Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers

Afonso Sampaio1 | Martin Savelsbergh2 | Lucas P. Veelenturf1 | Tom Van Woensel1

1 School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, Netherlands
2 H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA

Correspondence
Afonso Sampaio, Eindhoven University of Technology, P.O. Box 513, Atlas 4.402, 5600 MB Eindhoven, Netherlands.
Email: a.h.sampaio.oliveira@tue.nl

Funding information
This research was supported by the DATAS for Cities, Grant/Award Number: 438-15-507; Physical Internet Reisbeurzenprogramma, Grant/Award Number: 439-17-703

Abstract
Rapid urban growth, the increasing importance of e-commerce and high consumer service expectations have given rise to new and innovative models for freight delivery within urban environments. Crowdsourced solutions—where drivers are not employed by a carrier but occasionally offer their services through online platforms and are contracted as required by carriers—are receiving growing attention from industry. We consider a crowdsourced system where drivers express their availability to perform delivery tasks for a given period of time and the platform communicates a schedule with requests to serve. We investigate the potential benefits of introducing transfers to support driver activities. At transfer locations, drivers can drop off packages for pick up by other drivers at a later time. We frame the problem as a multidepot pickup and delivery problem with time windows and transfers, and propose an adaptive large neighborhood search algorithm that effectively identifies beneficial transfer opportunities and synchronizes driver operations. Computational experiments indicate that introducing transfer options can significantly reduce system-wide travel distance as well as the number of drivers required to serve a given set of requests, especially when drivers have short availability and requests have high service requirements.

KEYWORDS
crowdshipping, large neighborhood search, pickup and delivery, transfers

1 INTRODUCTION

Delivering goods in urban areas is one of the most challenging logistics activities. And given the continuing urbanization and the increasing share of e-commerce in retail sales, it will only become more challenging in the future. Therefore, creative ideas and innovative concepts are needed and have to be explored. Among the most promising, so far, is crowdshipping, where transportation capacity is provided by individuals willing to provide their time and their vehicle for a (short) period of time. The use of crowdshipping to improve business-to-consumer delivery and to support high consumer service levels is being explored by various companies. As an example, e-tailer Zalando relies on Trunkrs to offer same-day delivery for its customers in certain cities in Europe. Trunkrs uses crowdsourced delivery, but also established courier services to provide the reliability demanded by its customers (the e-tailers). For an overview of crowd-based applications in transportation and logistics, see Buldeo Rai et al. [6] and Sampaio et al. [26].

The primary challenge associated with the use of crowd-sourced transportation capacity is that the capacity is provided by individuals who are not under contract and incentive schemes have to be used to secure capacity when and where it is needed. One option is to dynamically adjust compensation based on the required capacity, that is, increase compensation at times when...
and at places where more capacity is needed. Another option, the one we assume is used in our research, is to ask individuals to commit to work for a specific period of time, a block, in return for a minimum pay assurance. Such a scheme is employed, for example, by Amazon Flex and Grubhub (a meal delivery platform).

Because blocks tend to be relatively short, as this provides flexibility both for the company and for the individuals, we consider a system in which goods to be delivered can be transferred, that is, be taken from their origin to their destination by more than one driver. This allows a company to make use of available crowd-sourced capacity to handle short-distance as well as long-distance transportation tasks.

To model the proposed crowd-sourced delivery system, we consider a pickup and delivery problem with time windows and transfers (PDPTW-T). At transfer locations, drivers can drop off packages for pick up by other drivers at a later time. The number and location of transfer points are strategic decisions and not considered in our study. A transfer location can be situated at accessible locations—facilitating the operations between drivers—such as gas stations, stores and supermarkets. It can also be an automated facility, such as a locker station, situated in one of those locations. Characterizing delivery settings that might benefit from the introduction of transfers has yet to be studied (to the best of our knowledge, this is only investigated by Mitrović and Laporte [20]).

When the transfer of requests is not an option, the problem is a pickup and delivery problem with time windows (PDPTW) and defined as follows. A fleet of vehicles is available to satisfy a set of transportation requests, each defined by a load to be transported (goods or people) by a single vehicle from a pickup location (origin) to a delivery location (destination), within given time windows for visiting each location. Multiple requests can be served by the same vehicle, as long as the vehicle capacity is never exceeded. When all requests are known in advance, the problem is referred to as the static PDPTW; when requests arrive during the execution of planned routes and routes have to be updated to accommodate new requests, the problem is referred to as a dynamic PDPTW. The objective is to determine a set of vehicle routes serving all requests, where the function that drives the costs is chosen according to the application at hand (e.g., distance, duration, number of vehicles, or pollution; see Savelsbergh and Sol [27]). The reader is referred to Berbeglia et al. [5] for a survey on dynamic pickup and delivery problems.

As mentioned above, transfer points are locations in the network where requests can be transferred between vehicles and temporarily stored. Hence, more than one vehicle can be used to serve a single request, for example, a request may be picked up at its origin by one vehicle, then dropped at a transfer point where another vehicle (potentially with other characteristics) picks it up and delivers it at its destination. Note that direct transportation from the pick up location to the delivery location by the same vehicle is still possible. Transfer points allow, among others, service by a mixed fleet of vehicles (electric) truck, van, or bike), but also allow integration of freight and passenger transportation. As any pickup and delivery problem can be seen as a special case of a pickup and delivery problem with transfers, pickup and delivery problems with transfers are NP-hard. A critical challenge when solving pickup and delivery problems with transfers is synchronization. Whereas allowing goods to be temporarily stored (or people to wait) at an intermediate location provides more flexibility and, thus, may allow more effective use of resources, these resources need to be carefully synchronized in order to obtain a feasible solution.

Our contributions can be summarized as follows. We analyze the potential benefits of transfers in pickup and delivery operations in urban areas, focusing specifically on settings in which drivers operate short shifts (as is likely to happen in crowd-shipping settings models). In such settings, the flexibility provided by transfers may allow serving long-distance requests that would otherwise be impossible. We propose a heuristic for the solution of instances of the multiple-depot PDPTW-T. The heuristic produces high-quality solutions in a reasonable amount of time. We compare the performance of the heuristic to that of a state-of-the art heuristic for the (multiple-depot) PDPTW. The main limitation of our approach, at this point in time, is that it assumes that all information is available at the start of the planning horizon. In the future, we plan to extend our technology to a dynamic setting, where information arrives over time during the planning horizon. Nevertheless, we believe that our current analysis already provides relevant and informative insights into the potential benefits of transfers for transportation systems relying, in part, on crowd-sourced capacity.

The remainder of the paper is organized as follows. The relevant literature and related problems are discussed in Section 2. The notation used throughout the text is introduced in Section 3, as well as a formal problem definition of the problem and a mathematical programming formulation. The heuristic, an adaptive large neighborhood search (ALNS) heuristic, is introduced in Section 4, and the results of an extensive computational study are presented in Section 5. Finally, we present conclusions and directions for further research in Section 6.

2 | LITERATURE REVIEW

The literature considering the use of transfer points in pickup and delivery problems can be divided in two groups: (a) pickup and delivery problems with transshipment (PDP-T) and (b) pickup and delivery with cross-docking (PDP-CD). Whereas both, transfers and cross-docks, act similarly in providing a consolidation mechanism in which short term storage for a limited
amount of requests allows vehicles to potentially improve their loading plans, a few differences are worth mentioning. First, in a cross-docking system transportation requests are executed by a set of separate pickup and delivery routes. Inbound vehicles collect the items, bring them to the CD where they are consolidated, and then a set of outbound vehicles delivers them to their final destinations [17]. Transfer locations, on the other hand, can be seen as a possible transfer opportunity that is, a request can be carried out either through transfers and, in this case, one vehicle collects the load and another one delivers it to its final destination, or the request can be served directly by the same vehicle. Second, the CD is the start and end location for each route in the plan, whereas a transfer can be any location where vehicles can exchange loads throughout their routes. Third, in some CD applications, consolidation activities can only be executed when all vehicles are in the CD. In the PDP-T, this is not required as long as synchronization requirements are met.

The static (i.e., all transportation requests are known in advance of the optimization) PDP variants have given rise to a substantial amount of research, for an overview, the reader is referred to Berbeglia et al. [4]. Only recently the possibility of allowing transfers of goods (or people) is addressed in the literature. Mitrović and Laporte [20] propose a two-phase heuristic to solve the PDP-T consisting of a construction phase, in which a start solution is obtained, and an improvement phase defined by iteratively removing and inserting requests in a candidate solution. Nakao and Nagamochi [21] consider the PDP-T, without time windows on the requests, and analyze lower bounds for the case where one transfer location is available and vehicles are allowed to visit the transfer at most once. The authors show that \( z_{\text{PDP}} < (6\sqrt{k}) \cdot z_{\text{PDP-T}} \), where \( k \) is either the number of routes in an optimal solution of the PDP-T or the number of requests, \( z_{\text{PDP}} \) and \( z_{\text{PDP-T}} \) are the optimal travel costs for the PDP and the PDP-T, respectively. Cortés et al. [10] propose the first mathematical formulation for the PDP-T and a branch-and-cut approach able to solve small instances. Motivated by an air cargo carrier application, Qu and Bard [23] introduce an insertion heuristic to identify profitable circumstances to exploit the transshipment option. The authors develop a GRASP algorithm and propose a set of randomly generated instances. More recently, Rais et al. [24] propose a new model for the problem, distinguishing between vehicle (routes) and request flows and using multicommodity flows to match these two. Masson et al. [18] propose an ALNS algorithm for the problem, and tackle the dial-a-ride problem (DARP) with transfers in Masson et al. [19]. A similar problem is proposed by Ghilas et al. [13], where requests are allowed to be transferred to/from scheduled lines such as bus, train and metro, operating between two terminals. In a recent survey from Guastaroba et al. [14], pickup and delivery problems with transfers are mentioned as one of the extensions of the PDP with cross-docking.

An increasing number of new start-ups within the shared-economy context, exploring different ways of monetizing under-used assets for example, cars, rooms, is giving rise to new strategic and operational issues that have to be addressed in order to support such systems. It is only recently that crowdsourced delivery systems have received academic attention. Based on the crowdshipping concept envisioned by Walmart, Archetti et al. [1] introduce the vehicle routing problem with occasional drivers (VRPOD). Occasional drivers are in-store customers willing to fulfill the delivery for an online customer on their journey (performing a small detour, if necessary) after leaving the store. The authors highlight the challenges associated with designing appropriate compensation mechanisms and the importance of employing company drivers in order to ensure a certain service level. Dayarian and Savelsbergh [12] explore using in-store customers to supplement company drivers to deliver dynamic incoming, on-line customer orders. Dahle et al. [11] build on the VRPOD and introduce the pickup and delivery problem with time windows and occasional drivers (PDPTW-OD). The authors propose compensation schemes for the occasional drivers and show that cost savings of 10%-15% can be achieved even when the company utilizes a suboptimal scheme.

