applied sciences

MDPI

Article

Efficient Delivery Services Sharing with Time Windows

Wanyuan Wang *, Hansi Tao and Yichuan Jiang

School of Computer Science and Engineering, Southeast University, Nanjing 211189, China; hstao@seu.edu.cn (H.T.); yjiang@seu.edu.cn (Y.J.)

* Correspondence: wywang@seu.edu.cn

Received: 16 September 2020; Accepted: 20 October 2020; Published: 22 October 2020

Abstract: Delivery service sharing (**DSS**) has made an important contribution in the optimization of daily order delivery applications. Existing DSS algorithms introduce two major limitations. First, due to computational reasons, most DSS algorithms focus on the fixed pickup/drop-off time scenario, which is inconvenient for real-world scenarios where customers can choose the pickup/drop-off time flexibly. On the other hand, to address the intractable DSS with the flexible time windows (**DSS-Fle**), local search-based heuristics are widely employed; however, they have no theoretical results on the advantage of order sharing. Against this background, this paper designs a novel algorithm for DSS-Fle, which is efficient on both time complexity and system throughput. Inspired by the efficiency of shareability network on the delivery service routing (**DSR**) variant where orders cannot be shared and have the fixed time window, we first consider the variant of DSR with flexible time windows (**DSR-Fle**). For DSR-Fle, the order's flexible time windows are split into multiple virtual fixed time windows, one of which is chosen by the shareability network as the order's service time. On the other hand, inspired by efficiency of local search heuristics, we further consider the variant of DSS with fixed time window (**DSS-Fix**). For DSS-Fix, the beneficial sharing orders are searched and inserted to the shareability network. Finally, combining the spitting mechanism proposed in DSR-Fle and the inserting mechanism proposed in DSS-Fix together, an efficient algorithm is proposed for DSS-Fle. Simulation results show that the proposed DSS-Fle variant algorithm can scale to city-scale scenarios with thousands of regions, orders and couriers, and has the significant advantage on improving system throughput.

Keywords: delivery service sharing; shareability network; approximation algorithm; time windows

1. Introduction

The popularization of smartphones has made people increasingly connected, and has provided the great convenience for daily order delivery service. For example, by using the ride-ordering apps, such as DiDi Chuxing (www.didiglobal.com) and Uber (www.uber.com), customers can be picked up by the private car or taxi at any time, and, by using the food-ordering apps, such as Meituan (www.meituan.com), Eleme (www.ele.me), and Freshhema (www.freshhema.com), food can be delivered to customers' homes within a very short time. During July 2019, the average daily ride orders of DiDi Chuxing broke 24 million, and the average daily food orders of Meituan broke 30 million. Delivery service routing (**DSR**), which routes the cars/couriers to deliver orders, has made such an important contribution of serving these millions of orders. To improve customer experience, a significant characteristic of the ordering apps is to provide the flexible pickup or drop-off windows [1,2]. For example, in the food-ordering apps, the customer has the flexibility to choose the food drop-off time, e.g., during the lunch time, it is feasible to drop off the food from 11:30 a.m. to 12:30 p.m. Moreover, to improve the system throughput

(i.e., the number of orders served), multiple orders can be shared and served by the same car/courier at a time [3–5]. For example, in the food-ordering scenarios, this food ordered from the same shop can be packed by a courier and delivered to the customers in a sharing fashion. Due to its potential in real-world applications, this paper mainly focuses on delivery service sharing variant with flexible pickup/drop-off time windows (**DSS-Fle**), where orders can be shared and served at a time.

Recently, many efforts have been devoted to addressinng two variants of DSS problem-DSR with flexible time windows (DSR-Fle) and DSS with a fixed time window (DSS-Fix). In the DSR-Fle literature, where orders cannot be shared, Vazifeh et al. [6] first propose a shareability network to model the special DSR variant with fixed pickup window (**DSR-Fix**), which can be addressed by the polynomial Hopcroft–Karp maximum-matching algorithm. Bertsimas et al. [7] extend to the NP-hard DSR-Fle, and propose an iterative heuristic algorithm of removing the undesirable edges in the shareability network in each iteration. Although the heuristic algorithm can scale to real-world city-scale scenarios, it has no theoretical guarantee on system throughput. On the other hand, a mixed integer program is used to formulate the DSS-Fix variant, and the time-consuming Lagrangian decomposition approach is proposed for the exact solution [8]. To reduce the computation complexity, a bipartite matching based approximation algorithm is proposed for matching a delivery resource with two orders [9], which might be inefficient for system throughput maximization. Ma et al. [10] propose a heuristic algorithm to generate the order sharing path, which works by finding and inserting the nearby orders into a current partial sharing path. The constructed sharing path might be efficient on system throughput and computation time; however, neither considers the flexible time windows nor provides any systemic analysis on the advantage of order sharing.

Against this background, the main contribution of this paper is to design a time-efficient algorithm for DSS-Fle, and to theoretically guarantee the advantage of order sharing. We extend the seminal shareability network mechanism (that is used in DSR-Fix), to DSR-Fle and DSS-Fix, respectively. For DSR-Fle, we split the flexible time windows into multiple virtual fixed times, carefully choosing one of these fixed times as the order's service time, and employ the DSR-Fix mechanism to generate the routing plan. The DSR-Fle variant algorithm can provide the theoretical $\frac{1}{\kappa}$ approximation ratio with respect to system throughput, where κ is the maximum time window of orders. For DSS-Fix, we first search the orders that can be shared, and insert these shared orders into the shareability network of DSR-Fix. Compared to the DSR-Fix variant without considering order sharing, DSS-Fix can improve the system throughput. Finally, we combine the flexible time windows spitting mechanism (proposed for DSR-Fle) and shareability network updating mechanism (proposed for DSS-Fix) to design an efficient algorithm for DSS-Fle. Besides the time efficiency, we systemically guarantee that order sharing is beneficial on maximizing system throughput, i.e., DSS-Fix $\succ$ DSR-Fix, and DSS-Fle $\succ$ DSR-Fle; the symbol $\succ$ indicates the prior relation on maximizing system throughput. More specifically, this paper advances the state of the art in the following ways. Compared with the most related DSR-Fle studies, which employ the local search heuristic, we propose the approximate algorithm with system throughput guarantee. Compared with most related DSS-Fix studies, which do not take the flexible time windows into account, we propose a time-efficient algorithm for DSS-Fle and systematically analyze the advantage of sharing on maximizing system throughput. Finally, we conduct a series of experiments on synthetic datasets to validate the proposed algorithms.

The remainder of this paper is organized as follows. In Section 2, we provide a review of the related literature on DSR with fixed/flexible time windows, and with/without sharing. In Section 3, we formulate the DSS-Fle problem and propose the algorithm framework. In Section 4, we propose four algorithms for the DSR-Fix, DSR-Fle, DSS-Fix, and DSS-Fle four variants, respectively. In Section 5, we conduct a series of experiments on synthetic datasets to validate the proposed algorithms efficiency and effectiveness. Finally, we conclude our paper in Section 6.

2. Related Work

In this section, we categorize the related work into two groups: delivery service routing (DSR) and delivery service sharing (DSS). The latter is further categorized into vehicle-oriented sharing and customer-oriented sharing two subgroups.

2.1. Delivery Service Routing (DSR)

The DSR problem is a combinatorial optimization problem seeking to serve a number of customers with a fleet of vehicles [1,2]. Let $G = \langle V, C \rangle$ denote a graph, where V denotes the nodes, say customers that need service, $c_{ij} \in C$ denotes the time distance between nodes v_i and v_j. Each customer v_i has a time window specified by an interval $[e_i, l_i]$, corresponding to the earliest and latest possible serving times, respectively. A vehicle route starts at the depot v_0, serving some number of customers at most once within their windows. Solomon [1] proposes a heuristic of inserting a new un-routed customer into the current partial route, between two adjacent customers with the minimum insertion cost. To further improve routing efficiency, Potvin and Roussea [11] propose a 2-opt exchange heuristic where the initial route is improved by exchanging two existing route links for another two new route links—being aware that some lateness customers can be tolerated, and the soft time window is introduced and a tabu search heuristic of swapping sequences of consecutive customers between two routes is proposed in [12]. To minimize the number of vehicles and traveling cost, a two-stage hybrid algorithm is proposed in [13], in which the simulated annealing heuristic is used to minimize the number of vehicles, and the neighborhood search is used to minimize travel costs. Considering the necessity of visiting recharging stations in electric DSR problem, meta-heuristic search [14] and branch-price-and-cut-based exact algorithms [15] are proposed to minimize the route cost. To minimize the average distance traveled per customer, the weighted DSR is introduced [16]. For weighted DSR, the regret-insertion heuristic is proposed to construct the initial solution and iterative local link exchange heuristics is proposed to improve the routing efficiency. In the above DSR literature, customers have the same pickup location (i.e., the depot), and each drop-off location is modeled as a node, which can be served successfully once it is visited by a vehicle. In contrast to the existing DSR with the same pickup location, this paper studies the delivery service sharing (DSS) variant where the customer is specified by the origin, destination, and flexible pickup/drop-off time windows, and the customer can be served successfully if and only if he is picked up from its origin and delivered to its destination with the time window.

