INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2022) 1–37
DOI: 10.1111/itor.13234

Solving a pickup and delivery routing problem for fourth-party logistics providers

Preben Bucher Johannessen[a], Ahmad Hemmati[a] and Mohammad Moshref-Javadi[b,*]

[a] *Department of Informatics, University of Bergen, Norway*
[b] *Gies College of Business, University of Illinois at Urbana Champaign, Champaign, IL 61820, USA*
E-mail: johannessen@deutschebahn.com [Johannessen]; ahmad.hemmati@uib.no [Hemmati]; moshref@illinois.edu [Moshref-Javadi]

Received 1 December 2021; received in revised form 8 August 2022; accepted 24 October 2022

Abstract

This paper studies a pickup and delivery routing problem for fourth-party logistics providers. The problem aims to schedule routes of vehicles to pick up orders from suppliers and deliver them to factory locations considering multiple time windows at suppliers and factory locations, a non-conventional cost structure, and certain factory dock constraints. We formulate the problem as a mathematical model and develop an efficient algorithm based on the adaptive large neighborhood search to solve the problem. The algorithm incorporates several heuristics to efficiently explore the search space for optimal solutions. The algorithm is refined through extensive statistical experiments to optimize the performances of the heuristics and to tune the parameters of the algorithm. The mathematical model and algorithm are evaluated on several problem instances based on a real case study in Europe. The numerical results demonstrate that the solution algorithm consistently obtains near-optimal solutions to real-sized problem instances.

Keywords: fourth-party logistics; multiple time windows; metaheuristic; pickup and delivery; routing problem

1. Introduction

The automobile and machinery manufacturing industry is dynamically changing due to the international market and cost competition (4Flow, 2019). This has made the supply chain and logistics of paramount importance in this industry to not only perform cost-efficiently but also deliver the auto parts and products to the worldwide markets with minimum delays. This challenging task highlights the need for external expertise for auto manufacturers to manage the logistics operations, providing the ground and needs for the fourth-party logistics (4PL) companies. These providers aim to manage the logistics and distribution operations across and beyond the companies, and even

*Corresponding author.

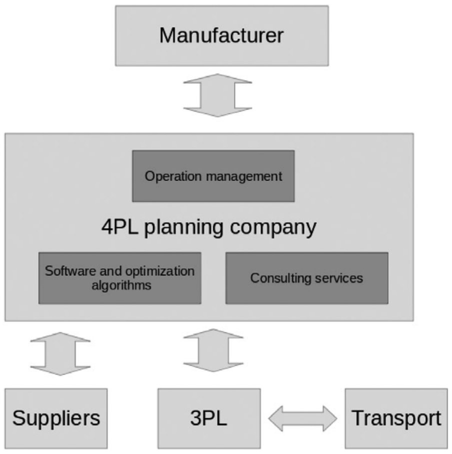


Fig. 1. The role of a 4PL company for providing service to a manufacturer, suppliers, the third-party logistics provider (3PL), and transportation companies. The arrows represent the flow of information

countries, with high flexibility in planning and modeling of logistics operations to provide efficient and robust solutions. The manufacturers, through the 4PL providers, outsource several activities to the third-party providers (3PL). These activities include not only the management of their logistics operations but also the organizational and executive activities, management of resources, scheduling, and coordination between different involved parties, often across the entire supply chain. Figure 1 illustrates the role of a 4PL company in relationships with a manufacturer as its customer, 3PL company, supplier, and transportation company. This figure shows that the 4PL company serves as the manager of transportation- and logistics-related operations to plan and manage resources on behalf of the manufacturers.

One of the main challenges in 4PL operations is the *routing problem*, which is represented as a *pickup and delivery problem with time windows* (PDPTW). In this problem, the number of vehicles and sequences of pickups and deliveries for each vehicle must be obtained such that all the orders are picked up within the predetermined time windows from the suppliers and delivered to the factories within the predetermined time windows. However, the role of a 4PL company differentiates this problem from the classic pickup and delivery problems. Most importantly, since a 4PL company is not handling its own fleet of vehicles, it relies on other third-party transportation companies, and, therefore, it depends on their spot prices. This has several implications for a 4PL company's cost structure, as illustrated in Fig. 2. Primarily, this cost structure has three characteristics which differ from the conventional transportation cost structures. First, it does not include the transportation costs from the origin to the first stop and also from the last stop to the origin because the 4PL company does not own the vehicles. Second, the spot price of moving products from location A to B to C to D is calculated based on the total *vehicle schedule* $ABCD$, which depends on several factors, particularly including distance, product's weight, and fixed labor costs. Finally, the transportation company charges the manufacturer a fee per stop. All of these prices can vary depending on the vehicle types and transportation carriers, such as less-than-truckload or full-truckload. Considering all of the aforementioned complexities, planning and scheduling of routes by the 4PL companies is a complicated problem. This problem can be significantly challenging when various constraints

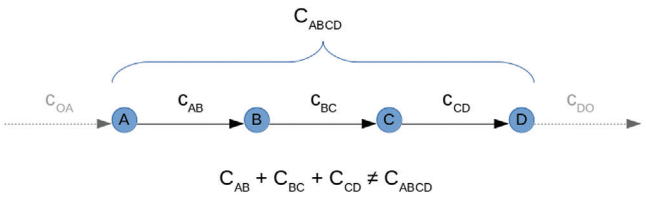


Fig. 2. Spot prices when using a transportation company to serve different stops in a vehicles schedule. The 4PL company does not pay for the costs of transportation from and to the origin, C_{OA} and C_{DO}. The total cost of vehicle schedule C_{ABCD} depends on the distance traveled, the total weight of transportation, and the fixed costs of hiring the transportation company for the task

are incorporated in the problem, such as the number of vehicles, working hours, vehicle capacity, and dock management rules in factories. Due to the fact that the fleet of vehicles is not owned by the 4PL company or manufacturers, the number of vehicles is not limited to the vehicles owned by a single company. Rather, a 4PL company schedules orders for pickups and deliveries performed by various 3PL companies. Therefore, obtaining high-quality vehicle routes and effectively using the available vehicles from various companies result in a significant amount of cost savings, as well as reduction in greenhouse gas emissions. Another constraint is considering available working hours of companies, as well as the defined time windows of manufacturers and suppliers across different days. Therefore, the scheduling of pickups and deliveries must be performed accordingly, ensuring minimum delays while maximizing resource utilization. In addition, the 4PL companies need to consider the availability of resources at customer locations, especially dock availability and dock–vehicle compatibility constraints. For example, some manufacturers may have more than one loading/unloading docks at factories which can be used to serve certain types of vehicles based on the available equipment, such as cranes, or compatibility of the vehicles and docks. The manufacturer might also limit the number of docks that can be visited by each vehicle for delivering the products to reduce traffic in the yard.

Due to such complexities, it is essential for 4PL companies to use efficient solution algorithms to be able to obtain optimal or near-optimal solutions to the underlying routing and scheduling problems. In this paper, we study the fourth-party logistics problem (4PLP). We present a mathematical model of the problem and develop an efficient algorithm to solve the underlying *pickup and delivery problem with multiple time windows* (PDPMTW) which considers several real-world constraints, such as time windows at suppliers and factories, resource constraints at factories, and a complex cost structure. The model and algorithm, however, are developed in a general framework for use in other industries used by the 4PL providers. This problem incorporates several real-world constraints, including type, capacity, and availability of vehicles, dock availability at factories, vehicle–dock compatibility, and product–vehicle compatibility.

The rest of the paper is organized as follows. Section 2 reviews the literature on the relevant routing problems. Section 3 presents the 4PLP along with the mathematical formulation. Section 4 describes the developed solution algorithm to solve the 4PLP. The detailed experimental results are presented in Section 5. We also analyze the performance of the algorithm in comparison with the benchmark results of the mathematical model. Finally, Section 6 concludes the paper and presents the potential directions for future research.

2. Literature review

The pickup and delivery problem (PDP) aims to schedule a set of orders for transportation between some given pairs of origins and destinations. The PDP is also known as a generalization of the vehicle routing problem in which either all the origins or destinations are located at the depot (Savelsbergh and Sol, 1995). This problem has various applications in ground transportation (Wang et al., 2015), maritime transportation (Hemmati et al., 2014), and air transportation (Azadian et al., 2017). Some surveys and classifications of the PDP are presented by Mitrovic-Minic (1998), Berbeglia et al. (2007, 2010), and Parragh et al. (2008).

Several extensions to the pickup and delivery problem are modeled based on various real-world assumptions and constraints, such as the pickup and delivery problem with time windows (PDPTW). Ghilas et al. (2016) consider the PDPTW and scheduled truck routes. The problem concerns scheduling a set of vehicles to serve customer orders such that each trip can be partly or fully served using public transportation services. The problem is solved using an adaptive large neighborhood search (ALNS) algorithm. Goeke (2019) study the pickup and delivery routing problem with time windows for electric vehicles. Due to the limited battery capacity of electric vehicles, they stop at recharging locations for partial or full recharge. They propose a granular tabu search algorithm to solve this problem with several experiments on small- and large-size problem instances. Favaretto et al. (2007) consider the pickup and delivery problem with multiple time windows and multiple visits to pickup and delivery locations. An algorithm based on the ant colony system is designed to solve the problem. An exact algorithm based on set partitioning and cutting planes for the green pickup and delivery problem is developed by Sun et al. (2019). The problem aims to minimize the total carbon emissions in a pickup and delivery problem with a heterogeneous fleet of vehicles. Manier et al. (2016) present an exact mathematical model to solve the PDPMTW with vehicle time and capacity constraints. The model is used to solve problems with up to 50 nodes. A variable neighborhood search algorithm to solve the PDPMTW is proposed by Ferreira et al. (2018). Since the algorithm only generates feasible solutions, all the computation time is spent on searching the feasible space, enhancing the efficiency of the algorithm. Dondo and Cerdá (2015) consider the pickup and delivery routing problem with time windows and cross-dock scheduling. In addition to the vehicle routing problem for pickups and deliveries, the assignment of the available docks at facilities for loading and unloading tasks is also considered. A branch-and-cut search and a sweep-based model are developed to solve the problem. A new variant of the PDPTW is developed by Dahle et al. (2019), the so-called PDPTW with occasional drivers. This problem considers different payment schemes for drivers which enable detours and divergence from the optimal routes when occasional drivers are not considered in the problem. The problem is formulated as a mixed-integer programming model, which is used to solve instances with 70 orders.

Since the PDP is an NP-hard problem, several articles focus on proposing efficient algorithms to solve this problem and its variants. Nanry and Barnes (2000) propose a tabu search algorithm to solve the PDPTW considering capacity constraints of the vehicles. The algorithm differs from the conventional tabu search algorithm in that it justifies the strategies and parameters of the algorithm, such as neighborhood search and switch strategies, based on the performance of the algorithm. Li and Lim (2003) propose a tabu-embedded simulated annealing meta-heuristic to solve the PDPTW. Besides the problem instances in Nanry and Barnes (2000), they solve several generated instances based on Solomon (1987). Lau and Liang (2002) develop a tabu search to solve the PDPTW, using several embedded construction heuristics to generate an initial solution. They also propose a strategy to generate problem instances and benchmarking solutions for the PDPTW. Bent and Van Hentenryck (2006) present a two-stage hybrid algorithm for the PDPTW. The first stage uses a simulated annealing algorithm to reduce the number of routes, while the second stage uses the large neighborhood search (LNS) to reduce the total travel cost. The results provide new best solutions to several benchmark instances in Li and Lim (2003). Ropke and Pisinger (2006) propose an ALNS algorithm for the PDPTW. The ALNS heuristic is composed of a number of competing subheuristics which are used adaptively with respect to their past performances. The algorithm is evaluated on more than 350 benchmark instances with up to 500 orders. Using this algorithm, they improve the best known solutions to more than 50% of the problem instances. A new two-index mathematical model of the PDPTW is formulated by Furtado et al. (2017). This two-index formulation enables the assignments of vehicles to routes directly which enhances the computational performance of the algorithm. Wang et al. (2015) develop a simulated annealing algorithm for the simultaneous pickup and delivery problem with time windows. The algorithm is built based on the concept of multiple Markov chains which compares the best obtained solutions obtained after a few threads of runs and uses the best of the solutions for the next segments of runs. The algorithm improves the best obtained solutions for several problem instances adopted from the literature.

In this paper, we consider a new problem, the 4PLP, which is a PDP with multiple time windows. The problem considers several real-world constraints, including the number of vehicles, a nonconventional cost structure, dock limitations at factory locations, compatibility of orders and trucks, and compatibility constraints of trucks and factory docks. To solve this problem, we mathematically model the problem and develop an algorithm based on the ALNS. The algorithm uses several heuristics to search the solution space effectively. In summary, the main contributions of the paper are the followings:

- Propose a new routing and scheduling problem faced by 4PL providers. The problem, named the 4PLP, is a PDPMTW, considering several real-world constraints, such as multiple time windows at supplier and factory locations, available vehicles, available docks at factories, a complex cost structure, and compatibility of vehicles, orders, and docks at factories.
- Formulate a mixed integer programming model of the problem.
- Develop an efficient algorithm based on the ALNS which utilizes seven insertion and removal heuristics. The algorithm also uses a heuristic to change the search neighborhood to escape from local optimal solutions.
- Present a systematic and statistics-based approach to evaluate and select the promising combination of the heuristics for the algorithm.

3. Problem description and formulation

The 4PLP aims to schedule routes of vehicles to pick up orders from suppliers and deliver them to factory locations regarding several operational constraints and prespecified cost structures. Each order is picked up at a supplier by a truck and delivered to a dock at a factory location. This problem considers several constraints, such as dock availability constraints at factory locations, pickup and delivery time windows, and compatibility of orders, vehicles, and docks which are explained in the following. The goal of the problem is to optimize the schedules of the vehicles' stops at the supplier and factory locations to minimize the total costs of transportation, vehicle stops, and penalty of unserved orders.

Each factory location has one or multiple docks where the vehicles can stop to unload the products. Each vehicle may pick up products from one or multiple suppliers and deliver them to one or multiple docks at one or multiple factory locations. The vehicle does not have to perform all the pickups before it can perform the deliveries. That is, once a vehicle is done with some of the deliveries to factory locations, it may visit a supplier on its route to pick up more orders. At a factory location, a vehicle may visit one or multiple docks within the same factory to deliver the orders. Each factory determines an upper limit on the number of docks that can be used by a vehicle in a visit. This constraint enforced by the factories aims to reduce traffic in the factory yard and, thus, to facilitate pickups and deliveries. Each supplier or factory location can determine multiple time windows for the vehicle to pick up or deliver orders, respectively. If a vehicle arrives earlier than the time window lower bound, it has to wait until the time window starts. Given the hard time windows, all the orders must be served within the specified time windows.