Behrend and Meisel [3] investigate a form of crowdshipping, in which private drivers offer to execute delivery jobs for other people on trips they would make anyway, within item-sharing contexts, where assets for example, tools, leisure equipment, can be temporarily rented. Whereas the prevalent practice on item-sharing platforms is that the transportation of an asset is delegated to the consumer (either the consumer actually performs the task or hire a courier company), the authors show that crowdshipping can increase profits and service levels for an item-sharing platform. Arslan et al. [2] consider a service platform that dynamically creates matches between parcel delivery tasks and private drivers. Since the platform is only able to control the available number of drivers, a regular, dedicated vehicle fleet is also operated by the platform. The authors introduce a new variant of the dynamic pickup and delivery problem, and propose a rolling horizon framework for its solution. Results show that, compared to traditional delivery systems, crowdshipping systems can provide savings of up to 37%.

In another realization of the crowdshipping concept, similar to the approach considered in this work, drivers willing to perform transportation services use an online platform (Uber-Freight, Amazon Flex) and are matched to demand for such services in real-time. The drivers are independent, work for a given period of time and are paid in a hourly basis. To the best of our knowledge, there is no work dealing specifically with such scenarios. Nevertheless, in a similar context, taxi drivers might be willing, at times, to move freight within the city. Li et al. [16] introduce and explore the share-a-ride problem taking into account different requirements to transport people and freight using a taxi network. Taxis are allowed to deliver parcels as long as the service level for the passenger does not deteriorate significantly. A Freight Insertion Problem is proposed to insert parcel collections in a given routing plan for passengers aimed at minimal passenger disruptions (e.g., maximum ride-time for passengers, maximum detours, number of stops). Chen and Pan [7] refers to a “crowd of taxis” to propose, in the same vein as
the solution for reverse flows in Chen et al. [8], using the taxi fleet and a network of 24/7 shops to satisfy last-mile delivery requests.

The use of intermediate locations in crowdsourced delivery systems, as an interface between company drivers and the crowd, is investigated by Kafle et al. [15]. The authors propose a two-tiered delivery system, in which the second tier is crowdsourced to cyclists and pedestrians (the crowd). In the system, a truck carrier posts pickup and delivery requests on a platform and individuals in the crowd bid to carry out a subset of those requests. Parcels can be transferred between a truck and (one or more) individuals at relay locations. The company decides on the winning bids and plans the truck routes that visit the relay points and delivery addresses of requests for which no bids were received or were too expensive. Chen et al. [9] introduce the the multidriver multiparcel matching problem (MDMPMP), in which parcels may be transported by a single or by multiple drivers, using existing planned routes of the drivers. Parcels can be transferred between drivers, allowing for a more flexible matching of drivers and parcels, since drivers do not need to fulfill the complete parcel’s journey and use transfer opportunities to bring the parcel closer to its final destination.

3 | PROBLEM DEFINITION

The PDPTW-T is defined as follows. We are given a set of transportation requests $R = \{(i^+, i^-) | i^+ \in P, i^- \in D\}$, where $P$ and $D$ are the set of pickup and delivery locations, respectively, and where a request $r = (i^+, i^-)$ concerns the pickup of a load at $i^+$ and its delivery at $i^-$ within time window $[E_i^-, L_i^-]$, specifying the earliest time, $E_i^-$, when the load is available at the origin and the latest time, $L_i^-$, when the load must be delivered at the destination. A fleet of vehicles, $V$, is available to serve the requests. Each vehicle $v \in V$ starts and ends its shift at a depot located at $m_v \in M$, with $M$ the set of all depot locations. Each vehicle $v \in V$ has a duty period, or shift, $[E_v, L_v]$; the vehicle can depart its depot at $E_v$ and has to return to its depot by $L_v$. There is a set $\Gamma$ of transfer locations. A request $r_i \in R$ can be served by one vehicle visiting both $i^+$ and $i^-$, or by two vehicles with one vehicle visiting $i^+$ and the other visiting $i^-$. In the latter case, the request is transferred at a location $t \in \Gamma$. Note that we allow only a single transfer. Two vehicles serving a request are not required to be at the transfer location at the same time, that is, transfer locations offer short-term storage (a vehicle can drop off a request and leave, while another vehicle can come and pick up the request at a later time). Multiple vehicles can visit a transfer location at the same time and vehicles can wait at a transfer location. We focus on settings where the number of drivers is large and shift durations are short (relative to the planning horizon)—which is representative of crowdsourcing settings. We also assume, for simplicity of exposition, that the size of loads is small compared to the vehicle capacity, and that the vehicle capacity is never restricting.

3.1 | Mathematical formulation

The PDPTW-T can be modeled on a directed graph $G(W, A)$ with node set $W = P \cup D \cup M \cup \Gamma$ and $A = \{(i, j) | i, j \in W, i \neq j\}$. It takes a vehicle $t_{ij} > 0$ units of time to travel from $i$ to $j$, $(i, j) \in A$, and incurs a cost $c_{ij} > 0$. We assume travel times satisfy the triangle inequality. The PDPTW-T seeks to find a minimum cost set of vehicle routes serving all requests (either by one vehicle or by two vehicles). Rais et al. [24] propose a flow-based mixed integer programming formulation for the PDPTW-T, in which request flows are linked with vehicle flows. The vehicle flow is modeled by binary variables $x_{ij}$, indicating the traversing of arc $(i, j) \in A$ by vehicle $v$. Request flows are modeled by binary variables $y_{ij}$ indicating whether vehicle $v \in V$ carries request $r \in R$ over arc $(i, j) \in A$. Synchronization of transfer operations is achieved through variables $a_{ij}$ and $d_{ij}$, for arrival and departure times of vehicle $v$ at location $i$, respectively, and binary variables $x_{ij}^{r=m}$ indicating whether request $r$ is transferred between vehicles $v$ and $w$ at transfer location $t$. The formulation is given below

$$\min \sum_{v \in V, (i, j) \in A} c_{ij} x_{ij}, \quad (1)$$

subject to:

$$\sum_{(i, j) \in A} x_{ij} \leq 1 \quad \forall v \in V, i = m_v, \quad (2)$$

$$\sum_{(i, j) \in A} x_{ij} = \sum_{(j, i) \in A} x_{ji} \quad \forall v \in V, i = m_v, \quad (3)$$

$$\sum_{(i, j) \in A} x_{ij} - \sum_{(j, i) \in A} x_{ji} = 0 \quad \forall v \in V, \forall i \in W \setminus M, \quad (4)$$

$$\sum_{v \in V, (i, j) \in A} y_{ij}^{r=m} = 1 \quad \forall r \in R, r = (i^+, i^-), \quad (5)$$
The objective (1) is to find a minimum cost set of routes satisfying all requests. Constraints (2) enforce that each vehicle executes at most one route and constraints (3) enforce that a vehicle starts and ends a route at the same depot. Vehicle flow conservation is expressed by constraints (4). Constraints (5) and (6) guarantee that each request is served, that is, it is picked up and delivered, respectively. The flow conservation for requests, at transfer locations, is enforced by constraints (7), and at nontransfer locations by constraints (8). Note that for transfer locations, a request entering the transfer is allowed to leave in a vehicle.

Consistency of arrival and departure times is enforced by big-M constraints (9), which enforce that a request has to travel in a vehicle.

Transfer operations are synchronized by constraints (10) and (11): if request \( r \) is transferred between vehicles \( v \) and \( w \), then vehicle \( v \) can depart from transfer location \( t \) only after the arrival of vehicle \( v \) at transfer location \( t \).

As in the above formulation, we will assume for the remainder that a request is only transferred once, that a vehicle visits a given transfer location only once, and that two vehicles meet and exchange requests only once. A vehicle can visit multiple transfer locations on its route.

### 3.2 Benefits of transfers

To illustrate and assess the potential benefits of using transfers, we analyze a set of stylized instances and compare solutions with and without transfers.

Consider an instance in which the pickup and delivery locations are vertices of a regular \( n \)-sided polygon with side length \( S \), inscribed in a circle of radius \( Y \). A single depot \( (m \in M) \) is located at the center of the circle and also acts as a transfer location \( (t \in \Gamma) \). Let the common shift for vehicles be \([0, 4Y]\) and let the common time window for requests be \([0, 4Y]\) as well, with \( S > Y \).

Next, we locate the pickup and delivery locations of the requests in such a way that the ratio \( \frac{k_1}{k_2} \) can get very large, where \( k_1 \) is the minimum fleet size required to serve all requests when transfers are not allowed and \( k_2 \) is the minimum number of vehicles required when transfers are allowed. Figure 1A-D illustrates instances with 3, 4, 5, and 6 requests, respectively.

In Figure 1A, for example, six requests, \((1^+, 1^-), \ldots, (6^+, 6^-)\) are shown. If fewer than six vehicles are available to serve all requests when transfers are not allowed, then two requests must be picked up and delivered by the same vehicle. However, this
is impossible, because two requests with a common pickup location do not have a common delivery location, which implies that the vehicle needs to visit three locations and the travel time is at least \( Y + S + S + Y \), which exceeds \( 4Y \). Observe that, if more requests are added, a fleet of six vehicles would still be enough to serve all requests, no matter where the pickup and delivery locations for the added requests are located. Thus, in the best solution without transfers, \( k_1 = 6 \) vehicles (routes) are required, each departing from the depot, visiting two vertices of the polygon (one with the pickup and another with the delivery), and returning to the depot.

When transfers are allowed, the best solution requires \( k_2 = 3 \) vehicles, each performing a route departing from the depot, visiting one vertex of the polygon, then the transfer location, followed by a vertex and returning to the depot. All loads are picked up and taken to the transfer location where they are consolidated and then taken to their final destination. It is not possible to use fewer than three vehicles. If two vehicles are used, then one of them has to visit two locations to pickup loads and take them to the transfer location, which would not leave enough time for delivery. Thus, for this setting, we have \( \frac{k_1}{k_2} = 2 \).

