management: framework, models and applications. World Scientific Books.