2.2. Delivery Service Sharing (DSS)

For the DSS group, we mainly focus on vehicle-oriented DSS and two customer-oriented DSS subgroups. For vehicle-oriented DSS, some customers are served in sequence by the same vehicle, and for customer-oriented DSS some customers can be shared and served by the same vehicle at the same time.

2.2.1. Vehicle-Oriented DSS

Each customer c can be represented by a tuple $(v_c^p, v_c^d, [t_c^e, t_c^l])$, where v_c^p denotes the pick-up location, v_c^d denotes the drop-off location, and the time window $[t_c^e, t_c^l]$ represents its minimal and maximal possible pickup/drop-off times [17]. In the vehicle-based DSS, each vehicle can only serve one customer at a time, and customers are served in sequence. For the special scenario with the fixed pickup/drop-off time, i.e., $t_c^e = t_c^l = t_j$, the vehicle-shareability network is constructed in [18]. In the shareability network, each node represents a customer, and there is an directed edge between customers c and c' if and only if c' can be served immediately after c', i.e., $t_c + t_{cc'} \leq t_{c'}$, where $t_{cc'}$ denotes the time distance of traveling from c's destination to the origin of c' and delivering c' to its destination. Given the constructed shareability

network, the Hopcroft–Karp algorithm [19] is utilized to maximize the number of customers served with n vehicles [6]. Considering the flexible pickup windows $[t_c^e, t_c^l]$, Bertsimas et al. [7] propose an iterative Backbone algorithm, and, at each iteration, the fixed time is chosen randomly within the time window and the desirable edges are remained. The optimal mixed integer programming is used to return the optimal solution within the remained shareability network. Being aware of the soft constraints of minimizing the delay of the fixed time window, a linear programming-based approximation algorithm is proposed in [20], and a heuristic local search strategy with a diversification updating strategy is used to improve the solution quality iteratively [21]. Compared with existing heuristics for vehicle-oriented DSS with flexible time windows, we propose an approximation algorithm which has theoretical performance guarantee on system throughput. Moreover, we also systemically analyze the customer-oriented sharing scenarios, where multiple customers can be served by a vehicle at the same time.

2.2.2. Customer-Oriented DSS

Customer-oriented sharing has been proposed as a promising means for improving DSS applications (e.g., ridesharing), where multiple customer requests can be shared and severed by a vehicle simultaneously [3–5]. A mixed integer programming can be used to model the static customer-oriented DSS where the customer requests are known in advance, and a time-consuming Lagrangian decomposition approach is used to return the exact solution [8]. To scale up the customer-oriented DSS, Bei et al. [9] propose a bipartite matching-based approximation algorithm. By considering the advantage of tip sharing with partners locating in proximal locations, a community-based trip-sharing paradigm is proposed, where customers are first clustered into local communities and optimal intra community trip sharing can be found by the integer programming [22]. By utilizing the offline partial paths, online nearby customer requests can be constructed and inserted by a local search in a real-time manner [23–25]. On the other hand, in ridesharing, the sharing paths should also satisfy the monetary constraints where passengers will not pay more compared with the scenario without ridesharing and taxi drivers will also make money for all the detour distance due to ridesharing [10]. Compared to the above customer-oriented DSS scenarios where customers have the fixed pickup/drop-off window, we study the more practical and flexible DSS variant where customers have time windows. Moreover, we design customer-oriented DSS algorithms with the systemic understanding of the power of customer sharing.

3. Model and Algorithm Framework

In this section, we first model the problem of delivery service sharing with flexible time windows, and outline the algorithm framework. Table 1 summarizes the notations used throughout the paper.

Table 1. Useful notations.

Notation	Description
$A = \{a_1, a_2, \cdots, a_n\}$	the set of n couriers available for food delivery services
$V = \{v_1, v_2, \ldots, v_m\}$	the set of m regions of a city
$R = \{r_1, r_2, \cdots, r_l\}$	the set of l customer orders need in service
$[t_j^e, t_j^l] \in [1, T]$	the time window of the order r_j
v_j^p	the pickup region of the order r_j
v_j^d	the drop-off region of the order r_j
τ_{jk}	the time distance of traveling from the drop-off region of r_j to the pickup region of r_k and delivering r_k to its drop-off region

3.1. The Model

Food-Ordering Delivery Applications. The model is mainly motivated by food-ordering applications, such as Meituan, Eleme, and Freshhema. There are a set of n couriers $A = \{a_1, a_2, \cdots, a_n\}$ available for food delivery services. Each courier a_i has a capacity $c \in \mathbb{N}_+$, indicating the maximum number of customer orders that can serve at a time (Here, we assume that each courier has the homogeneous delivery capacity, which can be easily extended to heterogeneous capacity scenario by making a match between the courier and delivery plans). Each day is discretized into periods; each period includes δ minutes (e.g., 5 min). The daily horizon $\mathcal{T} = \{1, 2, \cdots, T = \frac{24 \times 60}{\delta}\}$, and the parameter δ can be tuned such that $\frac{24 \times 60}{\delta}$ is an integer.

City Network. Let $G = \langle V, E, W, D \rangle$ denote the weighted city network, where $V = \{v_1, v_2, \ldots, v_m\}$ indicates m regions in the city and each $e_{ij} \in E$ indicates the edge between regions v_i and v_j. The time distance on edges $W = \{w_{ij}\}_{e_{ij} \in E}$ indicates that it will take w_{ij} periods to travel from v_i to v_j, and $w_{ii} = 0, \forall v_i$. Let $D = \{d_{ij}\}_{v_i, v_j \in V}$ denote the time distance of the shortest path from v_i to v_j. The time distance matrix D satisfies the triangle inequality, i.e., $d_{ik} \leq d_{ij} + d_{jk}, \forall v_i, v_j$ and v_k.

Customer Orders. Let $R = \{r_1, r_2, \cdots, r_l\}$ be a collection of customer orders. Each $r_j \in R$ is defined as a tuple $(v_j^p, v_j^d, [t_j^e, t_j^l])$, where $v_j^p \in V$ denotes the pick-up region of r_j, and $v_j^d \in V$ denotes the drop-off region. Each r_j has a flexible drop-off time window $[t_j^e, t_j^l] \subseteq [1, T]$, during which a customer wishes to finish her order. For example, $[t_j^e, t_j^l]$ = [16:30, 17:30] indicates that the food delivery order r_j must be delivered to customer home between the time 4:30 p.m. and 5:30 p.m.

Delivery Service Sharing with Flexible Time Windows (DSS-Fle). A delivery sharing plan $\mathcal{P}$ is a temporally-ordered route of pickup and drop-off regions of orders. Let $R_{\mathcal{P}} \subseteq R$ denote the subset of orders served in $\mathcal{P}$, and for each $r_j \in R_{\mathcal{P}}$, v_j^p should precede v_j^d, and the drop-off time should be within its flexible windows $[t_j^e, t_j^l]$. A courier is responsible for a plan, and for each plan, there are at most c orders in service at a time. DSS-Fle is formally defined as follows: given the set of n couriers A and the set of l customer orders R, find the n order-disjoint delivery sharing plans with the objective of optimizing order service rate, i.e., maximizing the number of orders served.

3.2. The Algorithm Framework

Before presenting the algorithm for DSS-Fle, we first study the following three special DSS-Fle variants:

- Delivery service routing with the fixed window (**DSR-Fix**). We first consider the DSR-Fix variant where orders have the fixed drop-off window and there is no order sharing in routing plan. In terms of the fixed drop-off window, we mean that each order r_j has a fixed drop-off time window, i.e., $t_j^e = t_j^l = t_j$, and each order can only be finished at its fixed window t_j. In terms of without order sharing, we mean that orders are served in sequence and each courier can only serve one order at a time.
- Delivery service routing with the flexible windows (**DSR-Fle**). In DSR-Fle, each order has a flexible drop-off time window $[t_j^e, t_j^l]$, but they are still cannot be shared by a courier in the same service time.
- Delivery service sharing with the fixed window (**DSS-Fix**). In DSS-Fix, multiple orders can be shared and served at a time, but each order r_j has the fixed drop-off time window t_j.

Inspired by the efficiency of a shareability network (proposed in [6,18]) on addressing DSR-Fix, we propose an approximation algorithm and a time efficient heuristic algorithm for DSR-Fle and DSS-Fix, respectively. Combining DSR-Fle and DSS-Fix algorithms together, we finally propose the DSS-Fle algorithm, which is efficient on both computation complexity and order service rate. The algorithm framework for DSS-Fle is outlined in Figure 1.