Figure 2, illustrates the potential benefits of transfers in a setting with two depots and a single transfer location. In the example, the six requests \((1^+, 1^-), \ldots, (6^+, 6^-)\) can be serviced by vehicles from any depot. Deliveries \(1^-\), \(2^-\) and \(3^-\) are located in a line and spaced by \( S \) time units. The time windows for those locations are such that if a vehicle picks all three requests, the deliveries should be visited in the order \(3^-, 2^-, 1^-\). In particular, latest delivery times are set as \( L_{1^-} = L_{3^-} - S \) and \( L_{2^-} = L_{1^-} - S \) and \( L_{3^-} \) small enough such that a vehicle visiting \(1^-\) or \(2^-\) before visiting \(3^-\) is not able to reach \(3^-\) before \( L_{3^-}\). Moreover, the shift length is such that a vehicle departing from \(1^-\) after visiting \(3^-\) and/or \(2^-\) is not able to return to \( m_1 \) in time. As a consequence, when transfers are not allowed, requests \((1^+, 1^-), (2^+, 2^-), (3^+, 3^-)\) have to be served by a different vehicle (Figure 2A). Observe that for requests \((4^+, 4^-), (5^+, 5^-), (6^+, 6^-)\) we have a similar situation, as the locations are positioned on symmetrical positions, with depot \( m_2 \) replacing \( m_1 \). When transfers are allowed, two vehicles, one from depot \( m_1 \) and one from depot \( m_2 \), can serve all requests (Figure 2B). Note that, after visiting transfer \( t \) and swapping the requests, the deliveries still have to be visited in the order \(3^-, 2^-, 1^-\) (and \(4^-, 5^-, 6^-\)).
Solving instances of the mathematical model presented in Section 3 in a reasonable amount of computation time is only possible for small instances; Rais et al. [24] presents results for instances with 10 and 14 locations (5 and 7 requests, respectively) and where transfers are allowed at every location. This is due, in part, to the symmetry (vehicles are indistinguishable) and the use of big-$M$ constraints, which results in weak linear relaxations. Therefore, we develop an ALNS algorithm which is able to handle large instances in short computation times.

ALNS algorithms have been successfully applied to many hard combinatorial problems, in particular to many variants of the vehicle routing problem (see Pisinger and Ropke [22]). An ALNS algorithm specifically designed for solving instances of the PDPTW is described in Ropke and Pisinger [25]. A successful ALNS algorithm relies on the availability of several fast-to-explore local search neighborhoods for modifying a solution. We focus on neighborhoods that remove requests from the current solution and reinsert them to create a modified solution. At each iteration, one of the neighborhoods is selected, giving priority to neighborhoods that have been successful in earlier iterations. Improving solutions are always accepted, but a diversification mechanism is incorporated which sometimes accepts worse solutions. The basic flow of our ALNS algorithm is given in Algorithm 1.

![Algorithm 1: Overview of the main steps in the proposed solution method (ALNS)]

**Algorithm 1**: Overview of the main steps in the proposed solution method (ALNS)

- **Input**: Set $R = \{(i^+, i^-)|i^+ \in P, i^- \in D\}$ with pickup and delivery requests; Set of transfer locations, $T$
- **Output**: Routing plan $S^*$

1. $I \leftarrow$ Construct an initial solution having a subset of the requests transferred;
2. $S^*, S^c \leftarrow I$;
3. **while** Termination criteria not satisfied **do**
   4. Choose a removal ($O^-$) and an insertion ($O^+$) operators;
   5. $S \leftarrow O^+(O^-(S^*))$;
   6. **if** $z(S) \leq z(S^*)$ **then**
      7. $S^c \leftarrow S$;
      8. **if** $z(S) \leq z(S^*)$ **then**
         9. $S^* \leftarrow S$;
      10. **end**
   11. **else**
      12. $S^c \leftarrow accept(S, S^c)$;
   13. **end**
   14. Update weights used for operators selection;
4. **end**
5. **return** $S^*$;
An initial solution, in which some of the requests are transferred, is obtained as described in Section 4.1. A removal and insertion operator are chosen (line 4) based on weights that reflect past performance (line 14). The removal and insertion operators are applied in sequence (line 5). The acceptance of the resulting solution is controlled by a scheme that is similar to those found in simulated annealing (SA) algorithms, that is, an exponential function with a cooling rate \( \alpha \) (line 12). The initial temperature \( T_0 \) is given by \( \frac{\alpha}{-\ln(0.5)}z_0 \), where \( z_0 \) is the cost of the initial solution and \( 0 < \omega < 1 \), i.e., a solution with cost \( (1 + \omega)z_0 \) has probability 0.5 of being accepted. A higher value \( \omega \) (and consequently \( T_0 \)) allows for more diversification throughout the search, especially during the early stages, but may still be insufficient to avoid getting trapped in local optimum. Therefore, we take a similar approach to Stenger et al. [28] and reset the temperature to \( \frac{\alpha}{-\ln(0.5)}z^* \) after \( v \) iterations in which no improving solution has been found, where \( z^* \) is the cost of the best solution found so far.

### 4.1 Initial solution

A critical aspect of an ALNS algorithm for solving the PDPTW-T concerns the strategies embedded for deciding which requests to transfer and at which locations. Such strategies have to be embedded in the solution improvement framework (i.e., removal/insertion operators that explicitly consider transfers), but starting with an initial solution in which some requests are being transferred might help guide the search—requests are considered one at a time and introducing a (new) transfer is not likely to look attractive as it involves detours to visit a transfer location. Thus, by having (some) transfers in the initial solution, an ALNS algorithm is more likely to make effective use of transfer options.

Any solution, \( s \), is characterized by a subset \( R'_s \subset R \) of requests being transferred. We create an initial solution, \( s_0 \), by forcing a subset of requests to be transferred in the initial solution. Next, we describe how we obtain \( R'_{s_0} \subset R \) (4.1.1) and how we decide on the transfer locations to use (4.1.2) when constructing \( s_0 \) (4.1.3).

#### 4.1.1 Selecting requests to transfer

For a request \( r_i = (i^+, i^-) \in R \), let \( t_i \) be the (direct) travel time between \( i^+ \) and \( i^- \). Selecting the requests that are forced to use a transfer in the initial solution is controlled by a threshold \( \tau' \). We let \( R = \{(i^+, i^-) \in R \mid t_i \geq \tau' \} \) be the candidate set for transferred requests in the initial solution.

Transferring of requests having relatively long travel times might yield cost savings as it may induce “regional” vehicle routes in which requests with pickup and delivery locations in a region are served without a transfer and requests with pickup and delivery locations in different regions are served using a transfer. The threshold \( \tau' \) is defined based on the characteristics of an instance, for example, half of the longest possible travel time.

#### 4.1.2 Selecting a transfer location

Given the set of candidate requests to be transferred in the initial solution, the next decision concerns which transfer location to use. To prevent transfers that require a large detour (compared to traveling directly from pickup to delivery location), transferring \( r_i \in R \) via \( t \in \Gamma \) is considered only when \( t_{i^+, t} + t_{t, i^-} \leq \gamma t_i \), where \( \gamma > 1 \) is a parameter controlling the maximum allowed ratio between the detour \( t_{i^+, t} + t_{t, i^-} \) and the direct travel time \( t_i \). Moreover, transferring request \( r_i \) via \( t \) has to be feasible for at least one combination of depots \( m_a \) and \( m_b \), that is, feasible routes \( m_a - i^+ - t - m_a \) and \( m_b - i^- - t - m_b \), that can be timed and synchronized to induce a feasible transfer at \( t \) have to exist. If there is no feasible transfer location for a request in the candidate set, the request is removed from the candidate set.

Among the feasible transfer locations \( t \in \Gamma \) for request \((i^+, i^-)\) we select the one that minimizes the difference between \( t_{i^+, t} \) and \( t_{t, i^-} \). This strategy favors transfer locations that are located more centrally between the pickup and delivery locations of a request, thus better balancing the detour required to visit the transfer between the two vehicles serving the request. Figure 3 illustrates the possible transfer assignment for a request \((i^+, i^-)\).

#### 4.1.3 Constructing the initial solution

The initial solution, \( s_0 \), is constructed such that all requests in \( R'_{s_0} \) are served by two vehicles and transferred at the location selected using the strategy presented in Section 4.1.2. All other requests \( r_i \in R \setminus R'_{s_0} \) are served by a single vehicle. To construct a feasible solution, all requests in \( R \setminus R'_{s_0} \) are put in the “request bank” and one iteration of the ALNS algorithm for the PDPTW by Ropke and Pisinger [25] is executed to obtain a partial solution for those requests. Then, for each request \( r_i \in R \setminus R'_{s_0} \), we use a “Greedy insertion with transfer” operator of our ALNS framework (described in Section 4.2) to (try and) insert \( r_i \) using the selected transfer location. If unsuccessful, two new routes are created: \( m_a - i^+ - t - m_a \) and \( m_b - i^- - t - m_b \), where depots \( m_a, m_b \in M \) are chosen such that the total travel time of the two new routes is minimized.
4.2 Improvement phase

At the heart of any ALNS algorithm is a set of operators, each modifying the current solution in a specific and limited way, that is, making small modifications to the current solution, and, thus, defining a neighborhood. As is common in many ALNS algorithms for routing problems, our operators either remove or insert sets of locations (either pickup or delivery locations).

For a given solution, let $\rho(i)$ and $\sigma(i)$ denote the direct (on the same route) predecessor and successor nodes, respectively, of node $i$. The detour cost associated with node $i$ is $\delta_i = c_{\rho(i),i} + c_{i,\sigma(i)} - c_{\rho(i),\sigma(i)}$. The detour cost associated with request $r = (i^+, i^-)$ is when $\sigma(i^+) \neq i^-$, and $\delta_r = c_{\rho(i^+),i^+} + c_{i^+,i^-} + c_{i^-,\sigma(i^-)} - c_{\rho(i^+),\sigma(i^-)}$ when $\sigma(i^+) = i^-$. Similarly, if request $r$ is not yet served in the solution, then the cost to insert its pickup, $i^+$, between nodes $u$ and $v = \sigma(u)$ is the detour cost $c_{u,i^+} + c_{i^+,v} - c_{u,v}$, which we also denote as $\delta_{i^+}$. In case of inserting $i^-$, the cost is given by $c_{u,i^-} + c_{i^-,v} - c_{u,v}$, which we denote by $\delta_{i^-}$. The insertion cost of request $r = (i^+, i^-)$, given insertion positions $(u^+, v^+)$ and $(u^-, v^-)$ for the pickup and delivery nodes, respectively, is the detour cost $\delta_r = \delta_{i^+} + \delta_{i^-}$ when $(u^+, v^+) \neq (u^-, v^-)$, and $c_{u,i^+} + c_{i^+,v} - c_{u,v}$ when $(u^+, v^+) = (u^-, v^-) = (u, v)$.

4.2.1 Removal operators

A removal operator deletes a number of requests from the current solution and adds them to a request bank (from which insertion operators select requests to be inserted). Let $O_k^-(s)$ denote the (partial) solution that results after removal operator $O_k^-$ is applied to solution $s$, and let $R_k^-(s)$ denote the set of requests deleted from solution $s$ by operator $O_k^-$. Algorithm 2 illustrates the steps performed by a removal operator.