The fleet of vehicles consists of a set of heterogeneous trucks provided by different logistic carries companies or through the 3PL carriers. The vehicles are heterogeneous with respect to the capacity limit, which is represented by the maximum weight and volume of the products that can be carried by a vehicle. We also consider the compatibility of the vehicles, products, and factories. Some products are not compatible with certain vehicles, for example, because the vehicle does not have the required temperature-control capabilities, and, therefore, they cannot be carried by the vehicle. Also, some vehicles are not compatible with some suppliers/factories due to the fact that the supplier/factory does not have the specific equipment needed to load/unload the shipment at that location.

The total costs consist of three parts. The first part is the cost of transportation for a route which is calculated based on the maximum onboard weight on a route (price per kg), as well as the maximum distance traveled by the truck on the route (price per kilometer). Based on these two measures, a fixed cost, a weight cost, and a distance cost are applied to each order. Since the transportation of products starts from the suppliers, the travel cost of the vehicle to the first pickup location is not included in the total costs. The second part is the fixed stop cost, which is associated with each stop of the vehicle at the supplier and factory locations. Finally, the third part represents the costs of unserved orders.

3.1. Mathematical model

Tables 1–3 define the notations used to formulate the mixed integer linear programming (MILP) model of the problem. The 4PLP can be represented as a graph, $G(A, N)$, where $N = \{1, 2, \dots, 2n\}$ is the set of nodes, n is the number of orders in the problem, and $A = \{(i, j) : i, j \in N, i \neq j\}$ is the set of arcs in the graph. If i is an order pickup node, then $i + n$ is its corresponding delivery node. The set of pickup nodes (suppliers) is denoted by $N^P := \{1, 2, \dots, n\}$, and the set of delivery nodes is denoted by $N^D := \{n + 1, n + 2, \dots, 2n\}$. Therefore, the set of all nodes is equivalent to $N = N^P \cup N^D$. Each delivery node i, that is, dock, belongs to a factory $f \in F$, forming the set of all docks in factory f denoted by N_f. A vehicle may visit multiple docks in a factory to deliver

Table 1
Indexes and sets used in the mathematical model

Indexes	
v	Index of vehicles
i	Index of nodes of the network
f	Index of factory
p	Index of time window
s	Index of stop location
α	Index of distance interval in cost structure
β	Index of weight interval in cost structure
Sets	
N	Set of all nodes $\{1, 2, \ldots, 2n\}$, where n is the number of orders
V	Set of vehicles
A	Set of arcs
A_v	Set of arcs that can be visited by vehicle v
N_v	Set of nodes that can be visited by vehicle v
N^P	Set of pickup nodes $[1, 2, \ldots, n]$
N^D	Set of delivery locations $[n+1, n+2, \ldots, 2n]$
F	Set of factories
N_f	Set of delivery docks in factory f
E_v	Set of elements (α, β) in the distance-weight matrix of the cost structure for vehicle v
P_i	Set of time windows for node i, $\{1, 2, \ldots, \pi_i\}$
T_i	Set of time window parameters $[\underline{T_{ip}}, \overline{T_{ip}}]$ at node i, where $p \in P_i$
S	Set of stops, including all pickup and factory locations
L_s	Sets of nodes in a stop location $s \in S$

orders. However, the number of docks that can be visited by a vehicle in factory f is limited by upper bound H_f. We also denote the set of all possible stops by $s \in S$. Therefore, L_s denotes the set of nodes which are located at stop s.

The set of vehicles is denoted by V, and the weight and volume capacities of each vehicle $v \in V$ are represented by K_v^{kg} and K_v^{vol}, respectively. We also introduce A_v and N_v as the set of arcs and nodes that each vehicle $v \in V$ can traverse, respectively. Also, N_v includes an origin node, $o(v)$ and a destination node $d(v)$ which are two unique auxiliary start and end nodes for each vehicle v. Since a 4PL company does not own its fleet and rather pays to other carriers for transportation, we do not include the costs of transportation from $o(v)$ to the first pickup location and also the costs of return trip of the vehicle to $d(v)$ after serving the last factory location.

Each order from pickup node $i \in N^P$ has weight Q_i^{kg} and volume Q_i^{vol}. Also, each node has a set of time windows T_i represented by $[\underline{T_{ip}}, \overline{T_{ip}}] \in [0, T]$, where $p \in P_i = \{0, 1, \ldots, \pi_i\}$ denotes the pth time window. All nodes, both pickup and delivery nodes, must be visited within one of the given time windows. The distance from node i to node j is denoted by D_{ij}, and the corresponding travel time for vehicle v is represented by T_{ijv}.

The cost structure consists of three components. First, the cost structure of vehicle v depends on the total distance that the vehicle travels as well as the maximum onboard weight of products that are transported by this vehicle. This cost is represented by two sets of cost intervals for both

Table 2
Parameters and variables used in the mathematical models

Parameters	
n	Number of orders
K_v^{kg}	Weight capacity of vehicle $v \in V$
K_v^{vol}	Volume capacity of vehicle $v \in V$
$o(v)$	Start node of vehicle v
$d(v)$	End node of vehicle v
Q_i^{kg}	Weight of order at node $i \in N$
Q_i^{vol}	Volume of order at node $i \in N$
H_f	A limit on the number of docking at each visit to factory $f \in F$
T_{ijv}	Travel time of vehicle $v \in V$ along arc $(i, j) \in A_v$
π_i	Number of time windows at node $i \in N$
$\overline{T_{ip}}$	Upper bound of time window $p \in P_i$ at node $i \in N$
$\underline{T_{ip}}$	Lower bound of time window $p \in P_i$ at node $i \in N$
γ_v	Number of distance intervals for vehicle v to use in matrix E_v
μ_v	Number of weight intervals for vehicle v to use in matrix E_v
$C_{v\alpha\beta}^{km}$	Cost per distance unit (km) based on distance-weight cost matrix element $(\alpha, \beta) \in E_V$ for vehicle v
$C_{v\alpha\beta}^{kg}$	Cost per weight unit (kg) based on distance-weight cost matrix element $(\alpha, \beta) \in E_V$ for vehicle v
$C_{v\alpha\beta}^{fix}$	Fixed cost based on distance-weight cost matrix element $(\alpha, \beta) \in E_V$ for vehicle v
C_i^{stop}	Costs of stopping at node i
C_i	Penalty cost of not serving order $i \in N^P$
D_{ij}	Distance between node $i \in N$ and $j \in N$
B_α	Upper limit on interval α in cost matrix E_V
Z_β	Upper limit on interval β in cost matrix E_V

Table 3
Parameters and variables used in the mathematical models

Decision variables	
x_{ijv}	Binary variable, equals 1 if vehicle v travels from node $i \in N$ to $j \in N$, otherwise 0
y_i	Binary variable, equals 1 if order $i \in N^P$ is not picked up, otherwise 0
l_{iv}^{kg}	Weight of vehicle v after visiting node i
l_{iv}^{vol}	Volume of vehicle v after visiting node i
h_i	Number of times docked inside a factory after visiting node $i \in N^D$
t_i	Arrival time of vehicle at node $i \in N$
u_{ip}	Binary variable, equals 1 if time window $p \in P_i$ at node i is used, otherwise 0.
$d_{v\alpha\beta}$	Total distance traveled by vehicle $v \in V$ if it fits in distance-weight pair $(\alpha, \beta) \in E_v$
$b_{v\alpha\beta}$	Binary variable, equals 1 if pair $(\alpha, \beta) \in E_v$ is used for vehicle $v \in V$, otherwise 0.
$l_{v\alpha\beta}$	Highest weight transported by vehicle $v \in V$ for pair $(\alpha, \beta) \in E_v$

the weight and distance measures. Each possible combination of weight and distance is represented by an index pair (α, β), where α is the distance interval index belonging to set $\{1, 2, \ldots, \gamma_v\}$ and β is the weight interval belonging to set $\{1, 2, \ldots, \mu_v\}$. The cost matrix E_v is created based on all the combinations of α and β values. Since the three components of the cost structure, fixed, distance, and weight measures depend on the (α, β) pair, therefore, the cost structure forms three cost matrices, including fixed cost matrix $C_{v\alpha\beta}^{fix}$, distance matrix $(C_{v\alpha\beta}^{km})$, and weight matrix $(C_{v\alpha\beta}^{kg})$.

The second component of the cost structure refers to each vehicle's stop, that is, once vehicle v stops at a supplier or factory i, a fixed stop cost C_i^{stop} is charged. Finally, the cost structure applies a penalty C_i in the objective function for every unserved order i.

Variable t_i denotes the visit time of node $i \in N$. Each delivery node is also assigned a variable (h_i) which indicates the number of docks in the factory used by a vehicle. Variables l_{iv}^{kg} and l_{iv}^{vol} represent the weight and volume of load transported by vehicle v after it departs node i. Variable x_{ijv} is a binary variable indicating if vehicle v travels between nodes i and j. The cost of not serving an order corresponding to node i is represented by C_i using a binary variable y_i, set to 1 if the corresponding order to node i is not picked up. The total distance traveled by vehicle v is denoted by variables $d_{v\alpha\beta}$ for each pair $(\alpha, \beta) \in E_V$ and $v \in V$. The maximum weight transported by a vehicle is represented by $l_{v\alpha\beta}$. Similarly, only one of these variables per vehicle obtains the maximum transported weight value, which is determined by the binary variable $b_{v\alpha\beta}$. Each $b_{v\alpha\beta}$ has a corresponding distance parameter B_α and a weight parameter Z_β which represent the upper bounds of the intervals in the distance and weight cost matrix, respectively. Using the defined notations, the mathematical formulation of the problem is represented as follows:

$$\min \sum_{v\in V} \sum_{(\alpha,\beta)\in E_V} (C_{v\alpha\beta}^{km} d_{v\alpha\beta} + C_{v\alpha\beta}^{kg} l_{v\alpha\beta} + C_{v\alpha\beta}^{fix} b_{v\alpha\beta}) + \sum_{v\in V} \sum_{s\in S} \sum_{\substack{i\in L_s \\ j\in N_v \notin L_s}} C_i^{stop} x_{ijv} + \sum_{i\in N^P} C_i y_i \tag{1}$$

subject to

$$\sum_{v\in V} \sum_{j\in N_v} x_{ijv} + y_i = 1, \quad i \in N^P, \tag{2}$$

$$\sum_{j\in N_v} x_{ijv} - \sum_{j\in N_v} x_{jiv} = 0, \quad v \in V, i \in N_v \notin \{o(v), d(v)\}, \tag{3}$$

$$\sum_{j\in N_v} x_{o(v)jv} = 1, \quad v \in V, \tag{4}$$

$$\sum_{j\in N_v} x_{jd(v)v} = 1, \quad v \in V, \tag{5}$$

$$\sum_{j\in N_v} x_{ijv} - \sum_{j\in N_v} x_{(i+n)jv} = 0, \quad v \in V, i \in N_v^P, \tag{6}$$

$$l_{iv}^{kg} + Q_j^{kg} - l_{jv}^{kg} \le K_v^{kg}(1 - x_{ijv}), \quad v \in V, j \in N_v^P, (i, j) \in A_v, \tag{7}$$

$$l_{iv}^{kg} - Q_j^{kg} - l_{(j+n)v}^{kg} \le K_v^{kg}(1 - x_{i(j+n)v}), \quad v \in V, j \in N_v^P, (i, n + j) \in A_v, \tag{8}$$

$$0 \le l_{iv}^{kg} \le K_v^{kg}, \quad v \in V, i \in N_v^P, \tag{9}$$

$$l_{iv}^{vol} + Q_j^{vol} - l_{jv}^{vol} \le K_v^{vol}(1 - x_{ijv}), \quad v \in V, j \in N_v^P, (i, j) \in A_v, \tag{10}$$

$$l_{iv}^{vol} - Q_j^{vol} - l_{(j+n)v}^{vol} \le K_v^{vol}(1 - x_{i(j+n)v}), \quad v \in V, j \in N_v^P, (i, n + j) \in A_v, \tag{11}$$

$$0 \le l_{iv}^{vol} \le K_v^{vol}, \quad v \in V, i \in N_v^P, \tag{12}$$

$$h_i + 1 - h_j \leq (H_f + 1)(1 - x_{ijv}), \quad v \in V, f \in F, i \in N_f, j \in N_f, j \neq i, \tag{13}$$

$$h_j \leq H_f, \quad f \in F, j \in N_f, \tag{14}$$

$$h_j \geq \sum_{\substack{i \in N_v \\ i \notin N_f}} (x_{ijv}), \quad v \in V, f \in F, j \in N_f, \tag{15}$$

$$\sum_{p \in P_i} u_{ip} = 1, \quad i \in N, \tag{16}$$

$$\sum_{p \in P_i} u_{ip} \underline{T_{ip}} \leq t_i, \quad i \in N, \tag{17}$$

$$\sum_{p \in P_i} u_{ip} \overline{T_{ip}} \geq t_i, \quad i \in N, \tag{18}$$

$$t_i + T_{ijv} - t_j \leq (\overline{T_{i\pi_i}} + T_{ijv})(1 - x_{ijv}), \quad v \in V, (i, j) \in A_v, \tag{19}$$

$$t_i + T_{i(i+n)v} - t_{(i+n)} \leq 0, \quad v \in V, i \in N_v^P, \tag{20}$$

$$\sum_{(\alpha,\beta) \in E_V} d_{v\alpha\beta} = \sum_{(i,j) \in A_v} x_{ijv} D_{ij}, \quad v \in V, \tag{21}$$

$$\sum_{(\alpha,\beta) \in E_V} l_{v\alpha\beta} \geq l_{iv}^{kg} \quad v \in V, i \in N_v, \tag{22}$$

$$B_{(\alpha-1)} b_{v\alpha\beta} \leq d_{v\alpha\beta} \leq B_\alpha b_{v\alpha\beta}, \quad v \in V, (\alpha, \beta) \in E_V, \tag{23}$$

$$Z_{(\beta-1)} b_{v\alpha\beta} \leq l_{v\alpha\beta} \leq Z_\beta b_{v\alpha\beta}, \quad v \in V, (\alpha, \beta) \in E_V, \tag{24}$$

$$\sum_{(\alpha,\beta) \in E_V} b_{v\alpha\beta} \leq \sum_{j \in N_v} x_{o(v)jv}, \quad v \in V, \tag{25}$$