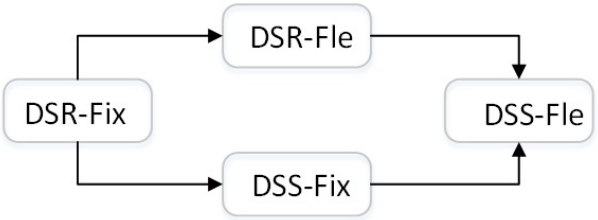


Figure 1. The algorithm framework for DSS-Fle. Derived from DSR-Fix, we first propose DSR-Fle and DSS-Fix algorithms, and derived from DSR-Fle and DSS-Fix together, we propose the DSS-Fle algorithm.

4. The Algorithms

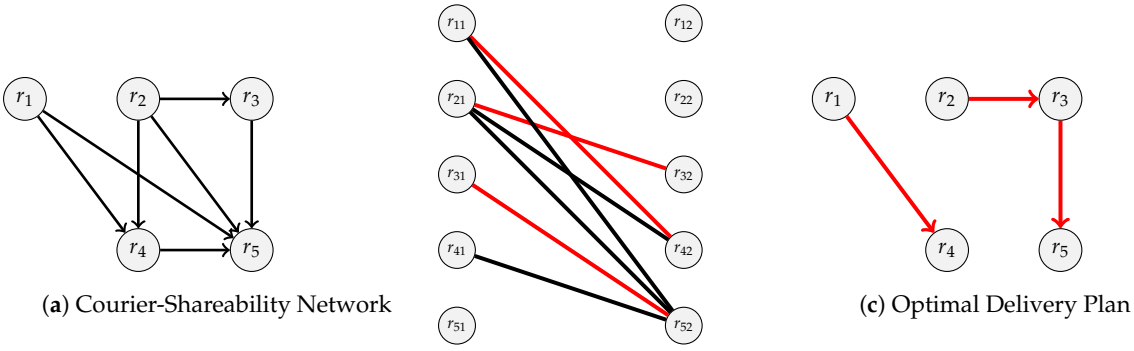
In this section, we propose four algorithms for DSR-Fix, DSR-Fle, DSS-Fix, and DSS-Fle variants, respectively, and theoretically analyze these algorithms' properties.

4.1. The DSR-Fix Variant Algorithm

In DSR-Fix, each order r_j has the fixed drop-off window, i.e., $t_j^e = t_j^l = t_j$, i.e., a courier must deliver r_j to the drop-off region v_j^d before t_j and wait until t_j to drop off it. Moreover, orders cannot be shared and must be served in sequence. For any two orders r_j and r_k, they can be served consecutively if one courier can pick up the order r_k immediately after finishing the order r_j. There must be a long enough period of time between drop-off periods of r_j and r_k such that $t_j + \tau_{jk} \leq t_k$, where $\tau_{jk} = d_{v_j^d v_k^p} + d_{v_k^p v_k^d}$ denotes the time distance of traveling from the drop-off region of r_j to the pickup region of r_k and delivering r_k to its drop-off region v_k^d. For such a DSR-Fix problem, an efficient courier-shareability network (**CSN**)-based algorithm is proposed in [6].

Definition 1. *Courier-shareability network (**CSN**). The CSN is a directed network CSN=(V, E), where each node $v_j \in V$ corresponds to an order r_j. The directed edge $(v_j, v_k) \in E$ if and only if r_j and r_k can be served consecutively, i.e., $t_j + \tau_{jk} \leq t_k$.*

The existence of an edge (v_j, v_j) in the CSN indicates that the two orders r_j and r_k can be served consecutively, and a delivery plan in CSN corresponds to a sequence of orders that can be served by a courier. Solving the DSR-Fix problem is equivalent to finding n node-disjoint paths in such a way to maximize the cover of V in the constructed CSN [18]. The problem of finding the maximum n node-disjoint paths cover on directed acyclic graphs can be solved efficiently by using the Hopcroft–Karp maximum-matching algorithm [26]. Algorithm 1 describes the CSN-based DSR-Fix algorithm, in which the Hopcroft–Karp maximum matching algorithm is shown in Algorithm 2. In Algorithm 2, Steps 1–2, for each order r_j, we create an in-order r_{j1} and an out-order r_{j2} in the bipartite graph G^β. In Steps 3–5, for each directed edge (r_j, r_k) in the CSN, we create an edge between the in-order r_{k1} and out-order r_{j2} in the bipartite graph G^β. Given the bipartite graph G^β, the Hopcroft–Karp algorithm is utilized to return the maximum matchings $\mathcal{M}$. In Algorithm 1, top n node-disjoint delivery plans correspond to top n longest consecutive matchings. Figure 2 shows a toy example on how to construct the CSN and how the optimal delivery path can be found by the Hopcroft–Karp maximum-matching algorithm.



(**a**) Courier-Shareability Network

(**b**) Maximum Matching

(**c**) Optimal Delivery Plan

Figure 2. An illustration on how CSN works. (**a**) The directed edge $(r_j,\ r_k)$ indicates that the order r_k can be finished immediately after r_j; (**b**) the Hopcroft–Karp maximum-matching (red arrows); (**c**) the optimal delivery plans $r_1 \rightarrow r_4$ and $r_2 \rightarrow r_3 \rightarrow r_5$.

Algorithm 1: CSN-based DSR-Fix (**CSN-DSR-Fix**).

Input : Customer Orders R.
Output: Served orders $R' \subseteq R$.

1 Construct the CSN $G(R, E)$;
2 Invoke Algorithm 2 to find top n node-disjoint delivery plans;
3 Return the served orders R';

Algorithm 2: Hopcroft–Karp Maximum Matching Algorithm.

Input : CSN $G(R, E)$.
Output: Order Matching $\mathcal{M}$.

1 **for** $r_j \in R$ **do**
2 Create an in-order r_{j1} and an out-order r_{j2} in the bipartite graph G^β;
3 **for** $r_j, r_k \in R$ **do**
4 **if** $(r_j, r_k) \in E$ **then**
5 Create an edge between the in-order r_{j1} and out-order r_{k2} in G^β;
6 Invoke Hopcroft–Karp algorithm [26] to return the maximum matchings $\mathcal{M}$.

4.2. The DSR-Fle Variant Algorithm

In this section, we formulate the DSR-Fle problem where orders have flexible drop-off time windows, prove that the DSR-Fle problem is NP-hard, and propose an approximation algorithm for the DSR-Fle.

4.2.1. Problem Formulation

We use an integer programming (IP) to formulate the DSR-Fle problem where each order r_j has flexible drop-off time windows $[t_j^e,\ t_j^l]$. The following decision variables are necessary:

- $x_{ij} \in \{0, 1\}$, set to 1 if order r_j is served by the courier a_i as a first order;
- $y_{kj} \in \{0, 1\}$, set to 1 if order r_j is served immediately after r_k;
- $z_j \in \{0, 1\}$, set to 1 if order r_j is served by a courier;
- $\theta_j \in [t_j^e,\ t_j^l]$, the drop-off period of order r_j.

The following IP can be used to formulate the DSR-Fle.

$$\max \sum_{r_j \in R} \sum_{a_i \in \mathcal{A}} x_{ij} + \sum_{r_k, r_j \in R} y_{kj} \tag{1}$$

$$z_j = \sum_{a_i \in \mathcal{A}} x_{ij} + \sum_{r_k} y_{kj}, \forall r_j, \tag{2}$$

$$\sum_{r_j \in R} y_{kj} \leq z_k, \forall r_k, \tag{3}$$

$$\sum_{r_j \in R} x_{ij} \leq 1, \forall a_i, \tag{4}$$

$$\begin{cases} t_j^e \leq \theta_j \leq t_j^l, \forall r_j, \\ \theta_j - \theta_k \geq (t_j^e - t_k^l) + (\tau_{kj} - (t_j^e - t_k^l)) y_{kj}, \forall r_j, r_k, \\ \theta_j \geq t_j^e + (\tau_{ij} - t_j^e) x_{ij}, \forall a_i, r_j. \end{cases} \tag{5}$$

Equation (1) is the object of maximizing the number of orders served. Constraint (2) ensures that the order r_j is served (i.e., $z_j = 1$) if and only if a courier a_i serves him as the first customer order (i.e., $x_{ij} = 1$) or after another order r_k (i.e., $y_{kj} = 1$). Constraint (3) ensures that the order r_j that can be served after the order r_k iff r_k is served (*i.e.*, $z_k = 1$). Constraint (4) ensures that each courier can only serve one request as the first customer order. Finally, we enforce the drop-off time window constraints in Constraint (5): the first constraint ensures that the drop-off period θ_j must lie in the window $[t_j^e, t_j^l]$; the second constraint ensures that r_j can be served immediately after r_k iff there are long enough periods between θ_j and θ_k such that the courier can travel from the drop-off region of r_k to the pickup region of r_j and deliver r_j to its drop-off region, the third constraint ensures that the courier a_i can serve the order r_j as the first order iff there are long enough periods for a_i to travel from his region to the pickup and drop-off regions of r_j and $\tau_{ij} = d_{v_{a_i} v_j^p} + d_{v_j^p v_j^d}$, $\forall a_i, r_j$, where v_{a_i} indicates the region where a_i starts from.