---

**Algorithm 2: Steps of removal operator $O_k^-$**

**Input**: Feasible solution, $s$

**Output**: Partial solution $O_k^-(s)$ and set of deleted requests $R_k^-(s)$

1. $q \leftarrow$ Number of requests to delete;
2. $O_k^-(s) \leftarrow s$;
3. $R_k^-(s) \leftarrow \emptyset$;
4. for $1, 2, ..., q$ do
   5. Select a request $r \in R \setminus R_k^-(s)$ to be deleted;
   6. $O_k^-(s) \leftarrow O_k^-(s) \setminus r$;
   7. $R_k^-(s) \leftarrow R_k^-(s) \cup r$;
5. end
6. return $O_k^-(s), R_k^-(s)$;

1. Worst request removal ($O_1^-$): The operator deletes $q$ requests in nondecreasing order of their detour cost.
2. Random request removal ($O_2^-$): The operator deletes $q$ requests selected at random.
3. Route removal ($O_3^-$): The operator uses a biased selection procedure to choose a route to delete from the current solution. Only routes that do not visit a transfer are considered. The probability that a route is selected to be deleted is proportional to the ratio of the waiting time in the route and the number of requests served in the route. Thus, the operator favors the deletion of less efficient routes. All requests in the chosen route are deleted. If the number of requests deleted is less than $q$, then another route is selected. The process continues until at least $q$ requests are deleted.
4. Transfer-based request removal ($O^T_d$): This operator considers requests that have been transferred in one or more previous solutions. For each request $r_i = (i^+, i^-)$, let $t^i$ denote the number of times request $r_i$ has been transferred in an incumbent solution encountered during the search. The probability that a request $r_i$ is selected is proportional to $1 - \frac{t^i}{\bar{t}_i}$, where $\bar{t}_i$ denotes the (total) number of incumbent solutions encountered during the search. Note that requests that are seldom transferred in incumbent solutions are more likely to be deleted than requests that are often being transferred in incumbent solutions.

5. Cluster removal ($O^C_\phi$): This operator is an adaptation of an operator used in Masson et al. [18,19] and based on the idea that if the pickups (deliveries) of a set of requests form a cluster, it may be beneficial if they are picked-up (delivered) by the same vehicle and dropped off at (collected from) a transfer location. In our implementation, we randomly pick a “root” request $(i^+, i^-)$ and compute sets $C^+_{i^+}$ and $C^-_{i^-}$, where $C^+_{i^+}$ contains pickup locations within radius $\mu = 60$ units of distance from $i^+$ being serviced by a different vehicle (and the delivery location is not within $\mu$ from $i^+$) and $C^-_{i^-}$ contains delivery locations within radius $\mu$ from $i^-$ being serviced by a different vehicle (and the pickup location is not within $\mu$ from $i^-$). The requests in the larger set are deleted. The parameter $q$ is ignored.

After the deletion of requests, it may happen that a vehicle visits a transfer location, but no transshipment of any request takes place. We allow such unnecessary visits to remain in the current solution for at most $\phi$ iterations. If in $\phi$ consecutive iterations a transfer location $t$ is not used to transfer any request, it is removed from the route(s). Allowing unnecessary visits to transfer locations can be beneficial, because inserting a request involving a transfer location that is not yet in the solution results in additional detour costs and, consequently, a deterioration in solution quality. If the transfer location is already in the solution, even if it is currently not used to transfer requests, the increase is only due to the detour cost incurred for the inserted request itself. Moreover, it also increases the likelihood that the transfer location is already visited by multiple vehicles. A downside of keeping transfer locations with unnecessary visits in the solution is that the time required to visit the transfer location may render an otherwise feasible insertion infeasible. Moreover, more insertions with transfers are evaluated and, as a consequence, execution time is longer. Despite these downsides, our experiments have shown that keeping transfer locations with empty visits provides a good mechanism for exploring transfer opportunities. A postprocessing step at the end of the search removes any unnecessary visits at transfer locations in the best solution found.

### 4.2.2 Insertion operators

The objective of an insertion operator is to reintroduce requests in the set $R^+_i(s)$, that is, the requests deleted by removal operator $O^T_i$. Let $O^+$ ($s$) denote the solution obtained after applying insertion operator $O^T_i$ to a (partial) solution $s$. Algorithm 3 presents the steps performed by an insertion operator that does not consider transfers, and Algorithm 4 presents the steps performed by an insertion operator that does consider transfers.

1. **Greedy insertion without transfer.** Requests are sequentially inserted in the least-cost position and route. If a request cannot feasibly be inserted, it remains in the request bank. The sequence in which requests are inserted defines an operator:
   - $O^+_{i^+}$: Requests are selected in decreasing order of direct travel time from pickup to delivery location.
   - $O^+_{i^-}$: Requests are selected in increasing order of insertion cost. More formally, if $\Delta_{rk}$ represents the least cost to insert request $r$ in route $k$ ($\Delta_{rk} = \infty$ if the insertion is infeasible), then the operator selects request $r'$ to be inserted in route $k'$ as $\arg\min_{r\in R^+_i(s)} \Delta_{rk}$.
   - $O^+_{2i^+}$: Requests are selected in decreasing order of regret, that is, the absolute difference between least insertion cost and the second least insertion cost. More formally, if $\Delta^1_{rk}$ is the least-cost insertion of request $r$ in a route $k$ and $\Delta^2_{rk}$ is the second least-cost insertion of request $r$ in a route $k$ (where $k$ may be the same or a different route), then the operator selects request $r'$ to be inserted in route $k'$ as $\arg\min_{r\in R^+_i(s)} \Delta_{rk}$.
   - $O^+_{2i^-}$: Similar to operator $O^+_{2i^+}$, but the request is selected randomly among the best $\psi$ insertions, where, after initial experiments, $\psi$ is set to 30% of the feasible insertions.

2. **Greedy insertion with transfer.** Requests are sequentially inserted in the least-cost positions and routes. If a request cannot feasibly be inserted, it remains in the request bank. Given a request $r = (i^+, i^-)$, an operator seeks two routes, $k_a$ and $k_b$, and a transfer location, $t$, such that the possible visit times at $t$ for $k_a$ and $k_b$ overlap (ie, there exist a time when both vehicles can be at $t$). If it is possible to insert $i^+$ before $i^-$ in $k_a$ and $i^-$ after $i^+$ in $k_b$ (properly accounting for any change in the arrival time at $t$ of route $k_a$ due to the insertion of $i^+$), the insertion is considered feasible. The sequence in which requests are inserted defines an operator:
   - $O^+_{i^-}$: Requests are selected in decreasing order of direct travel time from pickup to delivery location.
• \( O_6^+ \): Requests are selected in increasing order of insertion cost. More formally, if \( \Delta_{r,k_1,k_2} \) is the cheapest cost to insert request \( r = (i^+, i^-) \), where \( i^+ \) and \( i^- \) are inserted in routes \( k_1 \) and \( k_2 \), respectively (\( \Delta_{r,k_1,k_2} = \infty \) if the insertion is infeasible), then the operator selects a request \( r' \) and routes \( k'_1 \) and \( k'_2 \) by 
\[
(r', k'_1, k'_2) = \arg\min_{r \in R - i} \left( \Delta_{r,k_1,k_2} \right).
\]
Values \( \Delta_{r,k_1,k_2} \) are updated after the selected request has been inserted in its associated routes.

• \( O_7^+ \): Requests are selected in decreasing order of regret. More formally, if \( \Delta^1_{r,k_1,k_2} \) represents the least-cost insertion of request \( r \) using a transfer, and \( \Delta^2_{r,k_1,k_2} \) represents the second least-cost insertion of request \( r \) using a transfer (where the routes in which the pickup and delivery location are inserted may be the same or may differ), then the operator selects request \( r' \) and its associated routes as 
\[
(r', k'_1, k'_2) = \arg\min_{r \in R - i} (\Delta^2_{r,k_1,k_2} - \Delta^1_{r,k_1,k_2}).
\]
Values \( \Delta^1_{r,k_1,k_2} \) and \( \Delta^2_{r,k_1,k_2} \) are updated after the selected request has been inserted in its associated routes.

3. Transfer insert \( O_8^+ \): This operator facilitates the inclusion of transfers in a solution by creating routes with an unnecessary visit at a transfer location. Requests are selected in decreasing order of direct travel time between pickup and delivery location. For a request \( r = (i^+, i^-) \), let \( C_i \) be the circle with radius \( \tau_{i}^2 \), where \( \tau_{i} \) is the distance between \( i^+ \) and \( i^- \), centered at the midpoint of the line segment joining \( i^+ \) and \( i^- \). If there is a transfer location \( t \) within \( C_i \) and a feasible route \( m_{v} - i^+ - t - i^- - m'_{v} \) for vehicle \( v \in V \), then the route is created. If there are multiple transfer locations in \( C_i \), the one minimizing the difference \( |\tau_{i^+} - \tau_{t} - \tau_{i^-}| \) is chosen.

**Algorithm 3:** Steps of insertion operator that does not consider transfers

| Input | Partial solution \( \tilde{s} \) and set of requests \( R \) to be inserted |
|-------|---------------------------------------------------------------|
| Output| A complete solution \( \tilde{s} \), or a partial solution \( \tilde{s} \) plus a set \( \tilde{R} \) of requests that were not inserted |

1. \( L \leftarrow \) requests in \( R \) ordered accordingly to some criterion;
2. \( \tilde{R} \leftarrow \emptyset; \)
3. \( \tilde{s} \leftarrow \tilde{s}; \)
4. \( I \leftarrow \emptyset \) // list of feasible insertions;
5. for \( r = (i^+, i^-) \in L \) do
6. feasible \( \leftarrow \) False;
7. for each route \( v \) in \( \tilde{s} \) do
8. for \( k_1 \in \{0, 1, \ldots, K\} \), positions within route \( v \) do
9. evaluate insertion of \( i^+ \) at position \( k_1 \) in \( v \);
10. if insertion is feasible then
11. for \( k_2 \in \{k_1, \ldots, K\} \) do
12. evaluate insertion of \( i^- \) at position \( k_2 \) in \( v \);
13. if insertion is feasible then
14. feasible \( \leftarrow \) True;
15. \( I \leftarrow I \cup (k_1, k_2, v); \)
16. end
17. end
18. end
19. if feasible = False then
20. \( \tilde{R} \leftarrow \tilde{R} \cup r \)
21. end
22. end
23. return \( \tilde{s}, \tilde{R} \);

It is possible that an insertion operator does not succeed to insert all requests deleted by a removal operator. In this case, the remaining requests are inserted as follows: find a route \( k_a \) with transfer location \( t_a \) maximizing the number of of delivery locations that can be feasibly inserted in \( k_a \) after \( t_a \). Let the set of these delivery locations be denoted by \( D \). Similarly, find a route \( k_b \) with transfer location \( t_b \) maximizing the the number of pickup locations that can be feasibly inserted in \( k_b \) before \( t_b \). Let the set of these pickup locations be denoted by \( P \). If \( |D| \geq |P| \), the delivery locations in \( D \) are inserted in \( k_a \) (if feasible) after \( t_a \). For each inserted delivery location, if it is not possible to insert the corresponding pickup in a route already in the solution,
a new route $m_v - i^+ - t_n - m_v$ is created (assuming a vehicle $v$ is available). If $|D| < |P|$, a similar process is used, but starting with the pickup locations. If, at the end, there are still remaining requests, these are inserted in new routes without transfers.