$$h_i, t_i \geq 0, \quad i \in N, \tag{26}$$

$$u_{ip} \in \{0, 1\}, \quad i \in N, p \in P_i, \tag{27}$$

$$b_{v\alpha\beta} \in \{0, 1\}, \quad v \in V, (\alpha, \beta) \in E_V, \tag{28}$$

$$d_{v\alpha\beta}, l_{v\alpha\beta} \geq 0 \quad v \in V, (\alpha, \beta) \in E_V, \tag{29}$$

$$y_i \in \{0, 1\}, \quad i \in N^P, \tag{30}$$

$$x_{ijv} \in \{0, 1\}, \quad v \in V, (i, j) \in A_v, \tag{31}$$

Objective function (1) represents the total costs to be minimized. The first term includes the transportation costs calculated based on the order weight, vehicle-traveled distance, and fixed cost. These three costs depend on the maximum onboarding weight and truck-traveled distance represented by pair (α, β). The second term indicates the cost per stop of the vehicle at pickup and factory locations, and the third term is the penalty cost paid for unserved orders. Constraint (2) represents that an order is served if it is picked up only once by only a vehicle; otherwise, it is considered as an unserved order. Constraints (3)–(6) govern the flow of orders at each node visited by each vehicle, such that a vehicle has to leave a node if it enters it. These constraints also guarantee that each vehicle route starts and ends at the defined auxiliary depots. The total weight carried by each vehicle based on the visited pickup and factory locations is calculated by Constraints (7) and (8). Constraint (9) represents the weight capacity constraint for each vehicle. Similarly, Constraints (10)–(12) calculate the total volume loaded on a truck and also ensure the volume capacity constraints of the vehicles. Constraint (13) calculates the number of docks visited by a vehicle located in a certain factory, and Constraints (14) and (15) ensure that the number of visited docks in each factory by each vehicle cannot exceed the number of docks in the factory. Constraint (16) indicates that only one of the available time windows is used per node, while Constraints (17) and (18) guarantee that the vehicle visits each node within the time windows defined by the lower and upper bounds of the time window. Constraint (19) computes the arrival times of the vehicle at each node. Constraint (20) ensures that all the picked up orders must be delivered, and this delivery is done after the order is picked up at the supplier node. Constraint (21) calculates the total distance traveled by each vehicle. Note that only one variable $d_{v\alpha\beta}$ obtains the value equal to the total traveled distance by that vehicle, and the variables for the remaining pairs of (α, β) are set to zero. Similarly, Constraint (22) calculates the maximum onboarding weight transported by each vehicle. Constraint (23) corresponds to the total distance traveled by each vehicle to the defined distance interval α. Similarly, Constraint (24) relates the maximum onboarding transported weight by the vehicle to the defined weight interval β. Constraint (25) indicates that we do not consider a cost interval for those vehicles that do not leave their origin depot, and, therefore, we do not assign any fixed costs to such vehicles. Finally, Constraints (26)–(31) define the types and ranges of the variables.

4. The proposed solution algorithm

Due to the NP-hardness of the problem and high computation time based on our preliminary experiments, we develop a metaheuristic algorithm to solve this problem. An effective and efficient meta-heuristic algorithm needs to employ both intensification and diversification strategies to not only find better quality solutions but also effectively escape from local optimal solutions (Blum and Roli, 2003).

Our proposed algorithm is built based on the ALNS (Ropke and Pisinger, 2006), which is an extension of the LNS algorithms introduced by Shaw (1997). This algorithm employs the diversification and intensification strategies, particularly by using several heuristics, so-called the removal and insertion heuristics. Additionally, the ALNS stores and evaluates the performances of all the heuristics during the search process and adaptively chooses and applies the most effective heuristics based on their performance history.

The overview of the algorithm is shown in Algorithm 1. The algorithm starts with generating an initial solution s and saves the best solution s_{best} as the current solution s. Thereafter, the main loop starts which continues until the stop conditions are met. At the beginning of this loop, the algorithm evaluates whether it is stuck in a local optimal by counting twihe number of iterations that the best found solution is not improved. If this value is not improved for a specified number of iterations, the algorithm applies the wild escape heuristic to change the search neighborhood (see Section 4.6). In each iteration of this loop, a heuristic h is chosen and applied to the current solution s'. These

Algorithm 1. Proposed algorithm

```
1  Define the sets of Insertion and Removal heuristics $\mathbb{H}$
2  Generate initial solution $s$, initiate heuristic selection parameters
3  $s_{best} \leftarrow s$
4  Number of iterations after the current best global solution found $i \leftarrow 0$
5  **repeat**
6      **if** $i > escape\ condition$ **then**
7          Apply Wild Escape Algorithm to $s$ (see Section 4.6)
8      **end**
9      $s' \leftarrow s$
10     Select a heuristic, $h \in \mathbb{H}$ based on selection parameters
11     Apply heuristic $h$ to $s'$
12     **if** $f(s') < f(s_{best})$ **then**
13         $s_{best} \leftarrow s'$
14     **end**
15     **if** $accept(s', s)$ **then**
16         $s \leftarrow s'$ (see Section 4.5)
17     **end**
18     Update selection parameters and iteration counter $i$
19 **until** *stop condition is met*;
20 **return** $s_{best}$ and $f(s_{best})$
```

heuristics choose a number of demand orders q to be removed from the current solution s' using the removal heuristics and then be reinserted to the current solution using the insertion heuristics. The number of orders (q) that are removed and reinserted to the current solution can be used to control the level of diversification/intensification during the search process. Once the new modified solution s' is evaluated, the best found solution s_{best} is updated if the new solution results in a better objective function value. The algorithm not only accepts all the improving solutions but also accepts the nonimproving solutions probabilistically to be able to escape from local optimal. Therefore, solution s is updated if the modified solution s' is accepted. We use the Boltzmann probability function from the simulated annealing algorithm to accept non-improving solutions (Kirkpatrick, 1984) based on function $accept(s', s)$ described in Section 4.5. For further diversification, we use a hashset to create a list of already visited solutions and, therefore, avoid visiting these solutions during the search process. We employ this strategy by guiding the search towards new solutions using the adaptive weight adjustment method described in Section 4.4. The algorithm stops after a specified number of iterations.

4.1. Solution representation

Each solution is represented by a permutation-based structure. Each *vehicle route* S_v is separated by a 0, and each order pickup and delivery location is represented by a positive integer value.

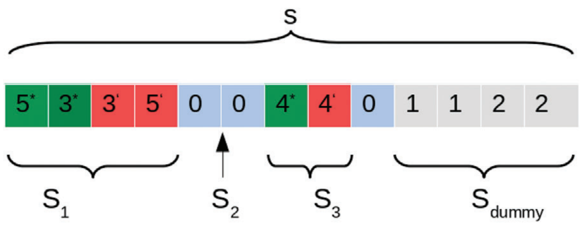


Fig. 3. Example of a solution representation of an instance with five orders and three vehicles. The dark green cells indicated with * represent a pickup node, while the red cells indicated with ' represent a delivery node. The 0 cells separate a vehicle route from other vehicle routes. The cells in the S_{dummy} section represent unserved orders.

Figure 3 illustrates an example of a solution s with five orders and three vehicles. A vehicle schedule represented by S_v contains the delivery and pickup schedule and their sequence starting from the left side of this representation. The pickup nodes are represented by green cells with *, while the delivery nodes are indicated by red cells with '. As an example, vehicle #3 picks up order #4 and delivers it right after pickup. Similarly, vehicle 1's schedule shows that it picks up order #5 and #3, and delivers order #3 and #5, respectively. S_2 is an example of an empty vehicle schedule, indicating that this vehicle is not used. The final part of the solution representation, S_{dummy}, shows a dummy vehicle's schedule containing those orders that are not served. A penalty cost is applied for the unserved orders. In this section, we continue to refer to s as a solution to the problem and S_v as a vehicle schedule.

4.2. Initial solution

To avoid premature convergence, we start with an initial solution s in which no orders are assigned to any of the vehicles, that is, all orders are assigned to the dummy vehicle S_{dummy}. This strategy enables us to start from a feasible solution and reduce the running time, as well as using this algorithm to solve any problem instances regardless of the initial solution.

4.3. Heuristics

In this section, we present the designed heuristics for our algorithm. Each of the following heuristics consists of one heuristic for removing some of the elements from a solution s and one heuristic for reinserting the removed elements to the solution. The first two heuristics *Swap* and *3-Exchange*, explained in Sections 4.3.1 and 4.3.2, are focused on diversification, escaping from local optimal solutions by shuffling a selected part of the current solution to generate new solutions regardless of their objective values. The *2-Opt* heuristic explained in Section 4.3.3 follows the intensification strategy to search for high-quality solutions in the search space. The *removal and insertion* heuristics described in Section 4.3.4 to 4.3.7 are inspired by the well-known removal and insertion heuristics in the literature for solving the PDP (Shaw, 1997; Ropke and Pisinger, 2006; Korsvik et al., 2011; Sze et al., 2016).

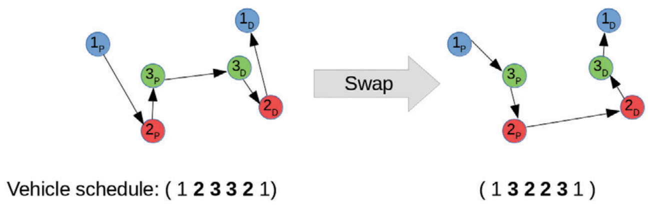


Fig. 4. A swap heuristic performed on a vehicle route with three orders. The numbered nodes indicate the locations of orders, and the letters P and D indicate pickup and delivery, respectively. 1_P is the pickup location of order 1, and 1_D indicates the delivery location of order 1 in the route. The bold numbers in the vehicle route are selected by the heuristic for swap

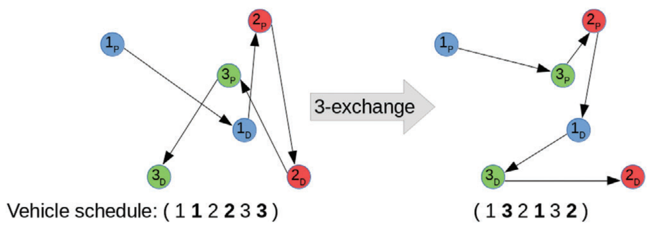


Fig. 5. A 3-exchange heuristic performed on a vehicle route with three orders. The numbered nodes indicate the locations of orders, and the letters P and D indicate pickup and delivery, respectively. 1_P is the pickup location of order 1. In this example, the bold numbers in the vehicle route are selected by the heuristic for exchange

4.3.1. Swap heuristic

The swap heuristic iteratively exchanges the pickup and delivery locations of two randomly selected orders in the route until it obtains a feasible solution, or it reaches a certain number of iterations. If the operator does not find a feasible solution, it returns the input solution without any changes to the solution. Figure 4 illustrates the swap heuristic.

4.3.2. 3-Exchange heuristic

The 3-exchange heuristic selects a vehicle route with at least two orders and iteratively performs an exchange of three randomly selected elements in the solution until a new feasible combination is obtained, or it reaches a certain number of iterations. If it does not find a feasible solution, it returns the input solution without any changes to the solution. In Fig. 5, the 3-exchange heuristic is applied to index positions 2, 4, and 6 in a vehicle route.

4.3.3. 2-Opt heuristic

The 2-opt heuristic selects a random vehicle with more than two orders. The vehicle route is divided into three segments and the order of visits in the middle segment is reversed. This procedure is performed for all the possible combinations of the three segment sizes. This operation is continued until no improvement can be obtained. Figure 6 illustrates one iteration of the 2-opt heuristic.

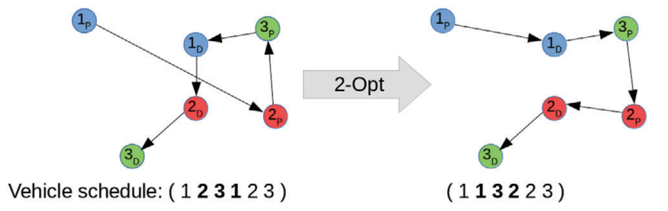


Fig. 6. One 2-opt heuristic operation performed on a vehicle route with three orders. The numbered nodes indicate the locations of orders, and the letters P and D indicate pickup and delivery locations, respectively. 1_P is the pickup location of order 1. The bold numbers indicate the nodes in the middle segment of the route. The order of visits in the middle segment is reversed in the new solution

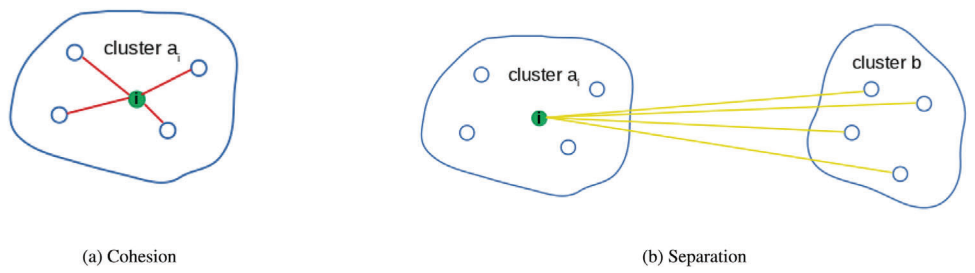


Fig. 7. The cohesion and separation factors are illustrated, respectively, by the red and yellow lines. The green node represents location i, and a represents its cluster. Cluster b is the cluster with the minimum average distance to i

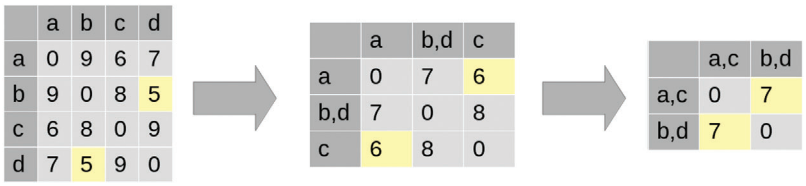
4.3.4. Random fit heuristic

This heuristic selects a random number of orders q, which are removed and reinserted to the solution. The value of q is between 2% and 10% of the number of orders in the problem. Once orders are removed from the solution, the heuristic selects a random vehicle v and inserts the removed orders to the vehicle route S_v randomly until a feasible vehicle route is obtained.