Theorem 1. *The DSR-Fle problem is NP-hard.*

Proof. We show a reduction from the three-dimensional perfect matching problem (3DM), which is known to be NP-hard [27]. Recall that 3DM is as follows: given three finite and disjoint sets I, J and K, each of size n, and a subset $T \subseteq I \times J \times K$ with size $m \geq n$ (i.e., T consists of triples (i, j, k) such that $i \in I$, $j \in J$, and $k \in K$), the 3DM problem asks if there exists a perfect matching with a subset $M \subseteq T$ with n triples, such that every element in $I \cup J \cup K$ occurs in exactly one triple of M.

For any 3DM instance, $\mathcal{I} = \langle I, J, K, T \rangle$. We construct a corresponding DSR-Fle instance as follows: for each element $i \in I$, we create an order r_i with the fixed drop-off window t_i. For each element $j \in J$, we create an order r_j, and, for each tuple $(\cdot, j, \cdot) \in T$, we create an available drop-off time $t(\cdot, j, \cdot)$ for r_j. For each element $k \in K$, we create an order r_k with the fixed drop-off window t_k. For each tuple $(i, j, k) \in T$, we create a directed edge from the order r_i to the order r_j, and a directed edge from r_j to the order r_k. The directed edge from r_i to r_j indicates that it is feasible to travel from r_i's drop-off region (at the period t_i) to r_j's pickup and drop-off region before the period $t(i, j, k)$. The edge directed from r_j to r_k indicates that it is feasible to travel from r_j's drop-off region (at the period $t(i, j, k)$) to r_k's pickup and drop-off region before the period t_k. The region of the orders r_i, r_j and r_k and time distance among orders can be easily constructed to satisfy the shareability requirements. Moreover, there are n couriers in the DSR-Fle problem. The construction can be done in the polynomial time. We will show that the DSR-Fle can serve $3n$ orders iff 3DM has a perfect matching.

The 'if' direction: Assume that 3DM has a perfect matching $M \subseteq T$ with n triples. In the DSR-Fle, for each courier, it can serve three orders r_i, r_j, and r_k consecutively, where $(i, j, k) \in M$. For each r_i and r_k, it is only served exactly once. This is because the drop-off windows of r_i and r_k are fixed, and, for r_j, it is only served once since element j occurs exactly in one triple of M.

The 'only if' direction: Assume that these n couriers can serve $3n$ orders in the DSR-Fle problem, where each courier's delivery plan is $\langle r_i, r_j, r_k \rangle$. Then, in 3DM, the n delivery plans correspond to the n triples (i, j, k), which is a perfect matching. This is because each element in $i \in I$, $j \in J$, and $k \in K$ occurs exactly once. □

4.2.2. The Approximation Algorithm

The main idea behind the approximation algorithm is that, for the order r_j, we split its flexible windows $[t_j^e, t_j^l]$ into $\kappa_j = t_j^l - t_j^e + 1$ virtual orders $\tilde{r}_j = \{\tilde{r}_{j1}, \tilde{r}_{j2}, \cdots, \tilde{r}_{j\kappa_j}\}$, each virtual order $\tilde{r}_{jh}$ has the fixed drop-off window $t_j^e + h - 1$, $1 \leq h \leq \kappa_j$. For DSR-Fix with the set of virtual orders $\tilde{R} = \{\tilde{r}_1, \tilde{r}_2, \cdots, \tilde{r}_m\}$, we can adopt Algorithm 1 to optimize the order service. Let $\tilde{R}' \subseteq \tilde{R}$ denote the set of virtual orders served by Algorithm 1. For these virtual orders $\{\tilde{r}'_{j1}, \tilde{r}'_{j2}, \cdots, \tilde{r}'_{j\kappa_j}\}$ that are served in $\tilde{R}'$ and belong to the same real order r_j, randomly choosing one virtual order $\tilde{r}'_{jh}$ $(1 \leq h \leq \kappa_j)$ as the real order r_j's drop-off period, i.e., $t_j^e + h - 1$. The details of the approximation algorithm are presented in Algorithm 3. In Steps 2–3, each real order $r_j \in R$ is split into κ_j virtual orders, and the set of virtual orders is denoted by $\tilde{R}$. In Step 4, Algorithm 1 is utilized to return the served orders $\tilde{R}' \in \tilde{R}$. Finally, in Steps 5–6, one of these virtual orders $\tilde{r}_j^h$ that belongs to the same real order r_j is chosen such that r_j is finished at the drop-off period $t_{jh} = t_j^e + h - 1$.

Algorithm 3: Splitting-Based Approximation Algorithm for DSR-Fle (**Spl-DSR-Fle**).

Input : Customer Orders R.
Output: Served orders $R' \subseteq R$.

1 $R' = \emptyset$;
2 **for** $r_j \in R$ **do**
3 Split r_j into $\kappa_j = t_j^l - t_j^e + 1$ virtual orders $\tilde{r}_j = \{\tilde{r}_{j1}, \tilde{r}_{j2}, \cdots, \tilde{r}_{j\kappa_j}\}$;
4 Invoke Algorithm 1 to return the served orders $\tilde{R}'$=CSN-DSR-Fix($\tilde{R}$);
5 **for** $r_j \in R : \exists \tilde{r}'_{jh} \in \tilde{R}'$ **do**
6 $R' = R' \cup r_j$ and the drop-off period of r_j is $t_j^e + h - 1$;

Theorem 2. *Let $\kappa = \max_{r_j \in R} \kappa_j$ denote the maximum window length of orders R, then the number of orders achieved from the Algorithm 3 at least $\frac{1}{\kappa}$ of the optimal value returned by Equations (1)–(5).*

Proof. Let Opt denote the optimal number of orders served in DSR-Fle, Opt_{spl} denote the optimal number of orders served in the "split" DSR-Fle variant, where each order r_j is split into κ_j virtual orders, and Alg_{spl} denotes the number of orders served by Algorithm 3. When no ambiguity is possible, we let Opt, Opt_{spl} and Alg_{spl} denote the number of orders served as well the order delivery plan. In the following, we will prove that $Opt \leq Opt_{spl}$ and $Alg_{spl} \geq \frac{Opt_{spl}}{\kappa}$, respectively.

$Opt \leq Opt_{spl}$. Let $Opt = \{Opt^1, Opt^2, \cdots, Opt^n\}$, where Opt^i denotes the order delivery plan of the courier a_i. It should be noted that Opt is also a feasible delivery plan in the "split" DSR-Fle variant. On the other hand, virtual orders served in Opt_{spl} might belong to the same real orders, and a feasible delivery

plan of Opt_{spl} does not necessarily correspond to a feasible delivery plan of Opt. Therefore, we have $Opt \leq Opt_{spl}$.

$Alg_{spl} \geq \frac{Opt_{spl}}{\kappa}$. Let $Opt_{spl} = \{Opt^1_{spl}, Opt^2_{spl}, \cdots, Opt^n_{spl}\}$, where Opt^i_{spl} denotes the order delivery plan as well as the orders served by a_i in the "split" DSR-Fle. For any delivery plan Opt^i_{spl}, and any consecutive three virtual orders $\tilde{r}_{jo}, \tilde{r}_{kp}$ and $\tilde{r}_{hq}$ served by a_i, it is feasible for a_i to serve any two virtual orders of $\tilde{r}_{jo}, \tilde{r}_{kp}$ and $\tilde{r}_{hq}$. This is because the triangle inequality $\tau_{j_o k_p} + \tau_{k_p h_q} \leq \tau_{j_o h_q}$, where j_o, k_p, and h_q are the virtual orders with the drop-off period $t^e_j + o - 1$, $t^e_k + p - 1$ and $t^e_h + q - 1$, respectively. This indicates that, for any delivery plan Opt^i_{spl}, removing any number of virtual orders does not break the shareability of other remained virtual orders. Therefore, in Algorithm 3, for each real order r_j (with κ_j virtual orders) that are served in the "split" DSR-Fle variant, at least one of its virtual orders $\tilde{r}_{jh}$ remains. Therefore, we have $Alg_{spl} = \sum_{r_j} 1_{r_j \in R'} \geq \sum_{\tilde{r}_j} \sum_{\tilde{r}_{jh} \in \tilde{r}_j} \frac{1_{\tilde{r}_{jh} \in \tilde{R}'}}{\kappa_j} \geq \frac{Opt_{spl}}{\kappa}$, where the function $1_{f(\cdot)}$ returns value 1 when the function $f(\cdot)$ is true; otherwise, it returns zero, $\tilde{R}'$ is the set of orders served in Opt_{spl}, and R' is the set of orders served in Alg_{spl}.

