10.24425/acs.2025.156309

Archives of Control Sciences
Volume 35(LXXI), 2025
No. 3, *pages* 505–560

Multi-criteria parcel locker-based vehicle routing with pickup and delivery

Jarosław RUDY

In this paper a variant of Vehicle Routing Problem with Pickup and Delivery for parcel locker-based delivery is considered. The mathematical model of the problem is formulated with a bi-criteria goal function to minimize both the total distance traveled by all vehicles and the maximal order delivery time. The model assumes alternative delivery locations, where the order has to be delivered to the closest locations where its originally indented location has no available lockers left. The problem considers two types of orders (delivery and pickup) and accounts for vehicle and locker capacities, a possibility of non-symmetric distances, varying in-day travel times and order size categories. An order-based solution representation is proposed. A procedure to compute the goal function value and establish solution feasibility via the use of Discrete Event Simulation is presented. Three additional solution representations are proposed based on the idea of grouping orders and a number of properties showcasing their advantages and drawbacks are established. A procedure to generate problem instances based on real-life data is proposed and used to construct a basic benchmark dataset with 320 problem instances with number of orders ranging from 70 to 7177. Two heuristic solving methods are proposed: a greedy algorithm and a Genetic Algorithm metaheuristic. Each algorithm was implemented in four variants, one for each of the proposed solution representations. A number of computer experiments are performed on the proposed benchmark dataset and its derivatives, to establish the effectiveness of solution representations and algorithms, as well as to research the influence of several parameters on the obtained solutions and their feasibility. Results indicate that the alternative representations vastly outperform the standard representation in terms of Hyper-volume Indicator and both considered criteria. The genetic Algorithm also significantly outperforms the greedy method, although the advantage decreases with the increase of the number of orders. Other experiments indicate how vehicle-to-order, delivery-to-pickup, and locker occupancy ratios affect the feasibility of the solution.

Key words: parcel locker-based delivery, vehicle routing, multi-criteria optimization, operations research

J. Rudy (e-mail: jaroslaw.rudy@pwr.edu.pl) is with Department of Control Systems and Mechatronics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland.

Received 25.03.2025.

1. Introduction

Parcel locker-based delivery (PLBD) is a popular modern delivery service, where instead of delivering parcels directly to customers' home or office, each customer designates one of the existing fixed parcel locker locations for delivery. Each day, couriers deliver parcels to designated points from which customers may retrieve the parcels at their leisure. This is convenient for both couriers (since fewer delivery points have to be visited with easier and more predictable parking spots) and customers (since couriers become independent from customer-specific time windows for parcel retrieval). Moreover, such a service is often also extended with parcel sending instead of just parcel delivery, meaning that customers can leave their parcels in the parcel locker of their choice. Such parcels are then collected during the day and transported to the company hub. Once again, this simplifies things for both couriers and customers, saving them from having to visit more distant company hubs or post offices. The effectiveness of PLBD compared to traditional home delivery has been the focus of several research papers, which consistently concluded that PLBD leads to a significant decrease in traffic congestion and pollution [3, 8, 19]. Similarly, research has shown that PLBD systems reduce the distance traveled by vehicles [12], the time spent by vehicles "on the curb", as well as parcel delivery times [20].

As with many processes in logistics and transport, PLBD can be viewed as an optimization problem and an extension of the well-known Vehicle Routing Problem (VRP). There exist obvious similarities between PLBD and common VRP variants such as the Capacitated Vehicle Routing Problem (CVRP), where each vehicle has limited capacity, and the Vehicle Routing Problem with Pickup and Delivery (VRP-PD). However, PLBD has several features that are uncommon in VRP problems. First, in regular VRP capacity of customers is negligible, i.e. we assume the customer has always the space necessary to accept the delivery. However, in PLBD each parcel locker might be a destination of multiple orders and it is possible that the orders will be collected by the customers only the next day. Thus, locker capacity must be considered. Secondly, parcels come in different weights and sizes. While the former is a typical VRP feature (i.e. weight affects whether a group of parcels fit into a vehicle), the latter would either require making parcel lockers big (so any parcel can fit in any locker) or making lockers of different sizes. In practice, different size categories are used, meaning that a smaller parcel can fit into a larger locker, but not the other way around. Third, we include the effect of in-day road conditions on the travel times between parcel locker locations. Finally, the most important feature is the existence of alternative locations. If a parcel cannot be delivered to a designated location (due to no available lockers left at the moment), the parcel should instead be delivered

to the nearest available location. Similarly, if no free locker is available in that alternative location either, couriers should check further locations in order of their distance from the originally intended delivery location. This ensures that customers do not have to travel far to retrieve their parcel.

In this paper, we consider an optimization problem for a PLBD system with the practical features mentioned above. As for the objective function, we view the problem from two sides: the company's side (where delivery costs based on the total distance traveled are minimized) and the customer's side (where the time of completion of all deliveries is minimized). At the same time, we consider practical problem instances of sizes up to over 7000 orders, while also using real-life geospatial data to construct a benchmark instance dataset. To our knowledge, our paper is the first approach to PLBD optimization with multiple alternative locations and problem sizes of this magnitude. Finally, as it seems naturally beneficial to group parcels intended for the same location together, we propose several solution representations based on different levels of grouping and showcase their theoretical and practical properties.