4.3.5. Cluster insertion and removal heuristic

Since some pickup and delivery locations are distributed in cities, and possibly they form some clusters, we design a heuristic which attempts to remove and reinsert clusters of orders rather a single order. This heuristic removes those orders with delivery and pickup locations from different clusters but served by the same vehicle, and then it bundles them and reinserts them to a single vehicle route. To this end, we use two clustering metrics to determine the sizes of clusters and effectively distribute the pickup and delivery locations to the clusters.

The silhouette coefficient. This coefficient (ν_i) is an effective metric to compare clusters of different sizes (Kaufman and Rousseeuw, 1990; Reddy and Vinzamuri, 2019). The Silhouette coefficient is calculated based on two factors, cohesion (ι_i) and separation (η_i) of a node i in a network of nodes. Figure 7 illustrates the cohesion and separation calculation factors for a node i as a part of cluster $a_i \in [1, 2, \ldots, k]$, where k is the number of clusters and a_i represents the index of the cluster for node i. The cohesion is the mean of the distances, d_{ij}, from i to all other nodes $j \in \kappa_{a_i}$ where κ_{a_i}



	a	b	c	d
a	0	9	6	7
b	9	0	8	5
c	6	8	0	9
d	7	5	9	0

	a	b,d	c
a	0	7	6
b,d	7	0	8
c	6	8	0

	a,c	b,d
a,c	0	7
b,d	7	0

Fig. 8. The tables represent the distance matrices for the pairs of nodes, a, b, c, and d. In the first table (left), d_{bd} marked in yellow is the shortest distance, and, therefore chosen to be clustered together. In the second table (middle), d_{ac} is the shortest distance and therefore forms a cluster. The third table (right) shows the final clustering results

is a set containing all nodes in cluster a_i. The separation η_i is the minimum of the mean of the distances, d_{ij}, from node i to all the nodes $j \in \kappa_{a_j}$ in cluster a_j, where $a_j \neq a_i$. The separation and cohesion factors can be calculated as follows:

$$\iota_i = \frac{1}{|\kappa_{a_i}| - 1} \sum_{j \in \kappa_{a_i}, i \neq j} d_{ij}, \tag{32}$$

$$\eta_i = \min_{\substack{b \in [1, 2, \ldots, k] \\ b \neq a_i}} \frac{1}{|\kappa_b|} \sum_{j \in \kappa_b} d_{ij}, \tag{33}$$

Then, the silhouette coefficient ν_i for node i is calculated as follows:

$$\nu_i = \frac{\eta_i - \iota_i}{\max \iota_i, \eta_i}, \quad \text{if } |\kappa_{a_i}| > 1, \tag{34}$$

$$\nu_i = 0, \quad \text{if } |\kappa_{a_i}| = 1,$$

The silhouette coefficient calculates the average ν_i for all $i \in N$. To decide the cluster for each pickup and delivery location, we choose the cluster with the smallest value of average ν_i.

Hierarchical single linkage clustering. To cluster the nodes when the number of clusters k is given, we use the hierarchical single linkage clustering algorithm (Reddy and Vinzamuri, 2019). Figure 8 illustrates the procedure of the algorithm for a set of nodes a, b, c, and d. In each iteration, we choose the pair of nodes (locations) with the smallest distance, d_{ij}, and group these nodes into one cluster. The new distance matrix contains the shortest distance from the merged nodes to any other nodes. The grouping procedure continues until k clusters are obtained.

Having defined these two metrics, we can compare clusters of different sizes and also perform the clustering of a given set of pickup and delivery locations for a given number of clusters. To use the clustering approach, we run a preprocessing algorithm that assigns each location $i \in N$ to their corresponding clusters, where k is decided by calculating the average silhouette coefficient, ν_i, for all $i \in N^P$ and keeping the clusters with the lowest value of average ν_i. Therefore, each location $i \in N$ is assigned to a cluster index a_i and is assigned to cluster set κ_{a_i}.

Algorithm 2 shows the cluster removal and insertion heuristic. The algorithm selects a number of orders q for removal from the current solution and sorts all the orders in the given solution s

Algorithm 2. Cluster heuristic

```
1  Input solution $s$, random selection degree $p$
2  Select number of orders to reinsert, $q$
3  Add orders to array $A$ in an ascending order based on cluster value $\sigma_i$
4  Generate empty order set $I$
5  repeat
6      Choose a random number $y$ between $[0, 1]$
7      Remove the order in position $y^p$ in $A$ from $s$
8      Add removed order to $I$
9  until $|I| = q$;
10 repeat
11     Sort $I$ descending based on best possible $\sigma_i$
12     Select the first order from $I$ and remove it from $I$
13     Insert the selected order in its best feasible position in solution $s$
14 until $|I| = 0$;
15 Output $s$
```

according to the cluster value σ_i, where i is the pickup location of the order. The *cluster value* of an order is calculated based on the visited locations by the vehicle, that is, those orders that are grouped together to be visited by the same vehicle. The idea of the cluster value is to evaluate the possibility of clustering an order i with other orders on a vehicle route. This measure (σ_i) is calculated by

$$\sigma_i = \frac{|\ell_{iv}| + |\ell_{(i+n),v}|}{2(|S_v| - 2)} \forall i \in N^P. \tag{35}$$

In this equation, set ℓ_{iv} contains all the locations visited by vehicle v from the same cluster κ_i as the pickup location i, excluding i and/or $i + n$. Set $\ell_{(i+n),v}$ includes all the locations visited by vehicle v from the same cluster κ_{i+n} as the delivery location $i + n$, excluding $i + n$ and/or i. Set S_v is the vehicle route of vehicle v, that is, $|S_v|$ is the number of locations visited by vehicle v, and, therefore, $2(|S_v| - 2)$ is the maximum possible common cluster locations that $|\ell_{iv}| + |\ell_{(i+n),v}|$ can contain. We divide this by $2(|S_v| - 2)$ to be able to compare the cluster value with orders on other vehicle routes. The resulting σ_i, therefore, yields a value between [0,1] where a high value indicates that orders are highly clustered, that is, the locations of other orders on the same vehicle route are within the same cluster. A value close to 0 indicates that order locations belong to different clusters.

Algorithm 2 uses the cluster value σ_i to remove the customer orders with the lowest rank. We choose to remove the order in position y^p in the ascending sorted list of σ_i, where y is a random number between [0,1] (Ropke and Pisinger, 2006). To reinsert the orders, the algorithm sorts the removed orders based on their σ_i values and inserts the orders with the highest cluster value σ_i in the best possible position in s that yields the least incremental costs. The insertion cost is dynamically recalculated after reinsertion of each order. The procedure continues until all the selected orders are reinserted to the routes (including the dummy route), keeping routes feasible.

Algorithm 3. Greedy heuristic

```
1  Input solution $s$, random selection degree $p$
2  Select number of orders to reinsert, $q$
3  Add orders $i \in N^P$ in array $A$ in descending order based on cost $C_i^S$
4  Generate empty order set $I$
5  repeat
6      Choose a random number $y$ between $[0, 1]$
7      Remove the order in position $y^p$ in $A$ from $s$
8      Add the removed order to $I$
9  until $|I| = q$;
10 repeat
11     Sort $I$ based on each order's minimum increase in objective value, $c_i$
12     Insert the first order $i$ from $I$ in its best feasible position in solution $s$
13     remove order $i$ from $I$
14 until $|I| = 0$;
15 Output $s$
```

4.3.6. Greedy heuristic

In this heuristic, those customer orders with the highest cost $C_i^S \; \forall \; i \in N_i^P$ are selected and reinserted in the solution where the minimum incremental cost of insertion is obtained. C_i^S is the increase in a vehicle's route cost when it includes order *i* compared to the route without this order. Algorithm 3 shows the pseudocode of the algorithm. To remove the orders from the solution, we sort the customer orders in descending order of their cost C_i^S values. The cost is calculated as $C_i^S = f(S_v) - f_{-i}(S_v)$ for a given order *i* served by vehicle *v*, where $f(S_v)$ is the objective value (total costs) of vehicle route S_v. In this equation, $-i$ indicates that we calculate the cost of vehicle route S_v without order *i*. We choose to remove the order in position y^p. To reinsert an order, we sort the removed orders based on their minimum increase in the objective value, c_i, which is calculated as $c_i = \min_{v \in V}(\Delta f_{iv})$. Here, Δf_{iv} represents the difference in the objective function value f_{iv} by inserting order *i* in position *v* with the lowest increase in the objective function value.

4.3.7. Similar regret heuristic

This heuristic identifies similar customer orders and replaces them to generate new solutions. We define a *relatedness factor* r_{ij}, which measures the relatedness of order *i* to order *j*. The relatedness factor depends on the following properties: (a) travel distance, (b) order weight, (c) whether the same vehicles can be used to serve each order, (d) whether both orders *i* and *j* belong to the same factory, and (e) overlapping time window. The relatedness factor is calculated by the following equation:

$$r_{ij} = \psi(D_{ij} + D_{(i+n)(j+n)}) + \omega|Q_i - Q_j| + \phi\left(1 - \frac{|V_i \cap V_j|}{\max(|V_i|, |V_j|)}\right) + \tau G_{ij} + \chi(U_{ij} + U_{(i+n)(j+n)}). \quad (36)$$

Since each term is normalized between 0 and 1, the relatedness measure r_{ij} has a range $0 \leq r_{ij} \leq 2\psi + \omega + \phi + \tau + \chi$. A lower value of r_{ij} represents that two orders i and j are more similar. In Equation (36), D_{ij} represents the distance between nodes i and j, and Q_i indicates the weight of shipment of order i. Set V_i contains the vehicles that can serve order i. Parameter G_{ij} is a binary parameter equal to 1 if the factories of orders i and j are different, while it is equal to 0 if they are delivered to the same factory. Parameter U_{ij} is the portion of overlapping time windows at the pickup and delivery locations of orders i and j divided by the total span of the two time window sets. A value of 0 indicates no overlapping time windows, while a value of 1 means identical time windows. This can be calculated as follows:

$$U_{ij} = 1 - \frac{T_{ij}^{O}}{T_{ij}^{A} - T_{ij}^{NO}}. \tag{37}$$

$\overline{T_{ip}}$ and $\underline{T_{ip}}$ are the upper and lower limits on the time windows, respectively. Parameter T_{ij}^{O} consists of all the time windows where order i overlaps with the time windows of order j. This is represented as follows:

$$T_{ij}^{O} = \sum_{\substack{p \in \pi_i \\ o \in \pi_j \\ \underline{T_{ip}} \leq \overline{T_{jo}} \\ \underline{T_{jo}} \leq \overline{T_{ip}}}} (\min(\overline{T_{ip}}, \overline{T_{jo}}) - \max(\underline{T_{ip}}, \underline{T_{jo}})), \tag{38}$$

where T_{ij}^{A} represents the total span of the time window sets of locations i and j. It starts from the first time window lower bound and ends at the last time window upper bound of these locations. This can be formulated as

$$T_{ij}^{A} = \max\,(\max_{p \in \pi_i} \overline{T_{ip}}, \max_{o \in \pi_j} \overline{T_{jo}}) - \min\,(\min_{p \in \pi_i} \underline{T_{ip}}, \min_{o \in \pi_j} \underline{T_{jo}}). \tag{39}$$

Parameter T_{ij}^{NO} is the opposite of the above factor T_{ij}^{O} and represents when orders i and j do not have a time window. This can be explained by night time when factories are not open. It can be formulated as follows:

$$T_{ij}^{NO} = \sum_{\substack{p \in \pi_i \\ o \in \pi : q_j \\ \underline{T_{ip}} \geq \overline{T_{j(o-1)}} \\ \underline{T_{jo}} \geq \overline{T_{i(p-1)}}}} (\min(\underline{T_{ip}}, \underline{T_{jo}}) - \max(\overline{T_{i(p-1)}}, \overline{T_{j(o-1)}})). \tag{40}$$

Figure 9 illustrates how T_{ij}^{O}, T_{ij}^{NO}, T_{ij}^{A}, and $T_{ij}^{A} - T_{ij}^{NO}$ are calculated for two locations i and j. For Equation (36), the following values are chosen: $\psi = 0.7$, $\omega = 1.0$, $\phi = 0.8$, $\tau = 0.3$, and $\chi = 0.3$.

Algorithm 4 shows the pseudo-code of the similar regret heuristic. It starts by removing a random order i from the solution and adding it to a set I. Then, it creates an array and adds all orders $j \notin I$ to this array and sorts the array in ascending order based on r_{ij} values. Thereafter, it selects a customer order with the largest relatedness value and removes it from s, and adds it to I. This

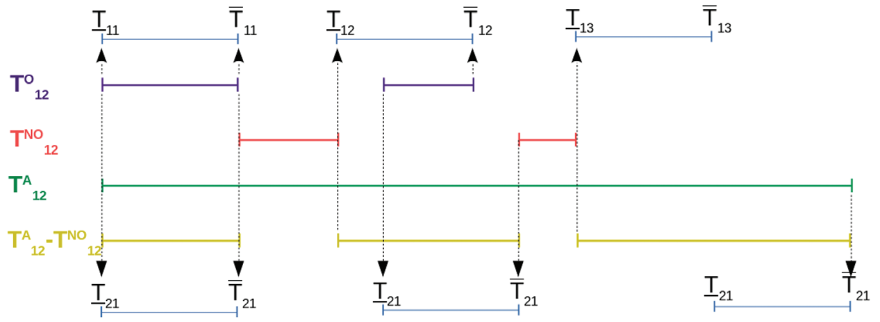


Fig. 9. This figure shows time window sets ($[\underline{T_{1p}}, \overline{T_{1p}}]$ and $[\underline{T_{2p}}, \overline{T_{2p}}]$) at two locations 1 and 2. The purple T^{O}_{12} is the overlapping time windows. The red T^{NO}_{12} represents the time when no location has a time window. The green T^{A}_{12} represents the whole span of both time windows, and the yellow $T^{A}_{12} - T^{NO}_{12}$ represents the intersection of the two time window sets

Algorithm 4. Similar regret heuristic

```
1  Input solution $s$, $p$
2  Select the number of orders to reinsert, $q$
3  Select a random order $i$ from $s$
4  Add $i$ to order set $I$
5  repeat
6      Add all orders $j \notin I$ in array $A$ ascending based on relatedness $r_{ij}$
7      Choose a random number $y$ between $[0, 1]$
8      Remove the order in position $y^p$ in $A$ from $s$
9      Add removed order to $I$
10 until $|I| = q$;
11 repeat
12     sort $I$ ascending based on regret value $c_i^*$
13     insert the first order $z$ from $I$ in its best possible position in $S_v$
14     update $s$ based on $S_v$
15     remove order $z$ from $I$
16 until $|I| = 0$;
17 return $s$
```

procedure continues until q orders are added to I. The insertion part of this heuristic improves the insertion algorithm from the greedy heuristic (see Section 4.3.6) by calculating a regret value, c_i^*. The regret value estimates the cost of not inserting an order i to the solution. We let S_{i1}, S_{i2}, and S_{i3} represent the vehicle schedules with, respectively, the first, second, and third lowest insertion cost for an order $i \in N^P$. That means $\Delta f_{S_{i1}} \leq \Delta f_{S_{i2}} \leq \Delta f_{S_{i3}}$, where Δ represents the difference in objective value $f_{S_{ik}}$ by inserting order i in its $k \in [1, 2, 3]$ best vehicle schedules. We define the regret value as follows:

$$c_i^* = \Delta f_{S_{i2}} - \Delta f_{S_{i1}} + \Delta f_{S_{i3}} - \Delta f_{S_{i2}}. \tag{41}$$

Equation (41) represents that c_i^* is the difference in inserting order i in its best position and its second best position plus the difference in inserting it in its second best position and its third best position. In each iteration, the heuristic chooses to insert the order with the largest c_i^* value. The selected order is inserted in its best possible position, and ties are broken by choosing the order with the lowest insertion cost c_i defined in Section 4.3.6.