Combining the above two conclusions, we have that $\frac{Alg_{spl}}{Opt} \geq \frac{Alg_{spl}}{Opt_{spl}} \geq \frac{1}{\kappa}$. □

Theorem 2 generalizes the DSR-Fix variant (i.e., $\kappa = 1$) that Algorithm 1 can achieve the optimal solution [6].

4.2.3. Improvement on the Spl-DSR-Fle Algorithm

We extend Algorithm 3 and propose an iterative algorithm to further improve system throughput. At each iteration g, let R^{fix}_g denote "fixed" orders, i.e., the set of orders whose drop-off period is determined and fixed, and R^{fle}_g denote the "flexible" orders, i.e., the set of orders whose drop-off period is flexible within windows $[t^e_j, t^l_j]$. For each flexible order $r_j \in R^{fle}_g$, we split its flexible time windows into $\kappa_j = t^l_j - t^e_j + 1$ virtual orders $\tilde{r}_j = \{\tilde{r}_{j1}, \tilde{r}_{j2}, \cdots, \tilde{r}_{j\kappa_j}\}$, and each $\tilde{r}_{jh}$ has the fixed drop-off period $t^e_j + h - 1$, $1 \leq h \leq \kappa_j$. At each iteration g, let $\tilde{R}^{fle}_g$ denote the virtual orders split from R^{fle}_g and $\tilde{R}_g = R^{fix}_g \bigcup \tilde{R}^{fle}_g$ denote all of the orders. Given $\tilde{R}_g$, we construct the corresponding CSN $G(\tilde{R}_g, E)$ and invoke Algorithm 3 to return the fixed orders R^{fix}_g that are served at the current iteration g. The details of the iterative DSR-Fle are proposed in Algorithm 4. In Step 3, the set of orders $\tilde{R}_g$ is constructed, which includes the set of fixed orders R^{fix}_g whose drop-off period is fixed, and the set of virtual orders $\tilde{R}^{fle}_g$ split from R^{fle}_g. In Step 4, the CSN $G = (\tilde{R}_g, E)$ is constructed with respect to $\tilde{R}_g$. In Step 5, Algorithm 3 is invoked to return the fixed orders served, R^{fix}_g. In Step 6, the set of flexible orders R^{fle}_g is updated by removing the fixed orders R^{fix}_g whose drop-off period has been determined. The iteration terminates until there is no improvement over previous iteration $g - 1$ on the number of orders served.

Lemma 1. *Algorithm 4 can always converge within finite iterations.*

Proof. Assume that, at the iteration g, if there is no improvement in the number of orders over that of the previous iteration $g - 1$, Algorithm 4 terminates and converges. Otherwise, it will always increase at least one order at each iteration; this is because each iteration generates a feasible solution of serving a set of real orders. On the other hand, there are at most $|R|$ orders to be served, thus, Algorithm 4 can always converge within at most $|R|$ iterations. □

Algorithm 4: Iterative Improvement over Spl-DSR-Fle (**Ite-DSR-Fle**).

Input : Customer orders R.
Output: Served orders $R' \subseteq R$.

$g = 0$, $R_g^{fix} = \emptyset$, and $R_g^{fle} = R \setminus R_g^{fix}$;
repeat
  g=g+1;
  Construct orders $\tilde{R}_g = R_{g-1}^{fix} \bigcup \tilde{R}_{g-1}^{fle}$;
  Construct the CSN $G = (\tilde{R}_g, E)$;
  Invoke Algorithm 3 to return the "fixed" orders R_g^{fix};
  $R_g^{fle} = R \setminus R_g^{fix}$;
until $|R_{g-1}^{fix}| = |R_g^{fix}|$;
Return $R' = R_g^{fix}$.

4.3. The DSS-Fix Variant Algorithm

In this section, we formulate the DSS-Fix problem, prove the DSS-Fix problem is NP-hard, and propose an efficient heuristic algorithm for DSS-Fix.

4.3.1. Problem Formulation and Complexity Analysis

In DSS-Fix, each order r_j has the fixed drop-off time window t_j, but multiple orders can be shared and served by a courier at a time. For example, orders r_j and r_k with proximal pickup regions can be picked up consecutively by a courier, and the sharing delivery plan can be $v_j^p \rightarrow v_k^p \rightarrow v_j^d \rightarrow v_k^p$. We say such a sharing plan is feasible if and only if (1) at any region, the number of orders in service is smaller than the courier capacity c, and (2) the courier can drop-off the order r_j (resp. r_k) at v_j^d (resp. v_k^d) no later than t_j (resp. t_k), but can before t_j (resp. t_k) while waiting until t_j (resp. t_k). The DSS-Fix problem is to find the optimal n sharing delivery plans to maximize the order service rate.

Theorem 3. *The DSS-Fix problem is NP-hard.*

Proof. We show a reduction from the metric Traveling Salesman Problem (**TSP**), which is known to be NP-hard. Recall that a metric TSP problem is as follows: given a weighted complete graph $G = (V, W)$, where $v_i \in V$ ($|V| = n$) is the vertex and $w_{ij} \in \mathbb{N}_+$ is the non-negative cost associated with the edge e_{ij} between v_i and v_j, the TSP asks if there exists a Hamiltonian path (i.e., a path visiting each vertex exactly once) with a cost equal to K. For any metric TSP instance $\mathcal{I} = \langle G, K \rangle$, we construct a DSS-Fix instance as follows. For each vertex $v_i \in V$, we create an order $r_i = (v_i, v^-, K)$, where v_i indicates the pickup region, v^- is the created auxiliary region indicating the drop-off region of r_i, and K indicates the drop-off period. The weight of the edge w_{ij} corresponds to the time distance between the pickup regions of r_i and r_j. Moreover, for each order r_i, the time distance between its pickup region v_i and its fixed drop-off region v^- is zero. In DSS-Fix, there is only one courier with capacity n. The courier stats from the depot region v_0 and the time distance between v_0 and any other region is zero. This construction can be done in polynomial time. We can conclude that the DSS-Fix can serve n orders within the period horizon K iff TSP has a Hamiltonian path with cost K. □

4.3.2. The Heuristic Algorithm

The main idea behind the heuristic algorithm for DSS-Fix is that we first search these orders that can be shared, then package the shared orders as a virtual order, and finally insert the virtual order into the DSR-Fix CSN.

Order Sharing Search. For two orders r_j and r_k whose pickup (resp. drop-off) regions that are proximal, it will be beneficial to pick up (resp. drop off) r_j and r_k consecutively. Imposing a bound of c implies that at most c orders can be shared and served by a courier at a time. In the following, we consider the case $c = 2$, i.e., search 2-order sharing where two orders can be served in a sharing manner.

Definition 2. 2-order sharing $\langle r_j, r_k \rangle$. *Given two orders r_j and r_k with fixed time window t_j and t_k, and assuming that $t_j \leq t_k$ without loss of generality, it is beneficial for serving the two orders in a sharing manner if these two orders satisfy the following two properties: (1) it is not feasible to serve r_j and r_k consecutively, i.e., $t_j + \tau_{jk} > t_k$, and (2) it is feasible to execute one of the following two sharing delivery plans $\langle v_j^p, v_k^p, v_j^d, v_k^d \rangle$ and $\langle v_k^p, v_j^p, v_j^d, v_k^d \rangle$. In particular, property (2) requires that a courier can start from certain period t_s and certain region v_s, and one of the following inequations should satisfy:*

$$\begin{cases} t_s + d_{v_s v_j^p} + d_{v_j^p v_k^p} + d_{v_k^p v_j^d} \leq t_j, \\ t_s + d_{v_s v_j^p} + d_{v_j^p v_k^p} + d_{v_k^p v_j^d} + d_{v_j^d v_k^d} \leq t_k. \end{cases} \tag{6}$$

$$\begin{cases} t_s + d_{v_s v_k^p} + d_{v_k^p v_j^p} + d_{v_j^p v_j^d} \leq t_j, \\ t_s + d_{v_s v_k^p} + d_{v_k^p v_j^p} + d_{v_j^p v_j^d} + d_{v_j^d v_k^d} \leq t_k. \end{cases} \tag{7}$$

Equation (6) corresponds to the sharing plan $\langle v_j^p, v_k^p, v_j^d, v_k^d \rangle$, and Equation (7) corresponds to the sharing plan $\langle v_k^p, v_j^p, v_j^d, v_k^d \rangle$. The motivation of the property (1) is shown as follows. In DSS-Fix, for two orders r_j and r_k that can be served consecutively, it does not take any advantage of sharing them by picking them up (resp. dropping off) consecutively. This is because, with the fixed drop-off period t_j and t_k, the sharing plan cannot finish r_j and r_k earlier, but might need the earlier pick up period. For example, for the sharing plan $\langle v_j^p, v_k^p, v_j^d, v_k^d \rangle$, after picking up r_j at v_j^p, the courier needs to travel to the pick-up region v_k^p of r_k and the drop-off region v_j^d of r_j. However, a courier can travel directly from v_j^p to v_j^d in the routing plan $\langle v_j^p, v_j^d, v_k^p, v_k^d \rangle$ without sharing, which needs fewer periods to finish r_j than that of the sharing plan. For values of $c > 2$, there are up to c orders that can be shared and served at a time. Because of the computational reasons, the capacity parameter c has a substantial impact on the feasibility of solving the DSS-Fix problem. Here, we mainly consider the case $c = 2$. However, as we show in the experiments, even the minimum possible number of order sharing (i.e, 2-order sharing) can provide immense benefits to DSS-Fix variants.