4.2.3 Operator selection

The adaptive weight mechanism used to control the selection of a removal and insertion operator is implemented by a roulette wheel procedure, as described in Pisinger and Ropke [22]. If the weight of operator $O_i$, $i \in 1, \ldots, \Pi$, is $\pi_i$, expressing the desirability of selecting operator $O_i$, then the probability of selecting $O_i$ is given by $\pi_i / \sum_{j=1}^{\Pi} \pi_j$. The removal and insertion operators are chosen independently, that is, with different roulette wheels. At the start of the search, all operators have equal weights. During the search, the weights are updated every $\kappa$ iterations, similarly to Pisinger and Ropke [22]. As discussed above, in our ALNS algorithm, two types of insertion operators are considered: operators in which transfers are considered, and operators in which transfers are not considered. Since the insertion of a node (request) can be feasible for one type, but not for the other, our ALNS algorithm selects two insertion operators, one of each type. The operator with the largest weight is applied first, and if there are remaining nodes (requests), then the other operator is applied.

The performance of each operator is recorded in its weight based on the solutions obtained after the operator is applied. In particular, the score $\overline{\pi}_i$ for operator $O_i$ starts at 0 at the start of each segment of $\kappa$ iterations, and is incremented by: $\sigma_1$ if an overall best solution is found, $\sigma_2$ if an improving solution is found, and $\sigma_3$ if a worse solution is found and accepted (as part of the annealing scheme). If $a_i$ is the number of times operator $O_i$ was used during the current segment (of $\kappa$ iterations), then the
new weight $\pi_i$ is given by $\pi_i(1 - \rho) + (\pi_i/a_i)\rho$, where $\rho \in [0, 1]$ adjusts the importance given to scores in past segments. Since two insertion operators may be applied in one iteration, their weights are updated independently.

4.2.4 Efficiency
The performance of an insertion-based neighborhood search heuristic relies heavily on its ability to efficiently evaluate the feasibility of an insertion, as a very large number of insertions will be attempted. We provide an appendix with supplementary material where we discuss the information that is kept with the current solution to allow fast (constant time) evaluation of the feasibility of an insertion. Moreover, we show how to update this information when the current solution changes.

5 COMPUTATIONAL EXPERIMENTS
In this section, we present the results of a set of computational experiments conducted to evaluate the performance of the ALNS algorithm and to characterize transportation settings which might benefit from the introduction of transfers. The ALNS algorithm was coded in C++ and the experiments were conducted using an Intel Core i5-2450M machine, running at 2.5 GHz $\times$ 4 with 4 GB of RAM, under Ubuntu 16.04. To investigate the benefits of transfers, we compare the solutions obtained by our ALNS algorithm to the solutions obtained by the ALNS algorithm of Ropke and Pisinger [25], which does not consider transfers. Furthermore, we analyze the impact of different problem characteristics, in particular, the vehicle’s shift length, the number of transfer locations, the geographic distribution of transfer locations, and the geographic distribution of customers.

5.1 Instance generation
To assess the potential benefits of transfers in freight transportation systems employing crowdsourced drivers, we generate sets of instances that capture characteristics of such systems. We consider a squared geographical area of $120 \times 120$ units of distance and assume that one unit of distance can be traveled in one time unit (e.g., 60 km/h). Four depots are located in the area as shown in Figure 4A, requests can be served from any depot, and the number of vehicles in each depot is unlimited. Pickup and delivery locations are drawn uniform randomly in the area, but we consider different scenarios, $C$, for origin-destination proximity: long-distance requests only ($L$), that is, at least 60 units of distance between a pickup and a delivery location, short-distance requests only ($S$), that is, no more than 60 units of distance, but at least 30 units between a pickup and a delivery location, and a third scenario having both long-distance and short-distance requests ($M$).

We generate instances with 50, 75, and 100 requests, each with a 180 units of time (e.g., 3-hour) time window, that is, $E_{tr} = 0$ and $L_{tr} = 180$ for all $r \in R$, for each class $L, M,$ and $S$. If $\gamma = \frac{1}{|R|}\sum_{r \in R} r_i$ indicates the origin-destination proximity of an instance, then for the generated instances in classes $L, M,$ and $S$, we have, on average, $\gamma = 74.67, \gamma = 57.69,$ and $\gamma = 45.82,$ respectively. We consider vehicle shift lengths $H \in \{180, 240, 300\}$ (i.e., drivers operate for three, four, or five hours). The instances are created such that even for the shortest shift length ($H = 180$), all requests can be served (i.e., there is at least one depot such that a vehicle from that depot can feasibly visit the pickup and delivery locations of a request on a route). Moreover,
TABLE 1  Adaptive large neighborhood search parameters and values

| Par. | Description                           | Value | Par. | Description                           | Value |
|------|---------------------------------------|-------|------|---------------------------------------|-------|
| τ    | ALNS iterations                       | 25 000| ψ_1  | Lower limit of removable requests     | 0.1   |
| α    | SA cooling rate                        | 0.98  | ψ_2  | Upper limit of removable requests     | 0.2   |
| ω    | Initial temperature adjustment         | 0.15  | σ_1  | Score for incumbent solution          | 40    |
| κ    | Level length                           | 250   | σ_2  | Score for improving solution          | 10    |
| ν    | Max. nonimproving iterations           | 250   | σ_3  | Score for worse, new solution         | 1     |
| ρ    | Reaction factor                        | 0.2   | φ    | Max. iterations with empty transfers  | 100   |
| µ    | Cover radius used in removal O_t^c     | 60    |      |                                       |       |

Note: Each row corresponds to an instance and each column to the average score over five runs for the relevant parameter value(s). The best parameter value(s) is shown in bold.

TABLE 2  Calibration of adaptive large neighborhood search parameters

| (ψ_1, ψ_2)    | (σ_1, σ_2, σ_3) | (ψ_1, ψ_2)    | (σ_1, σ_2, σ_3) |
|---------------|-----------------|---------------|-----------------|
| (0.1, 0.2)    | (0.2, 0.3)      | (0.3, 0.4)    | (40, 10, 1)     |
| 0.98          | 0.89            | 0.85          | 0.96            |
| 0.87          | 0.82            | 0.76          | 0.88            |
| 0.95          | 0.79            | 0.73          | 0.95            |
| 0.96          | 0.88            | 0.83          | 0.96            |
| 0.93          | 0.84            | 0.77          | 0.90            |
| 0.87          | 0.87            | 0.83          | 0.95            |
| 0.95          | 0.96            | 0.91          | 0.95            |
| 0.90          | 0.94            | 0.74          | 0.72            |
| 0.90          | 0.95            | 0.78          | 0.90            |

Note: Each row corresponds to an instance and each column to the average score over five runs for the relevant parameter value(s). The best parameter value(s) is shown in bold.

we consider two geographies for the transfer locations, MD-4T and MD-5T, shown in Figure 4A,B, respectively. In the former there are four transfer locations, and in the latter there are five transfer locations, with one additional transfer location in the center of the region. The full set of instances can be downloaded at https://doi.org/10.17632/pywzcgzyzrv.2.

We note that Mitrović and Laporte [20] generated instances of the PDPTW-T motivated by an application in last-mile freight transportation, and used these in their computational experiments. Furthermore, Masson et al. [18] generated instances of the PDPTW-T motivated by an application in which transportation requests of disabled people need to be served. As indicated earlier, our research is motivated by an environment in which crowd-sourced drivers, working for short periods of time, transport shipments from origin to destination. As a consequence, the instances available in the public domain are not representative of the operating environment we are exploring, and, thus, the instances are not well-suited for testing our methodology. Nevertheless, for the sake of completeness and as a validation of our methodology, we present results for a subset of the instances proposed by Mitrović and Laporte [20] in the appendix.

5.2 ALNS algorithm parameter tuning

In Table 1, we show the parameters (column Par.) and the values used in our ALNS. In the left side of the table, we present the parameters that are set to similar values used in Ropke and Pisinger [25]. In the right side of the table, we present the parameters we believe have a significant impact on the performance of our proposed ALNS algorithm and for which the values have been carefully calibrated (Table 2).

Parameter tuning was done using a set of six instances, three with 30 and three with 50 requests—one instance for each scenario of origin-destination proximity, L, S and M—, as described in Section 5.1. All parameters to be tuned are given a default value. Then, a single parameter value (or single parameter set values) is varied while the others are kept fixed. For each parameter value considered (of the parameter being tuned), we solve each of the six instances five times. The parameter value yielding the best results is chosen. The process continues until all parameters have been assigned a value. The parameters to be tuned, and their respective default values, are ((ψ_1, ψ_2), (σ_1, σ_2, σ_3), φ) = ((6, 10), (10,10,10), 100), where parameters 0 < ψ_1 < ψ_2 < 1 control the fraction of requests to be removed, σ_1, σ_2, and σ_3 are the reward values for operators and φ is the maximum number of iterations an unused transfer is allowed to remain in the current solution. In the initial solution for an instance, each request is served individually by a single vehicle without a transfer. To assess the quality of different parameter values, we assign a score to each solution obtained: $\frac{z^*}{z}$, where $z$ is the cost of the solution and $z^*$ is the cost of best solution encountered during an experiment. Because we also want the proposed ALNS algorithm to be reasonably efficient, we adjust the score to take run times into account as well: $0.7\frac{z^*}{z} + 0.3\frac{t^*}{t}$, where $t$ is the run time and $t^*$ is the fastest run time encountered during an experiment. Thus, a value close to 1.0 for a parameter value (the average score over the five runs) indicates superior performance.
TABLE 3 Algorithm comparison using stylized instances

| Instance | ALNS Ropke | Proposed ALNS | Optimal |
|----------|------------|---------------|----------|
| Polygon  |            |               |          |
| 4-sided  | 12         | 4336          | 1600     |
|          | 12         | 0.9           | 10       |
|          | 11.3       | 4             | 1600     |
| 5-sided  | 20         | 7104          | 2000     |
|          | 20         | 1.5           | 5        |
|          | 25.5       | 5             | 2000     |
| 6-sided  | 30         | 7244          | 2400     |
|          | 18         | 2.2           | 6        |
|          | 25.5       | 24            | 48.3     |
|          | 48.3       | 6             | 2400     |
| 7-sided  | 42         | 10652         | 2992     |
|          | 28         | 13.6          | 9        |
|          | 32.6       | 36            | 32.6     |
|          | 32.6       | 9             | 2800     |
| 8-sided  | 56         | 14988         | 3200     |
|          | 40         | 17.8          | 8        |
|          | 94.6       | 49            | 3200     |
|          | 94.6       | 8             | 3200     |

First, we tune the interval from which the number of requests to be removed by a removal operator is drawn uniformly, that is, \([\psi_1 | R|, \psi_2 | R|]\). The results show that \((\psi_1, \psi_2) = (0.1, 0.2)\) performs best. Allowing for a larger fraction of requests to be removed may lead to a wider exploration of the solution space, but it also increases time per remove-and-reinsert combination as requests are inserted one by one and execution time primarily depends on the number of requests inserted.