4.4. Adaptive weight adjustment

We have defined seven heuristics to effectively search the solution space. To select a heuristic in each iteration of Algorithm 1, we use the *roulette wheel strategy*. If we represent each heuristic by an index $h \in \{1, 2, \dots, m\}$, where m is the number of heuristics, we select heuristic h with a probability p_h calculated based on each heuristic's weight w_h as follows:

$$p_h = \frac{w_h}{\sum_{g=1}^{m} w_g}. \tag{42}$$

We choose an adaptive weight adjustment strategy (see Section 4.4) to calculate the weights of the heuristics and automatically update the above weights. This strategy updates the weight of each heuristic based on the performance of the heuristic using a scoring model. A better performance by a heuristic leads to a higher score for the heuristic. To this end, after a prespecified number of iterations, so-called a segment, the weight of each heuristic is updated based on its performance. At the beginning of the search, all the heuristics are given equal weights, resulting in an equal probability to select each heuristic. Throughout a segment, each heuristic is rewarded points based on its performance, which depends on whether it finds a new global best solution, a new better local solution, or a new solution. The sum of all the scores received during a segment is used in Equation (43) to update the heuristic's weight. If we let w_{hg} be the weight of heuristic h in segment g as well as π_h and λ_h be the score and number of times that heuristic h was run in the current segment, respectively, the updated weight is calculated as follows:

$$w_{hg} = w_{h(g-1)}\Upsilon + (1 - \Upsilon)\frac{\pi_h}{\lambda_h}, \tag{43}$$

where Υ represents a weight adjustment coefficient, which is set to 80%. We enforce a lower limit on the weight of each heuristic to ensure that every heuristic is selected at least for a certain number of times in each segment. Figure 10 shows the heuristics' weights p_h of five heuristics in one of the experiments.

4.5. Acceptance criteria and stopping condition

To prevent early convergence to local optimal, the generated solution in each iteration of the algorithm is accepted/declined based on the acceptance criteria of the simulated annealing algorithm (Kirkpatrick, 1984). Using this method, if the new generated solution is better than the current

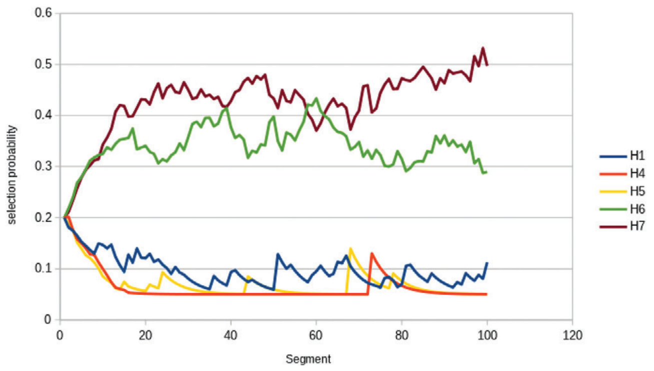


Fig. 10. The adjustment of the heuristic's weight probability p_h using our model. The x-axis represents a segment and the y-axis shows the weights for selecting each of the heuristics

solution with respect to the objective function value, it is accepted and set as the current solution. However, if the new generated solution is a nonimproving solution compared to the current solution, it accepts it with the probability $e^{-|f-f_{new}|/T}$, where $T < 0$ is the temperature, f is the objective value of the current solution, and f_{new} is the objective value of the new solution. The *cooling schedule*, that is, the method used to update temperature T, is inspired by Crama and Schyns (2003). This method sets a certain starting temperature T_{start} which decreases per iteration with a certain cooling rate of $0 < c < 1$. To determine T_{start} based on the problem instance, we run 100 iterations of the algorithm with a fixed acceptance rate of $a = 0.8$ on each instance and calculate the average objective function value ($f^T_{average}$) of all the accepted nonimproving solutions based on the described acceptance criteria. We use this value to calculate the starting temperature by (44):

$$T_{start} = \frac{f^T_{average}}{\ln(a)}. \qquad (44)$$

The algorithm stops after a prespecified number of iterations (10,000 iterations).

4.6. *Wild escape algorithm*

To enhance diversification of the algorithm and prevent local convergence, we use the wild escape strategy if the algorithm cannot improve the best found solution after 500 iterations. Using this strategy, the algorithm accepts any new found solution for 20 iterations regardless of its objective function value to change the solution search neighborhood. We use three heuristics random fit (see Section 4.3.4), 3-exchange (see Section 4.3.2), and swap (see Section 4.3.1) to generate the new solution in this strategy because these three heuristics do not target a specific improvement to the solution, and, rather, they generate new solutions randomly. The pseudocode of the algorithm is described in Algorithm 5.

Algorithm 5. Wild escape algorithm

```
1 Input solution $s$, $s_{best}$, set of heuristics $\mathbb{H}$
2 repeat
3     Choose a random heuristic $h$ from $\mathbb{H}$
4     Apply $h$ to $s$
5     if $f(s) < f(s_{best})$ then
6         $s_{best} \leftarrow s$
7     end
8 until stop condition met;
9 Return $s$
```

5. Numerical results

This section presents the numerical results of our mathematical model and algorithm on several problem instances. We explain the experimental setup for our numerical analysis, problem instances, heuristic selection procedure, and analysis of the algorithm for solving large-scale problem instances. The experiments are run on a 64-bit computer with a 1.8 GHz quad-core i7 processor and 16GB RAM. The mathematical model (see Section 3.1) is solved using the Gurobi solver version 8.1, and our proposed algorithm (see Section 4) is implemented in Java SE 11.0.4.

5.1. Problem instances

The problem instances are generated based on the provided data from a real case study. The data include the number of orders $|N|$, locations $|L|$, factories $|F|$, and number of vehicles $|V|$ which is between $|N|/2 \leq |V| \leq 2/3|N|$. The data also include information about the cost structure, capacity of the vehicles and their compatibility, such as cooling possibilities, special equipment for transport (e.g. fragile or hazardous goods), or special equipment required at the pickup or delivery location, for example, a crane to unload goods. We refer to these capabilities as *vehicle requirements*.

Orders are assigned to pickup and delivery locations randomly, and those orders that are assigned to the same location are given the same stop L_s. Each delivery location is independently assigned to a factory at random N_f. We assign the necessary vehicle requirement to 5% of pairs of order-location. This affects which vehicle is capable of picking up an order, N_v^P and N_v^D. We consider three different vehicle types, small, medium, and large. Large vehicles have low speed, with $Q_v^{kg} = 24k$ and $Q_v^{vol} = 102$, and they are compatible with all locations and orders. Medium vehicles have medium speed vehicles, with $Q_v^{kg} = 18k$ and $Q_v^{vol} = 71$, and they are compatible with all locations, except for those orders that require vehicle requirements. Small vehicles have high-speed vehicles, with $Q_v^{kg} = 12k$ and $Q_v^{vol} = 55$, and they are not compatible with any vehicle requirements. The Euclidean distance measure is used for each pair of location to calculate d_{ij} and the travel time T_{ijv} is scaled with 60% of the travel distance, added with a random variation of $\pm 10\%$ of the travel distance. The travel time also depends on the speed of the vehicle. That is, slow-speed vehicles have

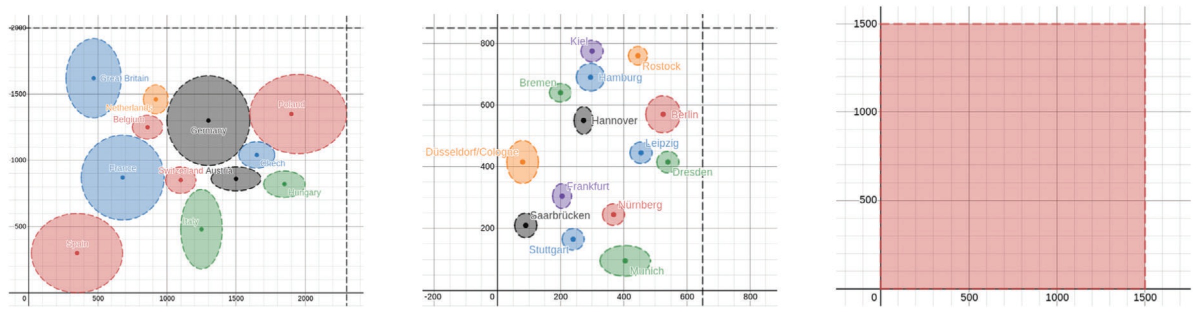


Fig. 11. Areas used for pickup and delivery locations when generating the instances

a 5% increase in travel time, while the medium-speed vehicles have a 2.5% increase, and the fast-speed vehicles have no increase in travel time. The cost matrices $C^{km}_{v\alpha\beta}$, $C^{kg}_{v\alpha\beta}$, $C^{fix}_{v\alpha\beta}$ obtained from the data are scaled based on the size of the instance and capacity of the vehicles. The cost of not serving an order C_i is set to a minimum lower bound (the most expensive transport) and scaled based on the weight, volume, and travel distance of the order. Time windows $[\underline{T_{ip}}, \overline{T_{ip}}]$ are generated with one to two time windows per day for each location, and three to seven days per week.

To generate the problem instances, we consider three areas for the geographical locations of customers. These three sets include Germany, Europe, and a uniformly distributed squared area. The two generated maps for Germany and Europe contain real-scale approximations of geographical points shown by an elliptic uniformly distributed area to represent a country or a city. To generate an instance, locations are uniformly and randomly chosen in the defined regions shown in Fig. 11.

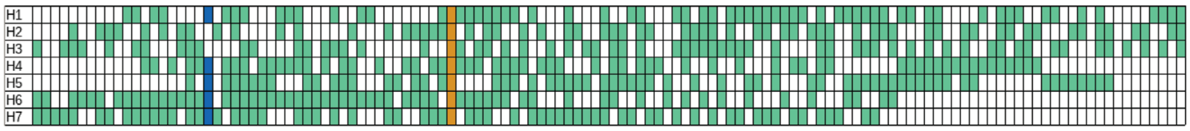
For our numerical analysis, we generate five problem sets. Each set consists of five instances of varying sizes as the following:

- Sets 1 and 2 are generated based on the European map from Fig. 11(a).
- Sets 3 and 4 are generated based on the German map from Fig. 11(b).
- Set 5 is generated based on the uniform distribution from Fig. 11(c).

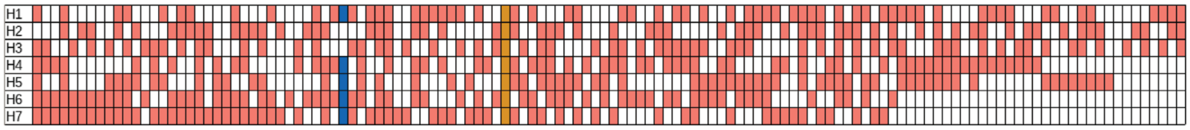
These five sets of instances contain 4, 12, 35, 80, and 150 orders, 3, 7, 20, 45, and 80 vehicles, and 7, 9, 22, 45, and 85 locations, respectively.

5.2. *Analysis of heuristics*

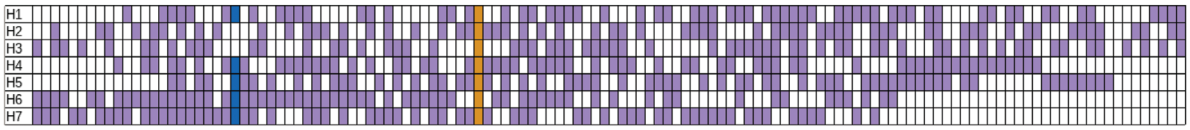
In this section, we conduct a comprehensive analysis of the performances of the heuristics. For the experiments we use five instances, each consists of 80 orders. The algorithm is applied to solve each problem instance 10 times, and the best and average objective function values of these runs, as well as the run-time, are reported. We analyze the performance of each heuristic individually, and in combination with other heuristics using statistical experiments, analysis of variance III, (ANOVA-III), multiple linear regression, and pairwise t-tests. For all statistical experiments, we use a 95% confidence interval. A final combination of heuristics is chosen to conduct the final experiments and



(a) By average improvement



(b) By best improvement



(c) By best+average improvement

Fig. 12. Ranking of improvements over the initial solution. Highlighted squares indicate the use of a heuristic. The combinations with better results appear on the left of this figure. Blue highlighted squares show the final composition from Section 5.4, and the yellow highlighted squares show the use of all heuristics

compare with the results of the mathematical model. For the sake of brevity, we use the following terms to refer to the defined heuristics:

- H1-swap: the swap heuristic described in Section 4.3.1.
- H2-exchange: the exchange heuristic from Section 4.3.2.
- H3-2opt: the 2-opt heuristic described in Section 4.3.3.
- H4-random: the random fit heuristic from Section 4.3.4.
- H5-cluster: the clustering heuristic described in Section 4.3.5.
- H6-greedy: the Greedy heuristic described in Section 4.3.6.
- H7-similar: similar regret heuristic described in Section 4.3.7.

5.2.1. Initial evaluations of heuristics

To evaluate the performances of the heuristics, we run the algorithm on a total of $2^7 = 128$ combinations of the heuristics and use the best and average percentage of improvements of 10 runs compared to the initial solution. First, we rank the combinations based on the best and average of improvements. We also use the ANOVA and regression analysis to assess the significance of the impact of the heuristic combinations. Finally, we run t-tests to evaluate whether certain heuristics in combination with other heuristics have positive impacts on the results.