Iterative DSS-Fix Algorithm. For any 2-order sharing $\langle r_j, r_k \rangle$, we package them as a whole and construct two virtual orders r_{jk}^s and r_{kj}^s for the shared plans (if any) $\langle v_j^p, v_k^p, v_j^d, v_k^d \rangle$ and $\langle v_k^p, v_j^p, v_j^d, v_k^d \rangle$, respectively. Let $\tilde{R}^s$ denote all of the virtual shared orders. Given the sharing orders $\tilde{R}^s$ and the real orders R, we construct the "sharing" CSN $G^s(R \cup \tilde{R}^s, E)$. The directed edge $(r_j, r_k) \in E$ indicates the (real/virtual) orders r_j and r_k can be served consecutively. For the CSN G^s, we invoke Algorithm 2 to return the set of served orders $R^{'}$. Given the optimal solution $R^{'}$ that might include virtual sharing orders, the CSN can be updated and screen out the real orders as follows.

Definition 3. CSN Update Rule. *The solution $R^{'}$ includes virtual sharing and real orders, which might involve the same order, and the CSN G^s should be updated to delete the undesirable orders, such as*

- *For the virtual order r^s_{jk} (resp. r^s_{kj}), if it is not served in the solution $R^{'}$, r^s_{jk} (resp. r^s_{kj}) will be deleted from the CSN G^s.*
- *For the virtual orders r^s_{jk} and r^s_{kj} that belong to the same 2-order sharing $\langle r_j, r_k \rangle$, if they are both served in $R^{'}$, one of the them will be deleted from the CSN G^s.*
- *For the virtual order r^s_{jk} (resp. r^s_{kj}), if it is served in $R^{'}$, and both of the real orders r_j and r_k are also served in $R^{'}$, r^s_{jk} (resp. r_{kj}) will be deleted from the CSN G^s.*
- *For the virtual order r^s_{jk} (resp. r^s_{kj}), if it is served in $R^{'}$, however, only one of the real orders r_j and r_k is served in $R^{'}$, both r_j and r_k will be deleted from the CSN G^s.*

After updating the CSN, Algorithm 2 is invoked again to derive the orders served in the next iteration. The details of the iterative DSS-Fix algorithm are proposed in Algorithm 5. In Step 2, the set of virtual sharing orders $\tilde{R}^s$ is constructed. Based on the constructed $\tilde{R}^s$ and the real orders R, the CSN graph G^s is constructed in Step 3. Steps 4–8 are the iterative process of finding the set of orders $R^{'}_g$ and updating the CSN G^s_g at each iteration g. The iterative process terminates until there is no improvement over the previous iteration on the number of real orders served.

Algorithm 5: Iterative Algorithm for DSS-Fix (Ite-DSS-Fix).

Input : Customer orders R.
Output: Served Orders $R' \subseteq R$.

1 Initialize $g = 0$ and $R^{'}_g = \emptyset$;
2 Search the set of 2-order sharing orders $\tilde{R}^s$;
3 Construct the CSN $G^s_g = (R \cup \tilde{R}^s, E)$;
4 **repeat**
5 $g = g + 1$;
6 Invoke the Algorithm 2 to return the set of orders $R^{'}_g$ served on G^s_{g-1};
7 Update the CSN G^s_g using the CSN update rule;
8 **until** $|R^{'}_{g-1}| = |R^{'}_g|$;
9 Return $R^{'}_g$.

Lemma 2. *Algorithm 5 can always converge within finite iterations.*

The proof is similar to that in Lemma 1 and is omitted here for space limitation.

Lemma 3. *Given the set of customer orders R, the number of orders served by the DSS-Fix algorithm (i.e., Algorithm 5) is not less than that served by the DSR-Fix algorithm (i.e., Algorithm 1).*

Proof. In DSS-Fix, the CSN $G^s = (R \cup \tilde{R}^s, E)$ not only includes the real orders R, but also includes the virtual shared orders $\tilde{R}^s$. Therefore, the number of orders served in the DSR-Fix variant CSN $G = (R, E)$ must not be larger than that in the DSS-Fix variant CSN $G^s = (R \cup \tilde{R}^s, E)$. Moreover, the CSN update rule always deletes the real order r_j that has been served in the virtual shared orders r^s_{jk} (resp. r^s_{kj}), which will not reduce the number of orders served in $G = (R, E)$. In other words, for each served order r_j in $G = (R, E)$ that is not served in $G^s = (R \cup \tilde{R}^s, E)$, there must be another one real order r_k or virtual orders r^s_{jk} (resp. r^s_{kj}) replacing r_j in $G^s = (R \cup \tilde{R}^s, E)$. Therefore, we have this conclusion. □

Lemma 3 theoretically guarantees the advantage of delivery *sharing* plan over delivery *routing* plan.

4.4. The DSS-Fle Variant Algorithm

On one hand, the DSS-Fle problem, where the order has flexible drop-off time windows, is a general variant of the DSS-Fix problem. On the other hand, the DSS-Fle problem, where orders can be served in a sharing manner, is also a general variant of the DSR-Fle problem. To solve such an NP-hard DSS-Fle problem, we can combine the DSR-Fle variant algorithm (i.e., Algorithm 4) and the DSS-Fix variant algorithm (i.e., Algorithm 5).

The main idea of the DSS-Fle variant algorithm is that we first employ Ite-DSR-Fle algorithm (i.e., Algorithm 4) to return "fixed" orders whose drop-off periods are determined and fixed. Given the "fixed" orders, we then employ the Ite-DSS-Fix algorithm (i.e., Algorithm 5) to search the order-sharing and generate the sharing delivery plan. The details of the DSS-Fle algorithm are shown in Algorithm 6. In Step 2, the Ite-DSR-Fle algorithm is employed to determine the fixed drop-off period of each order (For the order whose drop-off period cannot determined, a greedy heuristic of choosing its latest drop-off period is employed). Let R^{fix} denote the set of "fix" orders. In Step 3, the Ite-DSS-Fix algorithm is employed to optimize the set of fixed orders R^{fix}.

Algorithm 6: The DSS-Fle Variant Algorithm (**DSS-Fle**).

Input : Customer orders R.
Output: Served orders $R' \subseteq R$.
1 Initialize $R^{fix} = \emptyset$;
2 Invoke the Ite-DSR-Fle algorithm (i.e., Algorithm 4) to generate "fixed" orders R^{fix} with fixed drop-off periods;
3 Invoke the Ite-DSS-Fix(R^{fix}) algorithm (i.e, Algorithm 5) to return the served orders $R^{'}$;
4 Return $R^{'}$.

Lemma 4. *Algorithm 6 can always converge within finite iterations.*

This proof can be derived from the convergence results of Algorithms 4 and 5.

5. Experimental Evaluation

We generate synthetic datasets to validate the performance of proposed algorithms. All computations are performed on a 64-bit PC with 16 GB RAM and a Dual core 3.5 GHz processor. All records are averaged over 40 instances, and each record is statistically significant at a 95% confidence level. Existing public delivery datasets such as the New York City taxi-trip dataset (https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page) only includes the order's pick-up region, drop-off region, and request time, and does not include the flexible drop-off time windows. Thus, we use the synthetic data to validate the proposed algorithms.

Comparison Metrics. We mainly compare the running time and the order service rate (**OSR**) of the algorithms. The OSR is computed as follows: we compare the number of orders served with the total number of orders $|R|$, e.g., the OSR of the algorithm Alg is computed as $\frac{NoS(Alg)}{|R|}$, where $NoS(Alg)$ is the number of orders served by the algorithm Alg.

5.1. Validate the Ite-DSR-Fle Variant Algorithm

In this section, we mainly focus on the DSR variant without considering order sharing and validate the efficiency of the proposed Ite-DSR-Fle approximation algorithm.