Next, we tune the reward values for the operators, where we note that their relative difference is more important than the values themselves. We consider three sets of parameters: one in which finding new best and better solutions is highly rewarded, that is, \((\sigma_1, \sigma_2, \sigma_3) = (40, 10, 1)\), one in which success in finding new best and better solutions is recognized as a positive and rewarded, but not as much, that is, \((\sigma_1, \sigma_2, \sigma_3) = (4, 2, 1)\) and, finally, one in which we do not actively “encourage” the search for new best and better solutions, but rely on the randomness in the search process to encounter high-quality new best and better solutions, that is, \((\sigma_1, \sigma_2, \sigma_3) = (1, 1, 1)\). With the exception of one instance, the best results are obtained by highly rewarding success in finding new best and better solutions. This indicates that the SA acceptance mechanism, in combination with the restarting procedure, is enough to provide a satisfactory level of diversification in the search, and avoids getting trapped in local optima.

Finally, we calibrate the maximum number of iterations an unnecessary transfer location visit is allowed to remain in the current solution, \(\phi\). We consider two scenarios: one in which an unnecessary transfer location visit is removed immediately, that is, \(\phi = 0\), and one in which an unnecessary transfer location visit is kept much longer, that is, \(\phi = 100\). Even though run times increase when an unnecessary transfer location visit is kept longer, it appears to be worthwhile to do so. This is likely due to the fact that it significantly simplifies the search for beneficial transfer opportunities.

5.3 Stylized instances

We have used the proposed ALNS algorithm to solve stylized instances as shown in Figure 1, with 4-, 5-, 6-, 7-, and 8-sided polygons inscribed in a circle with radius 100 (where instances for 7- and 8-sided polygons are created as expected). We compare the solutions obtained by the proposed ALNS algorithm, where the center of the circle can be used as transfer location, to the solutions obtained by the ALNS algorithm of Ropke and Pisinger [25] (i.e., without any transfers), and to the optimal solutions. The results can be found in Table 3. For each instance, we report the total travel distance (Dist), the number of vehicles used (Veh), the total computational time in seconds (Time) and the number of requests transferred (Trans).

Our proposed ALNS is able to obtain the optimal solution for all but one of the instances. In constructing the polygons, we took distance and time to be equivalent (traveling one unit of distance requires one unit of time) and forced distances and travel times to be integer valued. As a result, for the 7-sided polygon it was not possible to obtain a completely regular shape. This hinders synchronization at the transfer points, and hence a lower quality solution resulted. The higher run times of our proposed ALNS algorithm compared to the one of Ropke and Pisinger [25] are the result of more complex and more time consuming insertion operators, because the need to consider transfers. This is especially true for these stylized, highly symmetrical instances, where proper synchronization of transfers is critical.

5.4 An example of transfer benefits

We first illustrate the potential benefits of transferring requests by analyzing the results of a single instance of class L with 25 requests, shift length 180, and transfer location geography MD-4T. Figure 5 shows two solutions, one in which transfers are not considered (Figure 5A) obtained with the ALNS by Ropke and Pisinger [25], and one in which transfers are considered (Figure 5C) obtained with our proposed ALNS (squares indicate depots, triangles indicate transfer locations, and circles indicate pickup and delivery locations).

In the solution without transfer (Figure 5A) all but six requests are served by a dedicated vehicle (i.e., a vehicle serving a single request), and 22 vehicles are used. In the solution with transfers (Figure 5C) only two requests are served by a dedicated...
vehicle and all other requests are transferred, and only 10 vehicles are used. To facilitate comparing the solutions with and without transfers and to better understand the benefits of transfers, we show, in Figure 5B,D, four rectangles, one for each depot, representing the area containing the routes originating from that depot. Observe that in Figure 5B, corresponding to the solution without transfers, there is a much larger overlap between the rectangles than in Figure 5D, corresponding to the solution with transfers. This illustrates that when transfers are possible, vehicles can stay closer to the depots and can perform pickups and deliveries of multiple requests within the shift; it is as if the transfer locations induce subregions within the region. When transfers are not possible, a vehicle performing a pickup has to also perform the corresponding delivery and consolidating multiple requests within the shift becomes more difficult.

Similar settings that is, transfer location geography MD-4T, 3-hour time window, shift length 180, are used to generate a set of 15 instances with 10 requests, where five instances have long-only, five instances have short-only and five instances have mixed-distance requests. These instances are solved using the IP formulation presented in Section 3 and employing Gurobi 8.01 as the IP solver with a time limit of two hours. For each instance, we compute a solution without transfers, using the ALNS algorithm by Ropke and Pisinger [25], and provide it as an initial solution to the solver. We also compute a solution using our proposed ALNS algorithm and provide it as an initial solution. The results can be found in Table 4, where we report the total distance (Dist), the number of vehicles used (Veh), and the optimality gap after two hours (Gap). Solutions in which one or more requests are transferred are labeled with a superscript $t$. 
The results show that for most instances, Gurobi is unable to improve the initial solution provided in two hours (note the large optimality gaps after two hours of computing). If a solution without transfers is provided, a better solution is found for five instances, where in four of them the improved solution has at least one request that is transferred. The solutions produced by our ALNS algorithm are equal or better (in terms of total distance and number of vehicles used) in 14 out of the 15 instances. When the solutions produced by our ALNS algorithm are provided as initial solution, Gurobi again finds a better solution for five instances, where in three of them the improved solution uses fewer vehicles.

### 5.5 Results for proposed generated instances

A set of 10 instances was generated for each combination of number of requests, origin-destination proximity, and vehicle shift length, as described in Section 5.1. Each instance in a set is solved five times using the ALNS algorithm by Ropke and Pisinger [25] and our proposed ALNS algorithm, and the average cost of the solutions as well as the lowest cost among them are reported. Two initial solutions were considered in the runs with our proposed ALNS algorithm: the solution obtained with the ALNS algorithm by Ropke and Pisinger [25] (in each of the five runs, the same solution was found). Column Best reports results for the best among the five solutions, whereas column Average reports the average over the five solutions. Column s_t indicates the initial solution used: s_t, solution produced by the ALNS algorithm of Ropke and Pisinger [25], or s_t, solution produced by the scheme described in Section 4.1. Column Δ_D(p) reports the reduction in distance, column Δ_V(%) reports the reduction in the number of vehicles used, column Trans reports the number of transferred requests, and column Δ(τ) reports the run time in seconds, excluding the time to obtain the initial solution. Note that scheme s_t requires little time (less than one second) whereas scheme s_t requires running the ALNS algorithm by Ropke and Pisinger [25] which takes a considerable amount of time (on average 60% of the running time of our proposed ALNS algorithm). Detailed results can be found in the supplementary appendix.

We observe that transfers can provide significant benefits, especially when the driver shift length is short (H = 180) and the distance between pickup and delivery locations is long (L)—we see a reduction of almost 50% for both the total distance and the number of vehicles used. When the distance between pickup and delivery location is short (S), the benefits are minor, in the

### Table 4 Experiments for instances with 10 requests

| Inst | Dist | Veh | Dist | Veh | Gap (%) | Dist | Veh | Gap (%) |
|------|------|-----|------|-----|---------|------|-----|---------|
| 1    | 1566 | 10  | 1566 | 10  | 68.3    | 1319 | 9   | 61.4    |
| 2    | 1358 | 9   | 1358 | 9   | 74.5    | 863  | 6   | 58.8    |
| 3    | 1590 | 10  | 1590 | 10  | 77.5    | 1117 | 7   | 65.4    |
| 4    | 1353 | 9   | 1187 | 8   | 67.2    | 1022 | 7   | 64.0    |
| 5    | 1536 | 10  | 1536 | 10  | 76.7    | 1130 | 8   | 64.5    |
| 6    | 645  | 5   | 645  | 5   | 32.7    | 645  | 5   | 32.7    |
| 7    | 734  | 5   | 734  | 5   | 46.4    | 718  | 5   | 47.2    |
| 8    | 613  | 4   | 613  | 4   | 37.2    | 613  | 4   | 37.2    |
| 9    | 840  | 6   | 816  | 6   | 38.1    | 801  | 6   | 39.5    |
| 10   | 803  | 5   | 747  | 5   | 42.7    | 793  | 5   | 46.0    |
| 11   | 1190 | 8   | 1190 | 8   | 71.7    | 658  | 4   | 37.3    |
| 12   | 1165 | 8   | 1100 | 7   | 70.8    | 985  | 7   | 61.1    |
| 13   | 1077 | 8   | 1077 | 8   | 67.5    | 845  | 6   | 57.8    |
| 14   | 903  | 6   | 903  | 6   | 52.5    | 725  | 5   | 40.8    |
| 15   | 935  | 6   | 911  | 6   | 50.3    | 868  | 6   | 60.0    |

*Note:* : The solution has transferred requests.
order of 1%-2%, regardless of driver shift length. As expected, the benefits decrease when driver shift lengths increase, because with longer driver shift lengths routes can cover larger distances and serve more requests in the same route, which tends to be less costly than using transfers.

The effect of the initial solution appears to be minor. Starting from an initial solution in which requests are transferred tends to result in final solutions with slightly more requests being transferred than starting from an initial solution in which no requests are transferred. This is likely due to the fact that insertions with transfer operators are more likely to be rewarded during the early stages of the search when the initial solution already has some transfers. When starting from a high-quality (locally optimal) solution without transfers, introducing transfers in the solution is likely to increase the total distance and insertions with transfer operators are less likely to be rewarded. For instances in which the distance between pickup and delivery location is short (S), and where transfers have a limited value, especially for larger shift lengths \(H = 240\) and \(H = 300\), starting from an initial solution with transfer is in fact detrimental to the quality of the final solution. We also note that the run time is slightly higher when starting from an initial solution with transfers, because, as noted earlier, the more time-consuming insert with transfer operations are performed more often.