Figure 12 shows the ranking of the heuristic combinations. A colored square in each row indicates that the corresponding heuristic is used. Those combinations that appear on the left side of the figure yield higher improvements over the initial solution. Figures 12(a) and 12(b) show the combination of heuristics ranked based on the average and best improvement over the initial solution for each heuristic combination. Finally, Fig. 12(c) shows the ranking considering both the average and best improvements which lead to obtain the overall ranking. The blue highlighted squares indicate the final composition from Section 5.4, and the yellow highlighted squares show the combination in which all the heuristics are used.

The ranking provides an overview of the performances of the heuristics. This ranking shows that using all the designed heuristics is not necessarily effective. The ranking also shows that heuristics H6-greedy and H7-similar are often included in the combinations which yield better performances.

The ANOVA (III) and multiple linear regression analysis help evaluate which heuristics have significant positive impacts on the performance of the algorithm. Table A1 shows the results of the ANOVA (III) analysis. The ANOVA analyses for the average improvement and best improvement of 10 runs are shown in Tables A1(a) and A1(b), respectively. In Table A2, we perform a multiple linear regression analysis of the same results used in the ANOVA analysis. The multiple linear regression analysis elaborates on whether a heuristic positively influences the effectiveness of the algorithm based on R^2.

The results of the ANOVA (III) analysis from Tables A1(a) and A1(b) reveal that four heuristics H4–H7 have significant impacts on the results with a 95% confidence interval. However, Tables A2(a) and A2(b) indicate that only three heuristics H4-random, H6-greedy, and H7-similar have positive coefficients, that is a positive significant influence, for both the average and best improvement. Thus, we can conclude that H4-random, H6-greedy, and H7-similar effectively search the solution space. We refer to these heuristics as the *significant* heuristics. H5-cluster negatively influences the algorithm's results; therefore, this leads us to conclude that this heuristic either performs poorly alone or has a negative influence on the algorithm.

According to the performances of heuristics H1-swap, H2-exchange, and H3-2opt, we can conclude that they do not contribute significantly to the results individually, that is, an algorithm containing only these heuristics do not yield a significant positive impact on the results with a 95% confidence interval. However, since they can positively affect the results if they are used in combination with the significant heuristics; therefore, we need further assessment of these heuristics. We refer to the set of heuristics H1-swap, H2-exchange, H3-2opt, and H5-cluster as the undecided group, or "G-heuristics" for further evaluations.

5.2.2. Further evaluations of heuristics

To evaluate the performances of the G-heuristics, we include those combinations that use one or multiple heuristics of the G-heuristics. To this end, we conduct this analysis in two parts. First, we evaluate if the G-heuristics, as a group, have a positive or negative influence on the result. We use 2-sample *t*-tests to compare the mean of the population where we use the G-heuristics along with one or more of the significant heuristics, to the mean of the population without using the G-heuristics. In the second part, we analyze if using the G-heuristics in combination with the significant heuristics have impact on the results. For this analysis, we use the ANOVA (III) statistical analysis and multilinear regression model.

Using the *t*-test, we compare the mean of the results obtained by the combinations which include one or multiple G-heuristics, with the results of those combinations without any G-heuristics. The results of the *t*-test are summarized in Table A3. In this table, P_N represents the results of those combinations without any G-heuristics, while P_H represents the results of the combinations with some of the G-heuristics. The results show that the null hypothesis cannot be rejected, that is, the effect of combining G-heuristics with other heuristics is not significant with a 95% confidence level. This is true for both the average and best improvement results for the right, left, and both-tails analyses. Therefore, further evaluations are required to make the final conclusion.

We continue the evaluations of the G-heuristics using different combinations of the G-heuristics and significant heuristics. The results of the ANOVA (III) analysis are summarized in Tables A4 and A5. In these tables, G+H4, for example, contains all the observations where at least one of the G-heuristics is used in combination with the H4-random heuristic. The results in Table A4(a) indicate that only two of the defined combinations have considerable impacts on the results with a 95% confidence level. Using the G-heuristics combined with heuristics H6-greedy and H7-similar, as well as its combination with heuristic H4-random have significant impacts on the average improvement results. Table A5(a) shows that such combinations also obtain a positive coefficient value. Since $R^2 = 0.95$ is considerably high, this model explains the results very well, and therefore, this supports the use of G-heuristics in combination with other heuristics in regard to the average improvement.

The same combinations show the positive significance in regard to the best improvement (Tables A4(b) and A5(b)). It can be seen that the combinations with the highest positive estimated coefficients still include heuristics H6-greedy and H7-similar, or heuristics H6-greedy, H7-similar, and H4-random. The increased $R^2 = 0.967$ represents that this model explains the results regarding the best improvements very well. This indicates that the combination including H6-greedy, H7-similar, and H4-random consistently contributes to the improvement of results. Therefore, this supports our conclusions from Section 5.2.1 on the significance of heuristics H6-greedy, H7-similar, and H4-random. This also indicates that there are some combinations of the G-heuristics and the significant heuristics that have positive impacts on the result with respect to both the average and best improvement results. In the next section, we conduct more evaluations on individual heuristics.

5.2.3. *Evaluation of individual heuristics*

We aim to determine which of the G-heuristics have a positive, or negative impact on the results using a pairwise *t*-test. This analysis assesses if a G-heuristic significantly improves or decreases the best and average improvements. We test if the mean of the results from combining the G-heuristics with the significant heuristics is significantly different from the results of combinations without any of the G-heuristics. Similar to the analysis in Section 5.2.2, we remove the combinations in which only G-heuristics are used due to their weak performance.

The results of the pairwise *t*-tests are summarized in Table A6. We use $P_{NA}^{H_i}$ to refer to the combinations without heuristic H_i for the average improvement, and parameter $P_{HB}^{H_i}$ as the combinations including heuristic H_i for the best improvement. We conduct this test for all the four heuristics in the set of G-heuristics, which includes H_1, H_2, H_3, and H_5.

For heuristic H1-swap, the means of both the best and average improvements are not significantly different with a 95% confidence interval. Regarding the results for heuristic H2-exchange, we reject the null hypothesis that the means are equal for the average and best improvements. The left tail alternative hypothesis is accepted, indicating that the mean of $P_{HA}^{H_2}$ and $P_{HB}^{H_2}$ is lower than $P_{NA}^{H_2}$ and $P_{NB}^{H_2}$, while the right tail alternative hypothesis is rejected for both the best and average improvements. The results of H3-2opt show that the null hypothesis is rejected, that is, the mean improvements by $P_{HA}^{H_3}$ is not equal to the mean results of $P_{NA}^{H_3}$, whereas it is accepted for $P_{HB}^{H_3}$ and

Table 4
Analysis of the wild escape heuristic. Best and average results obtained by the algorithm and its run-time when all the heuristics are used for the three defined settings

				Average objective			Best objective			Run-time (second)		
#Ord	#Veh	#Loc	Initial objective	No escape	Random reset	Wild escape	No escape	Random reset	Wild escape	No escape	Random reset	Wild escape
4	3	7	609,680.3	3,444.7	3,444.7	3,444.7	3,444.7	3,444.7	3,444.7	0.18	0.20	0.20
12	7	9	1,023,745.5	149,692.6	154,832.9	149,692.4	149,692.4	149,692.4	149,692.4	0.39	0.51	0.46
35	20	22	2,682,067.9	10,639.1	10,849.5	10,350.9	10,404.9	10,358.6	10,025.1	2.31	1.61	2.49
80	45	45	6,422,128.6	22,262.2	25,802.9	21,377.4	20,761.2	21,777.4	20,831.3	15.89	8.05	14.97
150	80	85	12,059,380.3	40,667.2	38,313.0	35,705.7	34,316.0	34,345.0	34,282.3	88.21	48.92	77.78

$P^{H_3}_{NB}$, respectively. The left tail alternative hypothesis is accepted, indicating that the mean of $P^{H_3}_{HA}$ is lower than $P^{H_3}_{NA}$ with a 95% confidence level. For H5-cluster, the null hypothesis is rejected for $P^{H_5}_{HB}$ and $P^{H_5}_{NB}$, while it is accepted for $P^{H_5}_{HA}$ and $P^{H_5}_{NA}$. The alternative hypothesis of the right tail of the best improvement is accepted, that is, the mean is significantly higher in $P^{H_5}_{HB}$ than $P^{H_5}_{NB}$.

The results summarized above demonstrate that using H1-swap has no significant effect on the performance of the algorithm. Also, H2-exchange and H3-2opt significantly deteriorate the average improvements, whereas H2-exchange decreases the improvements for the best obtained solutions. Therefore, it is not beneficial to include these heuristics in the algorithm. Finally, H5-cluster does not affect the average improvements; however, it positively affects the best improvement, indicating that it could be beneficial to include this heuristic in the final combination of the heuristics.

5.3. *Evaluation of wild escape algorithm*

To evaluate the implementation of the wild escape heuristic, we use the algorithm to solve problem Set 1 in three different settings: the algorithm without the wild escape heuristic (*no escape*), the algorithm with complete random restart (*random reset*), and the algorithm with the wild escape heuristic (*wild escape*). The results for these three settings are summarized in Table 4, which includes the average of objective values during each of the 10 runs, the best solution found, and the average run-time. On average, the wild escape heuristic outperforms both the *random reset* and *no escape* strategies. The *wild escape* strategy obtains the best results with respect to both the average and best found solutions in almost all the problem instances. Due to the effectiveness of the wild escape algorithm and comparable run-time, we include this heuristic in the final composition of the algorithm.

5.4. *Final experimental results and analysis*

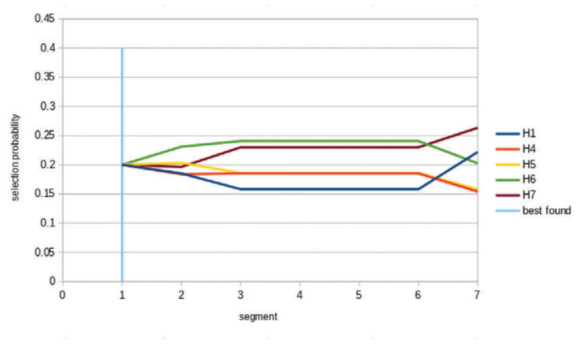
Based on the analyses in Section 5.2.1 to 5.2.3, the finalized algorithm uses the wild escape algorithm along with heuristics H1-swap, H4-random, H5-cluster, H6-greedy, and H7-similar.

We apply our algorithm to solve 25 problem instances described in Section 5.1. The numerical results by our algorithm and the mathematical model are summarized in Table 5. We enforce a

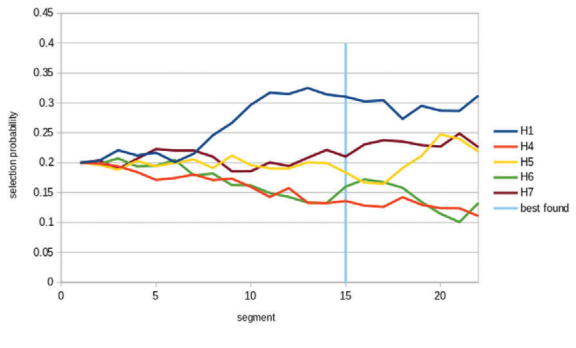
Table 5
Numerical results of the algorithm and mathematical model on the generated problem instances. *delta* is calculated as (solution objective MM - Algorithm best objective)/solution objective MM

				Mathematical model			Algorithm			
Inst Set	#Ord	#Veh	#Loc	Solution objective	Optimality gap	Run time (seconds)	Average objective	Best objective	Average run time	Delta
Set 1	4	3	7	3,444.7	0.00%	0.1	3,444.7	3,444.7	0.1	0.00%
	12	7	9	149,692.3	0.00%	9,963.7	149,692.4	149,692.3	0.5	0.00%
	35	20	22	2,091,776.5	99.91%	10,000.2	10,323.2	9,997.9	2.8	99.52%
	80	45	45	6,422,128.6	99.99%	10,000.0	21,170.5	20,911.5	21.1	99.67%
	150	80	85	NA	NA	NA	34,479.1	32,798.0	100.8	NA
Set 2	4	3	7	2,501.0	0.00%	0.1	2,501.0	2,501.0	0.1	0.00%
	12	7	9	5,987.8	0.00%	1,041.8	5,987.8	5,987.8	0.6	0.00%
	35	20	22	1,985,165.5	99.90%	10,000.3	14,382.4	14,272.1	2.6	99.28%
	80	45	45	6,809,899.5	99.99%	10,000.0	25,736.6	24,760.8	16.8	99.64%
	150	80	85	NA	NA	NA	36,927.2	35,932.1	112.1	NA
Set 3	4	3	7	1,404.0	0.00%	0.3	1,404.0	1,404.0	0.1	0.00%
	12	7	9	5,862.5	33.16%	10,000.0	5,862.5	5,862.5	0.9	0.00%
	35	20	22	679,594.8	99.71%	10,000.4	6,334.7	6,267.7	3.7	99.08%
	80	45	45	5,748,613.6	99.99%	10,000.5	12,609.0	12,347.6	32.2	99.79%
	150	80	85	NA	NA	NA	19,771.9	19,149.8	126.1	NA
Set 4	4	3	7	1,696.1	0.00%	0.7	1,696.1	1,696.1	0.1	0.00%
	12	7	9	3,285.2	21.45%	10,000.0	3,109.6	3,109.6	1.5	5.35%
	35	20	22	547,881.6	99.72%	10,000.2	4,652.8	4,494.5	5.1	99.18%
	80	45	45	6,201,301.5	99.99%	10,000.1	15,540.0	15,290.6	21.4	99.75%
	150	80	85	NA	NA	NA	22,508.6	22,252.6	103.4	NA
Set 5	4	3	7	5,154.9	0.00%	0.4	5,154.9	5,154.9	0.1	0.00%
	12	7	9	3,716.2	22.93%	10,000.2	3,716.2	3,716.2	0.6	0.00%
	35	20	22	1,757,079.6	99.90%	10,000.2	13,138.9	12,944.2	2.1	99.26%
	80	45	45	5,909,616.9	99.99%	10,000.0	24,034.0	23,855.5	9.1	99.60%
	150	80	85	NA	NA	NA	41,343.4	39,847.0	82.6	NA