Experimental Setup. Each period consists of 5 min and the horizon $T = 36$, which might represent the food-delivery peak-hour at noon ranging from 10:30 a.m. to 1:30 p.m. The city network consists of 100 regions; each region has four neighbors, connected by a small-world network [28]. The traveling period between adjacent neighbors follows $U(1, 4)$, where $U(a, b)$ indicates the uniform distribution between a and b. The time distance between two regions can be computed by the shortest path. There are 200 couriers starting routing with uniformly distributed initial regions. Customer orders are randomly generated as a Poisson process $P(\lambda)$ with a fixed arrival rate λ. The origin and destination of each order is uniformly drawn across the regions. By setting the arrival rate of the Poisson process so that the expectation of the total number of orders is 200, 500, 1000, respectively, these three levels of customer order demand correspond to the low, medium, and high demands. We give all customers a constant time window around their preferred drop-off period (that follows $U(1, T)$), from one to five periods.

Comparison Methods. We compare the proposed approximation **Ite-DSR-Fle** with the following two benchmarks.

- The integer-programming-based optimal solution (IPopt): We use the CPLEX (version 12.6) to solve this IPopt (i.e., Equations (1)–(5)) to return the optimal solution.
- Greedy: Each customer order r_j is assigned with its latest drop-off period $t_j^* = t_j^l$ as its fixed drop-off period, based on which the CSN-DSR-Fix (i.e., Algorithm 1) is employed to return the solution.

Experiment Results. Table 2 shows the OSR and running time of these algorithms with varying order demands and window lengths. From Table 2, we can observe that (1) the proposed Ite-DSR-Fle can perform close to the optimal IPopt with respect to maximizing OSR. For example, in the worst scenario with high order demand and the five periods flexible time window, Ite-DSR-Fle finishes $\sim$5% less OSRs than that of the optimal IPopt. This result is consistent with the theoretical approximation of the Ite-DSR-Fle algorithm. (2) Ite-DSR-Fle achieves nearly $\sim$6% more OSRs than that of the greedy heuristic in average. The potential reason is that the orders' drop-off periods should be coordinated rather than chosen by their latest period independently. Considering that there are thousands of customer orders, such an improvement is desirable. (3) Since IPopt needs to solve the IP Equations (1)–(5), the running time prevents it from scaling to large instances. For example, in the scenario with high demand, it will take a couple of minutes (e.g., 80.4 s) to return the routing plan. However, the Ite-DSR-Fle can generate the routing plan within about 1.4 s. (4) The more flexible time windows, the more OSR will be achieved by the Ite-DSR-Fle. For example, in the scenario of high order demand, the OSR of Ite-DSR-Fle(#1) is 0.435, while the OSR of Ite-DSR-Fle(#5) increases to 0.484, where #1 and #5 indicate that the lengths of time windows are 1 and 5, respectively. This can be explained by the fact that, the longer the time window, the more feasible routing plans can be generated, and the more orders can be finished.

In summary, in the DSR-Fle variant, the proposed Ite-DSR-Fle can achieve nearly the optimal OSR, while the running time is significantly reduced over the optimal solution.

Table 2. OSR and running time on small-scale instances. To make the table clear, we omit the statistical errors, and each cell is statistically significant at a 95% confidence level.

Time Window	Demand	Order Service Rate (OSR)			Runtime (Second)		
		IPopt	Ite-DSR-Fle	Greedy	IPopt	Ite-DSR-Fle	Greedy
One period	Low	0.528	0.502	0.459	26.4	0.21	0.05
	Medium	0.471	0.451	0.405	44.4	0.34	0.07
	High	0.451	0.435	0.385	60.1	0.44	0.09
Three periods	Low	0.530	0.506	0.473	27.8	0.30	0.06
	Medium	0.514	0.487	0.437	43.8	0.55	0.07
	High	0.481	0.454	0.389	63.4	0.64	0.07
Five periods	Low	0.576	0.543	0.488	33.0	0.45	0.07
	Medium	0.534	0.500	0.442	54.0	0.90	0.07
	High	0.529	0.484	0.433	80.4	1.07	0.08

5.2. Validate the DSS-Fle Variant Algorithm

In this section, we will validate the DSS-Fle variant algorithm, such as the advantage of order sharing over DSR variant algorithms, the advantage of flexible time selection mechanism over traditional fixed time selection heuristics, and the scalability on city-scale applications.

Experimental Setup. We consider that the city-scale network consists of 1000 regions, connected by a small-world like network, and each node has on average six neighbors. We consider that there are 1000 couriers with uniformly distributed depot regions. Each customer order has a fixed time window of three periods around their preferred drop-off period. The other settings are similar to that in Section 5.1.

5.2.1. Validate the Advantage of Order Sharing and Scalability

Comparison Methods and Metrics. We compare the proposed **DSS-Fle**, **Ite-DSS-Fix**, **Ite-DSR-Fle** and **DSR-Fix** algorithms on running time and OSR.

Experimental Results. Table 3 shows the OSR and running time of these algorithms with varying customer order demands. From Table 3, we can observe that (1) the DSS-Fle can always produce the highest OSR, which is followed by Ite-DSR-Fle, Ite-DSS-Fix, and DSR-Fix. This result is consistent with the theoretical analysis that DSS-Fle $\succ$ Ite-DSS-Fix, and Ite-DSR-Fix $\succ$ DSR-Fix. This can be explained by the fact that taking the flexible time windows and order sharing (though 2-order sharing) into consideration is beneficial for improving OSRs. The experimental order-sharing advantage is consistent with the theoretical results of Lemma 3. (2) For the large-scale instances with 10,000 orders, although DSS-Fle will take about 2 min to return the delivery plans, considering its benefit of improving $\sim$200 orders over DSR-Fix, DSS-Fle is still a good option for food ordering apps. In food-ordering apps, customers are more concerned with whether their food can be dropped off in time or not, and the response time is tolerable within several minutes [20]. (3) With the constant couriers (e.g., 1000 in this experiment), the more customer order demands, the more number orders that can be served. For example, in the case of 2000 orders, 1140 (= 2000 $\times$ 0.572) orders are served on average, while, in the case of 10,000 orders, nearly 1720 (= 10,000 $\times$ 0.172) orders will be served. This can be explained by the fact that more customers will generate more feasible delivery plans, thereby improving the number of orders served.

Table 3. Order Service Rate (OSR) and running time on large-scale instances. To make the table clear, we omit the statistical errors, and each cell is statistically significant at a 95% confidence level.

Demand	Order Service Rate (OSR)				Runtime (Second)			
	DSS-Fle	Ite-DSS-Fix	Ite-DSR-Fle	DSR-Fix	DSS-Fle	Ite-DSS-Fix	Ite-DSR-Fle	DSR-Fix
2000	**0.572**	0.554	0.563	0.544	3.20	0.38	3.02	0.36
4000	**0.326**	0.307	0.313	0.302	18.4	1.83	16.2	1.33
6000	**0.239**	0.222	0.227	0.219	44.6	4.01	43.9	3.67
8000	**0.196**	0.184	0.189	0.180	95.4	9.48	75.1	6.78
10,000	**0.172**	0.159	0.163	0.155	134.6	19.73	124.5	12.9

5.2.2. Validate the Advantage over Existing Heuristics

Comparison Methods and Metrics. We compare the proposed **DSS-Fle** with the following two existing heuristics:

- Greedy: Each customer order r_j is assigned with its latest drop-off period $t_j^* = t_j^l$ as its fixed drop-off period, based on which the CSN-DSR-Fix (i.e., Algorithm 1) is employed to return the solution.
- Insertion-based Heuristics (**Insert-Heu**): The Greedy algorithm is first employed to derive a solution of n delivery sharing plans $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$. For any order $r_j \in R$ that is not served in $\mathcal{P}$, a re-optimization of inserting r_j into a sharing plan $P_i \in \mathcal{P}$ is elaborated. Given the delivery plan $P_i = \langle v_1^p, v_1^d, v_h^p, v_h^d \rangle$ of a temporally-ordered route of pickup and drop-off regions of h orders, the insertion heuristic attempts to insert r_j pick-up region v_j^p and drop-off region v_j^p into these $2h$ regions. A feasible insertion of r_j into P_i must satisfy (1) without violating the order service in the rest of the plan P_i, and (2) the number of on-board orders is smaller than the courier's capacity c ($c = 2$ in this experiment). There are $O(l^2)$ possible ways of insertion for each order and each plan, where l is the number of orders. Given these n delivery plans and l orders in total, there are $O(nl^3)$ insertion computations.

Experimental Results. Figure 3 shows the OSRs of these algorithms with varying customer order demands. From Figure 3, we can observe that DSS-Fle can always serve the most orders, which is followed by Insert-Heu and Greedy. This result can validate (1) the advantage of flexible drop-off time selection mechanism (i.e., Algorithm 3) over a fixed drop-off time selection mechanism (i.e., Insert-Heu and Greedy), and (2) the advantage of insertion mechanism (i.e., Insert-Heu) over the one-short Greedy solution on improving OSR. Moreover, inspired by the advantage of Insert-Heu, it is interesting for the future work to combine the proposed DSR-Fle with the insertion mechanism for the online delivery services' applications.