Next, we consider transfer location geometry MD-5T. The results can be found in Table 6. The benefits are similar to what we have seen for transfer location geometry MD-4T, but for the fact that, on average, more requests are transferred, and, consequently, slightly higher CPU times are observed.

To show that our proposed ALNS algorithm scales well, we present results for instances with 75 and 100 requests in Tables 7 and 8, respectively. In each table, we show the results obtained for transfer location geometry MD-5T and using an initial solution with transfers. We observe once more that for instances in which there are requests with a long distance between pickup and delivery location, \(L\) and \(M\), the shift lengths are relatively short \(180\) and \(240\), the proposed ALNS algorithm is able to find solutions that reduce both the total distance and the number of vehicles used.

### 5.6 Evaluation of remove and insert operators

The performance of the remove and insert operators employed in our ALNS algorithm has been evaluated on two representative instances. The first instance, \(I_1\), has 50 long-only requests, shift length 180, and transfer location geometry MD-4T. As we have seen, such an instance is likely to benefit from transfers. The second instance, \(I_2\), has 50 short-only requests, shift length 300,
and transfer location geometry MD-4T—an instance less likely to benefit from transfers. We analyze which operators are used throughout the search when solving these two instances and also if this depends (strongly) on the initial solution.

Recall that the likelihood of an operator being selected is proportional to its weight, where the weight is initialized and updated during each segment of \( \kappa \) iterations (within a segment, operators are rewarded scores based on whether or not an operator advances the search, that is, obtains a new best solution, obtains an improving solution, or obtains a worse, but accepted solution). Therefore, to analyze the performance of an operator during the search, we report how the weight of the operator changes during the search and its contribution to advancing the search (the ratio of the number of times the operator advanced the search and the total number of times the search advanced).

### Table 6

Results for instances containing 50 requests and 3 hours time windows in the MD-5T setting

| Inst. | RP ALNS | Proposed ALNS | Average |
|-------|--------|---------------|---------|
|       | c Veh  | t(s) | s格 | \( \Delta_p (%) \) | \( \Delta_f (%) \) | Trans | t(s) | \( \Delta_p (%) \) | \( \Delta_f (%) \) | Trans | t(s) |
| 180, L | 6322.1 | 39.0 | 18.4 | 5.4 | 44.4 | 62.1 | -45.1 | -43.7 | 43.7 | 60.9 |
| 180, M | 4116.3 | 24.8 | 23.6 | 6.8 | 45.2 | 55.8 | -45.1 | -43.6 | 44.0 | 60.9 |
| 180, S | 2459.2 | 15.0 | 29.8 | 1.1 | 14.2 | 38.2 | -2.7 | -2.1 | 8.1 | 36.7 |
| 240, L | 3864.7 | 18.8 | 31.7 | 4.4 | 47.4 | 54.4 | -2.8 | -2.3 | 42.4 | 53.4 |
| 240, M | 3012.0 | 15.1 | 34.9 | 1.1 | 34.7 | 53.1 | -8.4 | -2.6 | 19.0 | 47.0 |
| 240, S | 2195.1 | 11.8 | 39.1 | 0.4 | 38.4 | 58.9 | -10.5 | -2.1 | 29.1 | 53.1 |
| 300, L | 3653.1 | 16.6 | 33.8 | 7.1 | 46.7 | 56.9 | -17.7 | -2.2 | 40.9 | 54.3 |
| 300, M | 2952.8 | 14.4 | 35.9 | 2.2 | 47.0 | 54.1 | -17.5 | -3.0 | 40.9 | 55.0 |
| 300, S | 2188.9 | 11.7 | 39.8 | 0.5 | 38.7 | 59.6 | -2.8 | 18.0 | 30.2 | 53.1 |

Note: Column RP ALNS reports results obtained with Ropke and Pisinger [25] (no transfers) and results obtained with our proposed ALNS (with transfers) are reported in column Proposed ALNS. In row \( s_f \), the initial solution to our method is given by the solution found with the ALNS by Ropke and Pisinger [25] and, in row \( s_f \), the initial solution is a solution in which some requests are initially transferred, as described in Section 4.1. Results for an instance in each class are the average over five runs. The ALNS of Ropke and Pisinger [25] provided the same solutions in all runs, thus we do not present Best of Average values for \( c \) or Veh.

### Table 7

Results for instances containing 75 requests and 3 hours time windows in the MD-5T setting

| Inst. | RP ALNS | Proposed ALNS | Average |
|-------|--------|---------------|---------|
|       | c Veh  | t(s) | \( \Delta_p (%) \) | \( \Delta_f (%) \) | Trans | t(s) | \( \Delta_p (%) \) | \( \Delta_f (%) \) | Trans | t(s) |
| 180, L | 8883.3 | 53.8 | 97.6 | -50.0 | -47.6 | 70.8 | 142.1 | -48.0 | -45.7 | 68.4 | 159.5 |
| 180, M | 5168.8 | 30.8 | 151.2 | -31.0 | -26.6 | 57.0 | 134.9 | -27.8 | -23.2 | 49.6 | 137.8 |
| 180, S | 3268.6 | 19.6 | 184.2 | -3.3 | 2.0 | 42.5 | 127.0 | -0.5 | 3.8 | 36.7 | 133.7 |
| 240, L | 5162.3 | 24.7 | 204.2 | -29.1 | -17.0 | 71.4 | 164.7 | -25.9 | -13.5 | 68.7 | 162.3 |
| 240, M | 3743.1 | 18.7 | 212.4 | -14.2 | -1.1 | 62.5 | 156.6 | -9.3 | 0.2 | 46.1 | 146.0 |
| 240, S | 2932.6 | 15.5 | 183.6 | 0.6 | 7.7 | 26.4 | 127.6 | 3.3 | 11.6 | 38.0 | 143.5 |
| 300, L | 4941.4 | 22.6 | 190.9 | -25.5 | -11.9 | 71.8 | 177.0 | -20.3 | -6.8 | 63.5 | 158.8 |
| 300, M | 3668.5 | 17.8 | 238.0 | -12.2 | 1.7 | 64.7 | 180.4 | -7.1 | 4.3 | 47.2 | 161.5 |
| 300, S | 2924.9 | 15.7 | 180.6 | 0.1 | 5.7 | 50.1 | 136.2 | 4.5 | 11.0 | 48.9 | 141.4 |

Note: Column RP ALNS reports results obtained with [25] (no transfers) and results obtained with our proposed ALNS (with transfers) are reported in column Proposed ALNS. Results for an instance in each class are the average over five runs. The ALNS of [25] provided the same solutions in all runs, thus we do not present Best of Average values for \( c \) or Veh. The initial solution for the proposed ALNS is a solution with transfers (\( s_f \)).
TABLE 8 Results for instances containing 100 requests and 3 hours time windows in the MD-5T setting

| Inst. | H, C | RP ALNS | Proposed ALNS | Average |
|-------|------|---------|---------------|---------|
|       |      |         | Best          |          |
|       |      |         | c Veh t(s)    | ΔD (%) | ΔV (%) | Trans | t(s) |
| 180,  | L    | 11613.5 | 69.7 163.9    | -53.1  | -50.5  | 97.6  | 277.6 | -51.1 | -48.2 | 95.4  | 300.2 |
| 180,  | M    | 7309.3  | 43.3 256.4    | -35.4  | -30.9  | 90.6  | 261.2 | -35.2 | -30.5 | 87.2  | 285.9 |
| 180,  | S    | 4071.9  | 24.0 335.3    | -9.0   | 0.8    | 67.9  | 280.3 | -2.4  | 3.0   | 50.6  | 280.8 |
| 240,  | L    | 6481.8  | 30.6 383.7    | -30.9  | -17.3  | 96.4  | 295.5 | -27.3 | -13.5 | 94.4  | 290.8 |
| 240,  | M    | 5069.5  | 24.9 450.4    | -20.8  | -7.2   | 93.0  | 310.8 | -16.7 | -4.3  | 85.9  | 322.1 |
| 240,  | S    | 3618.9  | 19.3 483.2    | -3.0   | 6.2    | 73.3  | 298.0 | 0.0   | 6.4   | 53.3  | 294.8 |
| 300,  | L    | 6186.0  | 28.2 432.6    | -26.0  | -9.9   | 98.0  | 332.8 | -22.3 | -6.0  | 94.1  | 322.1 |
| 300,  | M    | 4985.4  | 23.7 461.2    | -17.9  | -1.7   | 93.7  | 386.2 | -12.1 | 0.5   | 73.5  | 326.7 |
| 300,  | S    | 3619.7  | 19.1 461.5    | -2.1   | 7.3    | 59.7  | 317.1 | 1.2   | 7.0   | 45.0  | 288.5 |

Note: Column RP ALNS reports results obtained with Ropke and Pisinger [25] (no transfers) and results obtained with our proposed ALNS (with transfers) are reported in column Proposed ALNS. Results for an instance in each class are the average over five runs. The ALNS of Ropke and Pisinger [25] provided the same solutions in all runs, thus we do not present Best of Average values for c or Veh. The initial solution for the proposed ALNS is a solution with transfers ($s_t$).

FIGURE 6 Progression of operators during the search when solving instance $I_1$ starting from an initial solution without transfers (left) and with transfers (right). Removal operators: $O_1^-$, worst request; $O_2^-$, random request; $O_3^-$, route; $O_4^-$, transfer-based request; $O_5^-$, cluster. Insertion operators without transfers: $O_1^+$, travel time; $O_2^+$, insertion cost; $O_3^+$, regret; $O_4^+$, random. Insertion with transfers: $O_1^+$, travel time; $O_2^+$, insertion cost; $O_3^+$, regret; $O_4^+$, transfer [Color figure can be viewed at wileyonlinelibrary.com]

Figures 6 and 7 show the progressing of the operators for instances $I_1$ and $I_2$, respectively. In each figure, we show, separately, the progression of the removal and the insertion operators. We use a stacked bar graph to show the progression of the contribution to advancing the search and a line plot to show the progression of the weight. The values are collected at the end of every 500 iterations ($2\kappa$). The graphs on the left (labeled A on both Figures 6 and 7) show results obtained when using an initial solution without transfers, and the graphs on the right (labeled B on both Figures 6 and 7) show results obtained when using an initial solution with transfers.