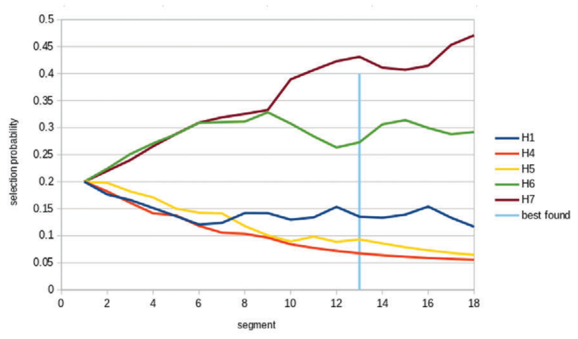
maximum processing time limit of 10,000 seconds for the experiments using the mathematical model. If the solver cannot find a feasible solution within this time limit, we report NA in column run-time. The results show that the performance of the mathematical model deteriorates as the size of the instance increases. Although the mathematical model can find the optimal solution to the smallest instances in all the five problem sets, it cannot find the optimal solution for set 3 and set 5. The optimality gap is considerably large for the third and fourth instances (with 35 orders and 80 orders) in all the problem sets, where the algorithm reaches the maximum time limit of 10,000 seconds. The comparison of the results of the algorithm with the mathematical model shows that the algorithm can efficiently obtain optimal solutions to small problem instances with 4 and 12 orders. For the larger problem instances (orders $\geq$ 35), the algorithm outperforms the mathematical model as the solver cannot find a near-optimal solution within the time limit. The delta value in Table 5 is higher than 99% for all of these problem instances, indicating that our algorithm is significantly



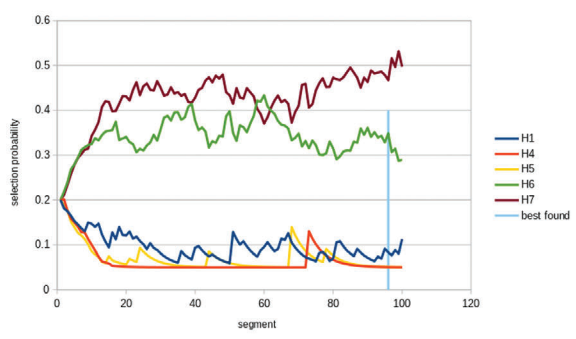
(a) Instance size 4 orders



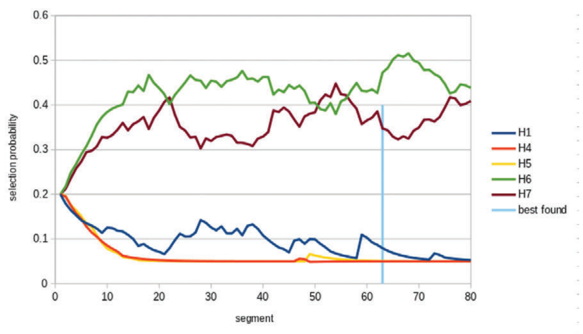
(b) Instance size 12 orders



(c) Instance size 35 orders



(d) Instance size 80 orders



(e) Instance size 150 orders

Fig. 13. Heuristics' weights obtained when running the algorithm on the problem instances in set 1. The y-axis represents the weight of each heuristic, and the x-axis represents a segment run. The blue vertical line represents the segment in which the best solution was found

efficient in finding near-optimal solutions. The comparison of the average and best found solutions shows that the algorithm is very robust and consistently finds optimal/near-optimal solutions to small and large problem instances.

Figure 13 shows the weights of the heuristics through each segment of the runs for some selected problem instances. Figure 13(b) shows that for smaller-size instances, the weights of heuristics

H1-swap, H5-cluster, and H7-similar increase until the algorithm obtains the optimal solution. This indicates that these heuristics are essential in the effectiveness of the algorithm for smaller-size instances. Figures 13(c)–(e) show that heuristics H6-greedy and H7-similar are critical to the performance of the algorithm since their weights are consistently higher than the weights of other heuristics. The spikes in the weights of heuristics H5-cluster and H1-swap indicate that they often help the algorithm to move to a new and/or better solution neighborhood. These observations also confirm our observations of the heuristics' performances from Sections 5.2.1 and 5.2.2, in which heuristics H6-greedy and H7-similar outperform the other heuristics. The observations in Table 5 show that our adaptive weight strategy plays an important role in the search process to adjust the heuristics' weights toward a more effective search in the solution space.

5.5. *Evaluation of the algorithm on the PDPTW*

In this section, we evaluate the performance of the proposed algorithm on some problem instances adopted from Homsi et al. (2020). The problem is known as the industrial and tramp ship routing and scheduling problem (ITSRSP), which is a common routing and scheduling problem in the shipping industry for bulk and liquid products, such as crude oil; chemical products (wet bulk), and phosphate rock (dry bulk). The ITSRSP extends the PDPTW with a heterogeneous fleet, compatibility constraints, different starting points and starting times for ships, and service flexibility with penalties. In this problem, a shipping company has a mix of mandatory and optional cargoes for transportation. Each cargo in the planning period must be picked up at its loading port within a specified time window, and then must be delivered to its destination port within a given time window. The shipping company controls a heterogeneous fleet of ships; each ship has a given initial time and location where it becomes available to handle new transportation tasks. Compatibility constraints, such as draft limits at the ports may restrict the mix of cargoes that a ship can transport. The shipping company may charter ships from the spot market to transport some of the cargoes. The goal of the problem is to construct the ships' routes and schedules, aiming at minimizing costs. In addition, it determines which spot cargo should be accepted for transportation, and which cargo should be transported by a spot charter to serve deliveries of all the mandatory cargo orders. Homsi et al. (2020) report extensive experimental analyses for the ITSRSP on several real-world problem instances. We use our proposed algorithm to solve the ITSRSP and compare the results with the results of their branch-and-price algorithm which obtained optimal solutions to 239 problem instances out of the 240 available instances.

The problem set includes 48 problem categories based on the types of load, types of sea, number of orders, and number of vehicles. Each category includes five problem instances, resulting in a total of 240 problem instances in this problem set. The algorithm is run five times on each problem instance to be able to report the best and average of obtained solutions for each problem instance. The results are presented in Table A7 for each problem category. In the first column, MUN and FUN represent mixed and full load, respectively. Also, SS, DS, C, and V stand for short sea, deep sea, the number of cargoes, and number of vessels for each problem category, respectively. The algorithm can solve small problems to optimally with a 0% gap. The overall average gap of 1.02% shows the promising performance of the algorithm.

6. Conclusion and future research

In this paper, we considered a 4PL routing and scheduling problem which determines the allocation of orders to vehicles and schedules the routes of vehicles for pickup and delivery regarding several real-world constraints, such as vehicle capacity, factory dock availability, product–vehicle compatibility, vehicle–dock compatibility, and a nonconventional cost structure. We formulated the problem mathematically and used a Gurobi solver to solve the problem. Due to the computational difficulty of the problem and limited performance of the mathematical model, we developed an algorithm inspired by the adaptive large neighborhood search. We incorporated seven heuristics in the algorithm to effectively search the solution space and escape from local optimal. These heuristics include swap heuristic, 3-exchange heuristic, 2-opt heuristic, random fit heuristic, clustering heuristic, greedy heuristic, and similar regret heuristic in addition to a wild escape heuristic which allows the algorithm to change the search neighborhood to enhance diversification. To determine the final combination of our heuristics to use in our algorithm based on their performance history, we conducted a comprehensive analysis of the heuristics. The experiments included the analysis of all possible combinations of heuristics, as well as individual analysis of each algorithm, using ANOVA (III), *t*-tests, and multiple linear regression analyses. On the basis of the complete evaluation of our heuristics, we chose a final combination of our heuristics, including the wild escape algorithm along with swap, random fit, clustering, greedy, and similar regret. The algorithm was used to solve several problem instances generated based on a real case study in Europe. The results demonstrate that the algorithm outperforms the mathematical model in terms of solution quality and run-time. The results also confirm that the greedy and similar regret heuristics are the best performing heuristics in terms of finding new improving solutions. Overall, the results illustrate that the proposed algorithm is a robust and efficient algorithm to find near-optimal solutions for a 4PL provider company.

For future research, we encourage research and development of heuristics to further improve the performance of the algorithm. The algorithm can also be tailored to solve similar problems in other industries considering specific constraints, such as worker availability.

References

4Flow, 2019. Industries & references - automotive manufacturers. https://www.4flow.com/ (accessed 24 October).

Azadian, F., Murat, A., Chinnam, R.B., 2017. An unpaired pickup and delivery problem with time dependent assignment costs: Application in air cargo transportation. *European Journal of Operational Research* 263, 1, 188–202.

Bent, R., Van Hentenryck, P., 2006. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research* 33, 4, 875–893.

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

Berbeglia, G., Cordeau, J.F., Laporte, G., 2010. Dynamic pickup and delivery problems. *European Journal of Operational Research* 202, 1, 8–15.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys* 35, 3, 268–308.

Crama, Y., Schyns, M., 2003. Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research* 150, 3, 546–571.

Dahle, L., Andersson, H., Christiansen, M., Speranza, M.G., 2019. The pickup and delivery problem with time windows and occasional drivers. *Computers & Operations Research* 109, 122–133.

Dondo, R., Cerdá, J., 2015. The heterogeneous vehicle routing and truck scheduling problem in a multi-door cross-dock system. *Computers & Chemical Engineering* 76, 42–62.

Favaretto, D., Moretti, E., Pellegrini, P., 2007. Ant colony system for a VRP with multiple time windows and multiple visits. *Journal of Interdisciplinary Mathematics* 10, 2, 263–284.

Ferreira, H.S., Bogue, E.T., Noronha, T.F., Belhaiza, S., Prins, C., 2018. Variable neighborhood search for vehicle routing problem with multiple time windows. *Electronic Notes in Discrete Mathematics* 66, 207–214.

Furtado, M.G.S., Munari, P., Morabito, R., 2017. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters* 45, 4, 334–341.

Ghilas, V., Demir, E., Van Woensel, T., 2016. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research* 72, 12–30.

Goeke, D., 2019. Granular tabu search for the pickup and delivery problem with time windows and electric vehicles. *European Journal of Operational Research* 278, 3, 821–836.

Hemmati, A., Hvattum, L.M., Fagerholt, K., Norstad, I., 2014. Benchmark suite for industrial and tramp ship routing and scheduling problems. *INFOR: Information Systems and Operational Research* 52, 1, 28–38.

Homsi, G., Martinelli, R., Vidal, T., Fagerholt, K., 2020. Industrial and tramp ship routing problems: closing the gap for real-scale instances. *European Journal of Operational Research* 283, 3, 972–990.

Kaufman, L., Rousseeuw, P.J., 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, New York.

Kirkpatrick, S., 1984. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics* 34, 5-6, 975–986.

Korsvik, J.E., Fagerholt, K., Laporte, G., 2011. A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers and Operations Research* 38, 2, 474–483.

Lau, H.C., Liang, Z., 2002. Pickup and delivery with time windows: algorithms and test case generation. *International Journal on Artificial Intelligence Tools* 11, 3, 455–472.

Li, H., Lim, A., 2003. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools* 12, 2, 173–186.

Manier, H., Manier, M.A., Al Chami, Z., 2016. Shippers' collaboration in city logistics. *IFAC-PapersOnLine* 49, 12, 1880–1885.

Mitrovic-Minic, S., 1998. Pickup and delivery problem with time windows: a survey. *SFU CMPT TR* 12, 1-43, 38.

Nanry, W.P., Barnes, J.W., 2000. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological* 34, 2, 107–121.

Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft* 58, 2, 81–117.

Reddy, C.K., Vinzamuri, B., 2019. A survey of partitional and hierarchical clustering algorithms. In Aggarwal, C.C. Reddy, C.K. (eds) *Data Clustering*. Chapman and Hall/CRC, New York, pp. 87–110.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40, 4, 455–472.

Savelsbergh, M.W.P., Sol, M., 1995. The general pickup and delivery problem. *Transportation Science* 29, 1, 17–29.

Shaw, P., 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. APES Group, Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35, 2, 254–265.

Sun, W., Yu, Y., Wang, J., 2019. Heterogeneous vehicle pickup and delivery problems: formulation and exact solution. *Transportation Research Part E: Logistics and Transportation Review* 125, 181–202.

Sze, J.F., Salhi, S., Wassan, N., 2016. A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: application to the vehicle routing problem. *Expert Systems with Applications* 65, 383–397.

Wang, C., Mu, D., Zhao, F., Sutherland, J.W., 2015. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup-delivery and time windows. *Computers and Industrial Engineering* 83, 111–122.