In summary, the proposed DSS-Fle can maximize system throughput in the tolerable time in the city-scale order delivery applications with thousands of couriers and customers. Moreover, the more computation time available, the larger system throughput that will be achieved.

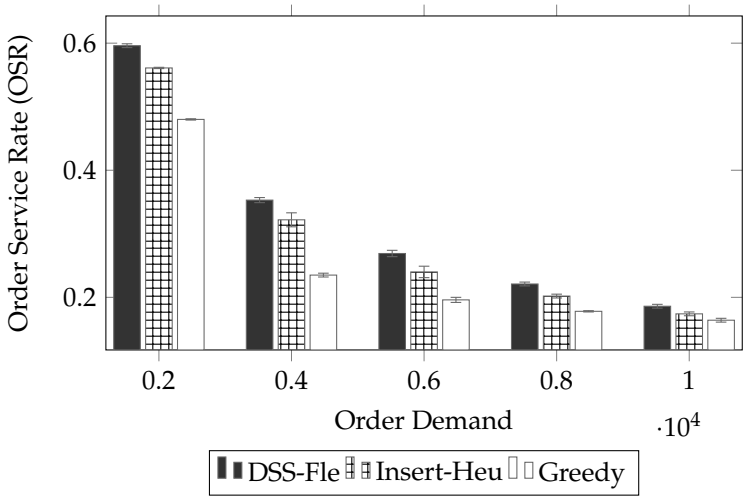


Figure 3. Order service rates of algorithms.

6. Conclusions

This paper studies the practical DSS-Fle problem, where customer orders have flexible drop-off time windows and can be served in a sharing manner. To address such a NP-hard problem, we first study the special DSR-Fix variant, which can be addressed by the CSN and Hopcroft–Karp maximum matching algorithm. By extending the DSR-Fix variant algorithm, we further propose an approximation algorithm (i.e., Ite-DSR-Fle) for the DSR-Fle problem and a heuristic algorithm (i.e., Ite-DSS-Fix) for the DSS-Fix problem. The Ite-DSR-Fle algorithm splits the flexible time windows into multiple fixed time windows, and the DSR-Fix variant algorithm can then be employed to serve orders at the desirable drop-off period. The Ite-DSS-Fix algorithm searches and inserts the sharing orders, and the DSR-Fix variant algorithm then can be employed to generate the desirable sharing plan. Finally, using Ite-DSR-Fle to generate the "fixed" orders and using Ite-DSS-Fix to search order sharing together yield the polynomial time DSS-Fle algorithm for order service rate optimization. Simulation results show that the proposed DSS-Fle algorithm is efficient both on improving order service rate and applying to city-scale scenarios with thousands of regions and customer orders.

Author Contributions: W.W. proposed the problem; W.W. and H.T. designed the algorithms and performed the simulation; W.W. and Y.J. contributed in analyzing the performance and writing this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the National Key Research and Development Project of China (2019YFB1405000), the National Natural Science Foundation of China (61932007, 61806053 and 61807008), the Natural Science Foundation of Jiangsu Province of China (BK20171363, BK20180356, and BK20180369), and the "Zhishan Young Scholar" Program of Southeast University.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Solomon, M.M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Oper. Res.* **1987**, *35*, 254–265. [CrossRef]
2. Bent, R.; Hentenryck, P.V. Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, 6–12 January 2007; pp. 1816–1821.
3. Kamar, E.; Horvitz, E. Collaboration and Shared Plans in the Open World: Studies of Ridesharing. In Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, CA, USA, 11–17 July 2009; p. 187.
4. Ota, M.; Vo, H.; Silva, C.; Freire, J. STaRS: Simulating Taxi Ride Sharing at Scale. *IEEE Trans. Big Data* **2017**, *3*, 349–361. [CrossRef]
5. Jiang, S.; Chen, L.; Mislove, A.; Wilson, C. On Ridesharing Competition and Accessibility: Evidence from Uber, Lyft, and Taxi. In Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, 23–27 April 2018; pp. 863–872.
6. Vazifeh, M.M.; Santi, P.; Resta, G.; Strogatz, S.H.; Ratti, C. Addressing the minimum fleet problem in on-demand urban mobility. *Nature* **2018**, *557*, 534–538. [CrossRef] [PubMed]
7. Bertsimas, D.; Jaillet, P.; Martin, S. Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications. *Oper. Res.* **2019**, *67*, 143–162. [CrossRef]
8. Hosni, H.; Naoum-Sawaya, J.; Artail, H. The shared-taxi problem: Formulation and solution methods. *Transp. Res. Part B Methodol.* **2014**, *70*, 303–318. [CrossRef]
9. Bei, X.; Zhang, S. Algorithms for Trip-Vehicle Assignment in Ride-Sharing. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, LA, USA, 2–7 February 2018; pp. 3–9.
10. Ma, S.; Zheng, Y.; Wolfson, O. Real-Time City-Scale Taxi Ridesharing. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 1782–1795. [CrossRef]
11. Potvin, J.Y.; Rousseau, J.M. An Exchange Heuristic for Routeing Problems with Time Windows. *J. Oper. Res. Soc.* **1995**, *46*, 1433–1446. [CrossRef]
12. Taillard, É.D.; Badeau, P.; Gendreau, M.; Guertin, F.; Potvin, J.Y. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transp. Sci.* **1997**, *31*, 170–186. [CrossRef]
13. Bent, R.; Van Hentenryck, P. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Comput. Oper. Res.* **2006**, *33*, 875–893. [CrossRef]
14. Schneider, M.; Stenger, A.; Goeke, D. The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations. *Transp. Sci.* **2014**, *48*, 500–520. [CrossRef]
15. Desaulniers, G.; Errico, F.; Irnich, S.; Schneider, M. Exact Algorithms for Electric Vehicle-Routing Problems with Time Windows. *Oper. Res.* **2016**, *64*, 1388–1405. [CrossRef]
16. Xinyu, W.; Shuai, S.; Jiafu, T. Iterative Local-Search Heuristic for Weighted Vehicle Routing Problem. *IEEE Trans. Intell. Transp. Syst.* **2020**, in press.
17. Berbeglia, G.; Cordeau, J.F.; Laporte, G. Dynamic pickup and delivery problems. *Eur. J. Oper. Res.* **2010**, *202*, 8–15. [CrossRef]
18. Santi, P.; Resta, G.; Szell, M.; Sobolevsky, S.; Strogatz, S.H.; Ratti, C. Quantifying the benefits of vehicle pooling with shareability networks. *Proc. Natl. Acad. Sci. USA* **2014**, *111*, 13290–13294. [CrossRef] [PubMed]
19. Hopcroft, J.E.; Karp, R.M. An n 5/2 Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* **1973**, *2*, 225–231. [CrossRef]
20. Das, A.; Gollapudi, S.; Kim, A.; Panigrahi, D.; Swamy, C. Minimizing Latency in Online Ride and Delivery Services. In Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, 23–27 April 2018; pp. 379–388.
21. Xiang, Z.; Chu, C.; Chen, H. A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. *Eur. J. Oper. Res.* **2006**, *174*, 1117–1139. [CrossRef]

22. Hasan, M.H.; Hentenryck, P.V.; Budak, C.; Chen, J.; Chaudhry, C. Community-Based Trip Sharing for Urban Commuting. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, LA, USA, 2–7 February 2018; pp. 6589–6597.
23. Santos, D.O.; Xavier, E.C. Dynamic Taxi and Ridesharing: A Framework and Heuristics for the Optimization Problem. In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI'13, Beijing, China, 3–9 August 2013; pp. 2885–2891.
24. Lowalekar, M.; Varakantham, P.; Jaillet, P. ZAC: A Zone Path Construction Approach for Effective Real-Time Ridesharing. In Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, 11–15 July 2019; pp. 528–538.
25. Zhu, M.; Liu, X.; Wang, X. An Online Ride-Sharing Path-Planning Strategy for Public Vehicle Systems. *IEEE Trans. Intell. Transp. Syst.* **2019**, *20*, 616–627. [CrossRef]
26. Boesch, F.T.; Gimpel, J.F. Covering Points of a Digraph with Point-Disjoint Paths and Its Application to Code Optimization. *J. ACM* **1977**, *24*, 192–198. [CrossRef]
27. Garey, M.R.; Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness; In *Mathematical Sciences*, W.H.Freeman: New York, NY, USA, 1990.
28. Watts, D.J.; Strogatz, S.H. Collective dynamics of 'small-world' networks. *Nature* **1998**, *393*, 440–442. [CrossRef] [PubMed]