Note that the stacked bars for the removal operators always sum up to 1, whereas some of the stacked bars for an insertion operator (with or without transfers) sum up to less than 1. This is due to the fact that two insert operators are selected at each iteration. If the first operator (with the largest weight) succeeds inserting all requests, then the second one is not used. Thus, for some iterations the search is advanced after applying only one of the selected operators.
Figure 7 shows that for both initial solutions, insertion operators using transfers are the ones contributing the most for advancing the search (note that the scales of the vertical axis of the weight charts are not same). The stacked bars show that, in many iterations, an insertion with transfer operator alone is able to advance the search (because in many cases the stacked bars for the insertion without transfer operators sums up to less than 1, indicating that the they were not used). The larger weights of the insertion with transfer operators also demonstrate that these operators are more successful in finding new best and better solutions. We see that insertion with transfer operator $O_8^+$ is mostly used at the start of the search process and more so when the initial solution does not have transfers. None of the other three insertion with transfer operators clearly dominates the others. However, we see that when the initial solution already has transfers, insertion with transfer operator $O_5^+$ is able to quickly find good transfer options for the requests with long distances between pickup and delivery locations (the initial spike in the weight chart shows the operator finds new best solutions). Regarding removal operators, note the important contribution of the transfer-based removal $O_4^-$, especially when starting with a solution without transfers.

When an instance benefits little from transfers, as is the case for instance $I_2$, the proposed ALNS algorithm recognizes this and adjusts accordingly (Figure 7). When the initial solution has no transfers, the search advances mainly using insertion without transfer operators. Even though insertion with transfer operator $O_8^+$ introduces a few unnecessary transfer visits early in the search, the transfer location is eventually removed from the solution. When the initial solution has transfers, improving solutions involving transfers are found in the early iterations, but as the search progresses and the insertion without transfer operators successfully insert more and more requests, the search advances primarily due to these operators (the turning point appears somewhere around iteration 17,000).

5.7 Block-based crowd-sourced transportation systems

In this section, we evaluate the impact of transfers in a block-based crowd-sourced systems, that is, system in which drivers are available for certain periods of time (blocks) during the planning. In particular, drivers commit to provide their services in one or more offered blocks, time slots of a given duration.

In the following experiments, we consider only instances with short and long requests (class M) and with only long requests (class L), as instance with only short requests (class S) do not benefit from transfers. Given a request $r = (i^+, i^-)$, the time window $[E_r, L_r]$ is generated as follows: $L_r = F + \tau_{i^+, i^-}$, where $F$ is a number drawn uniform randomly from $[60, 480 - \tau_{i^+, i^-}]$, and $E_r = F - 60$ (i.e., there is one hour of flexibility to perform the request). The planning horizon is nine hours, and crowd-sourced drivers sign up for three-hour blocks. We consider a geography similar to the one depicted in Figure 4B, but without locations $m_1 - m_4$, and where crowd-sourced drivers start and end their blocks at location $t_1$. 
Firstly, we consider a setting in which the planning horizon \([0,540]\) is divided into three nonoverlapping blocks \([0,180]\), \([180,360]\), and \([360,540]\). To assess the possible benefits provided by considering transfers, we compare the number of requests not served in a solution with and without the use of transfers (these requests would have to be served by dedicated, professional drivers). A visualization of the results can be found in Figure 8, which is based on 10 instances with 50 and 100 requests for class \(L\) and \(M\) (a total of 40 instances). We see, as expected, that the number of requests that cannot be served decreases when drivers are able to transfer requests. More requests with long direct travel times between pickup and delivery locations can be served using a transfer and drivers in different blocks. On average, 22% more requests can be satisfied by crowd-sourced drivers when transfers are utilized.

Next, we consider a setting in which the planning horizon \([0,540]\) is divided into overlapping blocks \([0,180]\), \([90,270]\), \([180,360]\), \([270,450]\), and \([360,540]\). The additional flexibility provided by overlapping blocks results in all requests being served in the instances, both with and without transfers. Therefore, to assess the benefits of transfers, we focus on the number of drivers required in each of the blocks. Figure 9 shows results for two specific instances, one in class \(M\) and the other in class \(L\), both with 100 requests. For each instance, the figure shows the number of open requests in the system at time \(t\) (x-axis), where a request \(r = (i^+, i^-)\) with time windows \([E_i^+, L_i^-]\) and \(E_i^+ \leq t \leq L_i^-\) is considered open, and the number of drivers used in a solution with and without the use of transfers. (Additional results, in which we average over all instances in a class, can be found in the appendix.) The two instances, however, are representative of the results observed.

First, we see that fewer drivers are needed when sharing is allowed, especially when there are only long requests. Second, we see that when transfers are allowed, the number of drivers in the system in the first block is larger than when transfers are not allowed. This signals that when transfers are allowed, many requests are picked up in the first block and brought to transfer points to be consolidated and delivered in subsequent blocks (which requires more drivers in the first block). As a consequence,
fewer drivers are required in subsequent blocks, as it takes less time to deliver consolidated requests. As a result, which may be advantageous, the number of drivers required across the blocks is more balanced.

6 | FINAL REMARKS

Our research has focused on investigating the potential benefits of using transfers in urban freight delivery systems. Our results show that these benefits can be significant, especially in settings where pickup and delivery locations are relatively far apart and driver shifts are relatively short. Thus, transfers may be especially valuable for urban freight delivery systems that rely, in part or completely, on crowdsourced delivery capacity, because crowdshippers tend to work for relatively short periods of time. Our results also show that the ability to transfer requests can substantially reduce the number of drivers required to serve a given number of requests. This too may be especially valuable in urban freight delivery systems that rely on crowdsourced delivery capacity, because crowdshippers are often compensated based on the number of requests they serve, and when the number of requests served per driver increases, it will become easier to attract crowdshippers to the system.

We are currently modifying our ALNS algorithm for use in dynamic environments, where requests appear throughout the operating period and (some of the) drivers may also enter and leave the system during the operating period, as this better reflects real-life settings.

ACKNOWLEDGMENTS

Funding for this research was provided by Netherlands Organization for Scientific Research (NWO), via the project DATAS for Cities (No. 438-15-507) and the Physical Internet Reisbeurzenprogramma (No. 439-17-703). The authors gratefully acknowledge the grants received to support their work.

REFERENCES

[1] C. Archetti, M. Savelsbergh, and M. Grazia Speranza, The vehicle routing problem with occasional drivers, Eur. J. Oper. Res. 254 (2016), 472–480.
[2] A. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk, Crowdsourced delivery—A dynamic pickup and delivery problem with ad hoc drivers, Transp. Sci. 53(1) (2019), 222–235.
[3] M. Behrend and F. Meisel, The integration of item-sharing and crowdshipping: Can collaborative consumption be pushed by delivering through the crowd? Transp. Res. B: Methodol. 111 (2018), 227–243.
[4] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, Static pickup and delivery problems: A classification scheme and survey, TOP 15 (2007), 1–31.
[5] G. Berbeglia, J.-F. Cordeau, and G. Laporte, Dynamic pickup and delivery problems, Eur. J. Oper. Res. 202 (2010), 8–15.
[6] H.B. Rai, S. Verlinde, J. Merckx, and C. Macharis, Crowd logistics: An opportunity for more sustainable urban freight transport? Eur. Transp. Res. Rev. 9 (2017), 39.
[7] C. Chen and S. Pan, “Using the crowd of taxis to last mile delivery in e-commerce: A methodological research,” Service Orientation in Holonic and Multi-Agent Manufacturing, T. Borangiu, D. Trentesaux, A. Thomas, and D. McFarlane (eds), Springer International Publishing, Berlin, 2016, pp. 61–70.
[8] C. Chen, S. Pan, W. Zhu, and R.Y. Zhong, Using taxis to collect citywide e-commerce reverse flows: A crowdsourcing solution, Int. J. Prod. Res. 55 (2017), 1833–1844.
[9] W. Chen, M. Mes, and M. Schutter, Multi-hop driver-parcel matching problem with time windows, Flexible Serv. Manuf. J. 30 (2017), 517–553.
[10] C.E. Cortés, M. Matamala, and C. Contardo, The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method, Eur. J. Oper. Res. 200 (2010), 711–724.
[11] L. Dahlke, H. Andersson, M. Christiansen, and M. Grazia Speranza, The pickup and delivery problem with time windows and occasional drivers, Comput. Oper. Res. 109 (2019), 122–133.
[12] I. Dayarian and M. Savelsbergh, Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders, Optim. Online (2017), http://www.optimization-online.org/DB_HTML/2017/07/6142.html
[13] V. Ghilas, E. Demir, and T. Van Woensel, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines, Comput. Oper. Res. 72 (2016), 12–30.
[14] G. Guastaroba, M. Grazia Speranza, and D. Vigo, Intermediate facilities in freight transportation planning: A survey, Transp. Sci. 50 (2016), 763–789.
[15] N. Kafle, B. Zou, and J. Lin, Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery, Transp. Res. B: Methodol. 99 (2017), 62–82.
[16] B. Li, D. Krushinsky, H.A. Reijers, and T. Van Woensel, The share-a-ride problem: People and parcels sharing taxis, Eur. J. Oper. Res. 238 (2014), 31–40.
[17] Y. Maknoon and G. Laporte, Vehicle routing with cross-dock selection, Comput. Oper. Res. 77 (2017), 254–266.
[18] R. Masson, F. Leluhédé, and O. Péton, An adaptive large neighborhood search for the pickup and delivery problem with transfers, Transp. Sci. 47 (2013), 344–355.
[19] R. Masson, F. Leluhédé, and O. Péton, The dial-a-ride problem with transfers, Comput. Oper. Res. 41 (2014), 12–23.
[20] S. Mitrovic and G. Laporte, The pickup and delivery problem with time windows and transshipment, INFOR: Inform. Syst. Oper. Res. 44 (2006), 217–227.
[21] Y. Nakao and H. Nagamochi, "Worst case analysis for pickup and delivery problems with transfer," IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E91-A** (2008), 2328–2334.

[22] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," Comput. Oper. Res. **34** (2007), 2403–2435.

[23] Q. Yuan and J.F. Bard, "A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment," Comput. Oper. Res. **39** (2012), 2439–2456.

[24] A. Rais, F. Alvelos, and M. Carvalho, "New mixed integer-programming model for the pickup-and-delivery problem with transshipment," Eur. J. Oper. Res. **235** (2014), 530–539.

[25] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," Transp. Sci. **40** (2006), 455–472.

[26] A. Sampaio, M. Savelsbergh, L. Veelenturf, and T. Van Woensel, "Crowd-based city logistics," *Sustainable Transportation and Smart Logistics: Decision-Making Models and Solutions, Chapter 15*, J. Faulin, S.E. Grasman, A.A. Juan, and P. Hirisch (eds), United States: Joe Hayton, 2018, 382–398.

[27] M. Savelsbergh and M. Sol, "The general pickup and delivery problem," Transp. Sci. **29**(1) (1995), 17–29.

[28] A. Stenger, D. Vigo, S. Enz, and M. Schwind, "An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping," Transp. Sci. **47** (2013), 64–80.

---

**How to cite this article:** Sampaio A, Savelsbergh M, Veelenturf LP, Van Woensel T. Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers. *Networks*. 2020;76:232–255. https://doi.org/10.1002/net.21963