Appendix: Detailed evaluation results of heuristics

Table A1
Analysis of variance (ANOVA III). The sum of squares, degrees of freedom, mean squares, *F*-statistic, and *p*-value are reported for each heuristic as the source of variability

(a) Average improvement statistics						(b) Best improvement statistics					
Source	Sum sq.	*df*	Mean sq.	*F*	*P* >*F*	Source	Sum sq.	*df*	Mean sq.	*F*	*P*>*F*
H1-swap	0.052	1	0.052	0.09	0.7682	H1-swap	0.075	1	0.0745	0.23	0.6306
H2-exchange	0.748	1	0.748	1.26	0.2628	H2-exchange	0.195	1	0.1949	0.61	0.4369
H3-2opt	0.038	1	0.038	0.06	0.8017	H3-2opt	0	1	0.0001	0	0.988
H4-random	22.432	1	22.432	37.66	0	H4-random	8.372	1	8.3718	26	0
H5-cluster	12.734	1	12.734	21.38	0	H5-cluster	3.679	1	3.6789	11.43	0.0008
H6-greedy	117.945	1	117.945	198	0	H6-greedy	66.236	1	66.2357	205.71	0
H7-similar	112.406	1	112.406	188.7	0	H7-similar	67.081	1	67.0813	208.33	0
Instances	371.704	4	92.926	156	0	Instances	344.971	4	86.2428	267.84	0
Error	350.257	588	0.596			Error	189.331	588	0.322		
Total	975.207	599				Total	671.378	599			

Table A2
Results of multiple linear regression analysis. We report the coefficient estimate, standard error of the coefficients, *t*-statistics to test if the term is significant, and *p*-value. Instances #1 to #4 are added to calculate the instance-specific random effects

(a) Average improvement statistics, $R^2 = 0.641$					(b) Best improvement statistics, $R^2 = 0.718$				
Term	Estimate	*SE*	*t*-Stat	*p*-Value	Term	Estimate	*SE*	*t*-Stat	*p*-Value
Intercept	994.58	0.11713	8491.5	0	Intercept	994.97	0.086114	11554	0
H1-swap	−0.018582	0.063017	−0.29487	0.7682	H1-swap	0.022291	0.046332	0.48112	0.63061
H2-exchange	−0.07063	0.063017	1.1208	0.26283	H2-exchange	−0.036047	0.046332	−0.77801	0.43687
H3-2opt	−0.01583	0.063017	−0.2512	0.80175	H3-2opt	−0.00069693	0.046332	−0.015042	0.988
H4-random	0.39108	0.063729	6.1366	1.5502e−09	H4-random	0.23891	0.046855	5.099	4.6112e−07
H5-cluster	−0.29466	0.063729	−4.6236	4.6407e−06	H5-cluster	−0.15838	0.046855	−3.3801	0.00077249
H6-greedy	0.89676	0.063729	14.071	5.7098e−39	H6-greedy	0.67202	0.046855	14.342	3.2e−40
H7-similar	0.87544	0.063729	13.737	1.923e−37	H7-similar	0.67629	0.046855	14.434	1.2062e−40
Instance #1	0.50928	0.099639	5.1113	4.3337e−07	Instance #1	0.64176	0.073257	8.7605	2.0732e−17
Instance #2	−0.051052	0.099639	−0.51237	0.60859	Instance #2	0.16783	0.073257	2.291	0.022316
Instance #3	1.6601	0.099639	16.662	2.4375e−51	Instance #3	1.837	0.073257	25.077	6.2887e−95
Instance #4	1.755	0.099639	17.614	4.4128e−56	Instance #4	1.6686	0.073257	22.778	8.2582e−83

Table A3
t-Tests for mean results of the G-heuristics versus no use of G-heuristics. 1 rejects the null hypothesis, and 0 represents a failure to reject it

Populations	Improvement type	Tail	H-stat	*p*-value	*t*-Stat	Confidence interval
$P_H - P_N$	Average	Both	0	0.5187	−0.6458	−0.3785 – 0.1912
$P_H - P_N$	Average	Right	0	0.7407	−0.6458	−0.3326 – inf
$P_H - P_N$	Average	Left	0	0.2593	−0.6458	−inf – 0.1453
$P_H - P_N$	Best	Both	0	0.6903	0.3986	−0.2174 – 0.3281
$P_H - P_N$	Best	Right	0	0.3452	0.3986	−0.1734 – inf
$P_H - P_N$	Best	Left	0	0.6548	0.3986	−inf – 0.2841

Table A4
Analysis of variance for the G-heuristics in combination with the significant heuristics

(a) Average improvement statistics						(b) Best improvement statistics					
Source	Sum sq.	*df*	Mean sq.	*F*	*P*>*F*	Source	Sum sq.	*df*	Mean sq.	*F*	*P* >*F*
G+H4	19.024	1	19.0236	538.34	0	G+H4	12.879	1	12.8793	609.42	0
G+H6	0	1	0	0	0.9859	G+H6	0.131	1	0.1307	6.19	0.0132
G+H7	0.005	1	0.0045	0.13	0.7204	G+H7	0.239	1	0.2389	11.31	0.0008
G+H4+H6	0.059	1	0.0589	1.67	0.1973	G+H4+H6	0.223	1	0.2227	10.54	0.0012
G+H4+H7	0.001	1	0.0009	0.03	0.8722	G+H4+H7	0.153	1	0.1525	7.22	0.0074
G+H6+H7	0.22	1	0.2202	6.23	0.0128	G+H6+H7	0.42	1	0.4195	19.85	0
G+H4+H6+H7	0.166	1	0.1657	4.69	0.0308	G+H4+H6+H7	0.395	1	0.395	18.69	0
Instances	309.022	4	77.2555	2186.21	0	Instances	295.738	4	73.9346	3498.45	0
Error	19.365	548	0.0353			Error	11.581	548	0.0211		
Total	385.274	559				Total	352.547	559			

Table A5
Results of multiple linear regression model for the G-heuristics in combination with the significant heuristics. Instances #1 to #4 are added to calculate the instance-specific random effects

(a) Average improvement statistics, $R^2 = 0.95$					(b) Best improvement statistics, $R^2 = 0.967$				
Term	Estimate	*SE*	*t*-Stat	*p*-Value	Term	Estimate	*SE*	*t*-Stat	*p*-Value
Intercept	995.85	0.035525	28,032	0	Intercept	995.88	0.027473	36,249	0
G+H4	−0.89285	0.038481	−23.202	1.8003e−83	G+H4	−0.73464	0.029759	−24.686	5.012e−91
G+H6	0.00068095	0.038481	0.017696	0.98589	G+H6	0.07402	0.029759	2.4873	0.013167
G+H7	0.013782	0.038481	0.35815	0.72037	G+H7	0.10006	0.029759	3.3625	0.00082635
G+H4+H6	0.049672	0.038481	1.2908	0.19731	G+H4+H6	0.096599	0.029759	3.246	0.0012417
G+H4+H7	−0.0061945	0.038481	−0.16097	0.87217	G+H4+H7	0.079948	0.029759	2.6865	0.0074396
G+H6+H7	0.096062	0.038481	2.4963	0.012841	G+H6+H7	0.13259	0.029759	4.4554	1.0157e−05
G+H4+H6+H7	0.083317	0.038481	2.1651	0.030808	G+H4+H6+H7	0.12865	0.029759	4.3232	1.8269e−05
Instance #1	0.53458	0.02512	21.281	1.0604e−73	Instance #1	0.70773	0.019426	36.432	1.6371e−148
Instance #2	0.031671	0.02512	1.2608	0.20793	Instance #2	0.27728	0.019426	14.273	1.5823e−39
Instance #3	1.71	0.02512	68.073	1.6221e−-269	Instance #3	1.873	0.019426	96.415	0
Instance #4	1.5959	0.02512	63.531	6.3233e−255	Instance #4	1.5781	0.019426	81.237	8.6749e−308

Table A6
t-tests for analysis of individual heuristics

Populations	Tail	H-stat	*p*-value	*t*-Stat	Confidence interval
$P_{HA}^{H_1} - P_{NA}^{H_1}$	Both	0	0.5427	−0.6090	−0.1580 – 0.0727
$P_{HA}^{H_1} - P_{NA}^{H_1}$	Right	0	0.7286	−0.6090	−0.1325 – inf
$P_{HA}^{H_1} - P_{NA}^{H_1}$	Left	0	0.2714	−0.6090	−inf – 0.0472
$P_{HB}^{H_1} - P_{NB}^{H_1}$	Both	0	0.9240	−0.0955	−0.1428 – 0.1296
$P_{HB}^{H_1} - P_{NB}^{H_1}$	Right	0	0.5380	−0.0955	−0.1208 – inf
$P_{HB}^{H_1} - P_{NB}^{H_1}$	Left	0	0.1076	−0.0955	−inf – 0.1076
$P_{HA}^{H_2} - P_{NA}^{H_2}$	Both	1	3.4853e−10	−6.5098	−0.0692 – 0.0412
$P_{HA}^{H_2} - P_{NA}^{H_2}$	Right	0	1.0000	−6.5098	−0.0661 – inf
$P_{HA}^{H_2} - P_{NA}^{H_2}$	Left	1	1.7426e−10	−6.5098	−inf – 0.0443
$P_{HB}^{H_2} - P_{NB}^{H_2}$	Both	1	4.5425e−10	−3.5486	−0.0370 – 0.0106
$P_{HB}^{H_2} - P_{NB}^{H_2}$	Right	0	0.9998	−3.5486	−0.0349 – inf
$P_{HB}^{H_2} - P_{NB}^{H_2}$	Left	1	2.2712e−04	−3.5486	−inf – 0.0127
$P_{HA}^{H_3} - P_{NA}^{H_3}$	Both	1	0.0244	−2.2635	−0.0276 – -0.0043
$P_{HA}^{H_3} - P_{NA}^{H_3}$	Right	0	0.9878	−2.2635	−0.0250 – inf
$P_{HA}^{H_3} - P_{NA}^{H_3}$	Left	1	0.0122	−2.2635	−inf – -0.0069
$P_{HB}^{H_3} - P_{NB}^{H_3}$	Both	0	0.6271	−0.4864	−0.0134 – 0.0081
$P_{HB}^{H_3} - P_{NB}^{H_3}$	Right	0	0.6865	−0.4864	−0.0116 – inf
$P_{HB}^{H_3} - P_{NB}^{H_3}$	Left	0	0.3135	−0.4864	−inf – 0.0063
$P_{HA}^{H_5} - P_{NA}^{H_5}$	Both	0	0.4702	−0.7231	−0.0301 – 0.0118
$P_{HA}^{H_5} - P_{NA}^{H_5}$	Right	0	0.7649	−0.7231	−0.0255 – inf
$P_{HA}^{H_5} - P_{NA}^{H_5}$	Left	0	0.2351	−0.7231	−inf – 0.0071
$P_{HB}^{H_5} - P_{NB}^{H_5}$	Both	1	1.7956e−04	−3.7972	−0.0167 – 0.0528
$P_{HB}^{H_5} - P_{NB}^{H_5}$	Right	1	8.9779e−05	−3.7972	−0.0197 – inf
$P_{HB}^{H_5} - P_{NB}^{H_5}$	Left	0	0.9999	−3.7972	−inf – 0.0499

Table A7
Evaluation of the algorithm on the PDPTW problems from Homsi et al. (2020)

Problem category	Lower bound by Homsi et al. (2020)	Best	Average	Min. gap%	Average gap%
SS-MUN-C7-V3	1,244,723.8	1,244,723.8	1,244,723.8	0	0
SS-MUN-C10-V3	2,074,204.4	2,074,204.4	2,074,204.4	0	0
SS-MUN-C15-V4	2,319,665.2	2,319,665.2	2,321,030.6	0	0.06
SS-MUN-C18-V5	2,575,253.8	2,575,253.8	2,575,965.6	0	0.02
SS-MUN-C22-V6	3,575,102.2	3,575,102.2	3,616,344.4	0	1.15
SS-MUN-C23-V1	2,294,171	2,302,963.4	2,309,484.6	0.38	0.67
SS-MUN-C30-V6	4,516,864.8	4,564,848.2	4,626,170.6	1.06	2.38
SS-MUN-C35-V7	4,790,688.4	4,803,117.2	4,861,170.6	0.25	1.53
SS-MUN-C60-V1	8,227,186.8	8,311,401.8	8,362,004.8	1.02	1.64
SS-MUN-C80-V2	1,0460,352	10,674,740.2	10,767,238.6	2.04	2.88
SS-MUN-C100-V3	12,836,636.8	13,102,804	13,264,492.6	2.08	3.32
SS-MUN-C130-V4	16,679,910.2	17,025,237.4	17,108,167.4	2.08	2.57

Continued

Table A7
(Continued)

Problem category	Lower bound by Homsi et al. (2020)	Best	Average	Min. gap%	Average gap%
SS-FUN-C8-V3	1,550,159.4	1,550,159.4	1,550,159.4	0	0
SS-FUN-C11-V4	1,165,548	1,165,548	1,166,450.2	0	0.08
SS-FUN-C13-V5	2,434,916.6	2,434,916.6	2,437,966.2	0	0.14
SS-FUN-C16-V6	3,664,758.8	3,664,758.8	3,674,431.4	0	0.26
SS-FUN-C17-V1	2,767,263.8	2,767,481.8	2,767,518	0	0.01
SS-FUN-C20-V6	3,175,169.6	3,175,169.8	3,182,503.6	0	0.23
SS-FUN-C25-V7	4,015,184.4	4,024,034	4,028,984.8	0.23	0.35
SS-FUN-C35-V1	3,263,905.2	3,264,732.8	3,269,839	0.02	0.17
SS-FUN-C50-V2	7,577,835.4	7,598,806.4	7,623,339.2	0.27	0.59
SS-FUN-C70-V3	10,484,147.8	10,586,647.8	10,627,010	0.99	1.37
SS-FUN-C90-V4	13,556,888.4	13,722,603.6	1,3760,974.8	1.22	1.51
SS-FUN-C100-V5	14,086,575.6	14,182,778	14,223,850.6	0.68	0.97
DS-MUN-C7-V3	6,181,445.2	6,181,445.2	6,181,445.2	0	0
DS-MUN-C10-V3	8,502,254.6	8,502,254.6	8,502,254.6	0	0
DS-MUN-C15-V4	12,217,923.6	12,217,923.6	12,427,880.4	0	1.77
DS-MUN-C18-V5	34,066,702.4	343,31,049.8	34,331,049.8	0.64	0.64
DS-MUN-C22-V6	39,427,492	39,454,569.8	39,716,623.2	0.07	0.75
DS-MUN-C23-V1	34,091,535.6	34,091,535.6	34,091,535.6	0	0
DS-MUN-C30-V6	20,552,646	20,624,263.4	20,639,655.2	0.36	0.43
DS-MUN-C35-V7	60,267,056	60,513,381.6	61,294,901.2	0.43	1.7
DS-MUN-C60-V1	85,097,329.4	86,785,225.2	88,511,693	2.05	4
DS-MUN-C80-V2	74,530,364.2	76,298,963.4	76,923,325	2.39	3.25
DS-MUN-C100-V3	152,663,613.8	157,078,220.8	159,555,512.2	2.89	4.52
DS-MUN-C130-V4	230,403,055.8	241,065,696.8	242,802,090.4	4.63	5.38
DS-FUN-C8-V3	7,453,236.2	7,453,236.2	7,453,236.2	0	0
DS-FUN-C11-V4	30,244,798	3,024,4798	30,244,798	0	0
DS-FUN-C13-V5	11,359,352.8	11,359,352.8	11,359,352.8	0	0
DS-FUN-C16-V6	44,680,989	44,743,419.4	44,743,419.4	0.15	0.15
DS-FUN-C17-V1	13,532,838.8	13,532,838.8	13,532,838.8	0	0
DS-FUN-C20-V6	16,761,493	16,761,493	16,765,955.2	0	0.02
DS-FUN-C25-V7	21.558.743.4	21,559,136.8	21,614,392.6	0	0.27
DS-FUN-C35-V1	87.473.158.4	87,596,993.6	88,173,408	0.13	0.78
DS-FUN-C50-V2	40.965.239.8	41,149,781.2	41,244,028.6	0.44	0.66
DS-FUN-C70-V3	150.576.798	151,264,279.6	151,516,467.2	0.46	0.63
DS-FUN-C90-V4	199.938.155.8	201,551,756.6	202,338,530.8	0.81	1.21
DS-FUN-C100-V5	214,956,251.8	216,698,140.2	217,243,959.8	0.82	1.07