Due to the above, the main contributions of this paper are as follows:

1. A bi-criteria mathematical formulation for a locker-based delivery process with pickup and delivery, vehicle and locker capacities, alternative delivery locations and minimization criteria of total distance and maximum delivery time is presented.
2. Four different solution representations based on various levels of grouping orders together are proposed and their theoretical properties are presented and discussed.
3. A general three-tier solution framework is proposed using high-level solution provided, a mid-level Discrete Event Simulation and a low-level decision agent.
4. Two solving methods, a greedy heuristic and Genetic Algorithm (GA) metaheuristic, are proposed with multiple variants for each of the proposed solution representations.
5. An instance generation procedure based on real-life parcel locker and geospatial data and a benchmark instance dataset based on that generation procedure are proposed.
6. Computer experiments are conducted on the proposed dataset with problem sizes of up to 7000 orders, researching the effectiveness of solution representations, algorithms, and the influence of problem parameters on solution quality and feasibility.

The remainder of the paper is structured as follows. In Section 2 we provide a literature review. Section 3 presents the formulation of the problem. Section 4 introduces a basic solution representation and discusses the procedure for computing the value of the goal function value using a Discrete Event Simulation (DES). In Section 5 three alternative solution representations are proposed and their theoretical properties are discussed. In Section 6 the solving methods are proposed. Section 7 presents an instance generation procedure and a dataset based on real-life geospatial data. In Section 8 the results of computer experiments of the proposed solving methods on the generated instance dataset are described. Finally, Section 9 contains the conclusions.

2. Literature review

In this section, we will provide a brief overview of the recent research papers related to PLBD, highlighting differences compared to the approach proposed in this paper. A more comprehensive review on existing studies on PLBD can be found in [1].

We will start with the research most related to the approach considered in this paper. Sitek *et al.* [25] considered a PLBD system with alternative delivery, pickups, and time windows. The authors assumed capacity constraints for both vehicles and delivery points. For each order, the authors defined a set of locations to which the order can be delivered. The objective function was a sum of distances and penalties, with the penalty being based on the chosen delivery point. The authors proposed a hybrid solving method combining Constraint Logic Programming (CLP) and Mathematical Programming (MP). Computer experiments were performed on instances of up to 2000 orders and 200 locations. A very similar approach was presented in paper [24], where CLP and MP were extended with a GA solving method, this time for instances up to 1500 orders and 250 locations. On the other hand, Orenstein *et al.* [17] consider a similar PLBD system, but they also consider fixed costs per vehicle as the third term of their objective function and take the unloading time into consideration as well. However, vehicle capacity and pickups are not considered. As a solving method, the authors proposed savings heuristics, the petal method and a Tabu Search (TS) method. The research was conducted on instances of up to 1500 orders, but only 50 locations. The authors consider a potential extension into a stochastic system.

Chen *et al.* [4] considered a multi-objective PLBD system with pickup and delivery, optimizing the number of vehicles, delivery costs, and customer satisfaction. Non-dominated Sorting Genertic Algorithm (NSGA-II) combined with Hybrid Insertion (HI) was used as a solving method and tested on 15 datasets using Hyper-volume Indicator (HVI) as quality measures, outperforming the traditional

NSGA-II method. Similarly, Idzikowski *et al.* [7] considered a PLBD with two criteria of time travel time and total penalty for late delivery of orders. The authors considered order deadline and arrival time, vehicle capacity, and time limit, as well as order service and vehicle parking times. However, alternative locations and parcel capacities were not considered. Three solving methods: GA, TS and a greedy heuristics were proposed and the results (using HVI) indicated that both GA and TS are useful depending on the circumstances. Interestingly, the authors used real-life data based on one of the major cities in Poland to construct instances of up to 3500 orders and 300 locations. An approach to optimize the balance between delivery costs and customer inconvenience was shown in the paper [10]. Solomon VRP instances of up to 100 customers and 14 lockers were solved through CPLEX to showcase the behavior of unified distance-customer costs.

Yu *et al.* [30] considered a PLBD system that mixes home deliveries with parcel locker options and time windows, with the goal of minimizing the total travel cost. The authors proposed a Simulated Annealing (SA) solving method tested on instances based on well-known Solomon VRP instances of up to 100 customers. Similarly, in paper [29] a Vehicle Routing Problem for a PLBD system with time windows, electric vehicles, and partial recharges. The authors assumed three types of customers: home delivery, self-pickup, and flexible customers. Vehicle capacity is considered, although the locker capacity is the same as the number of customers, making it unlimited in practice. The authors proposed an Adaptive Large Neighborhood Search (ALNS) algorithm and performed numerical experiments of up to 100 orders and 5 lockers. ALNS for a capacitated VRP for a PLBD system was also used by Saker *et al.* Saker [22]. The authors use destroy and repair operators together with various selection schemes. The results on instances of up to 1000 customers show a 6–30% error compared to Mixed Integer Programming (MIP), but with 90 times less computation time. Another mixed home delivery-parcel locker PLBD system was considered by Moradi *et al.* [15]. In addition to lockers and vehicles, an electric-powered zero-emission robot is added to each vehicle to serve a portion of the customers. The authors seek to minimize costs by optimizing locker locations and vehicle routes. A MIP solution implemented in Gurobi is proposed and tested on a set of benchmark instances. Finally, Pan *et al.* [23] considered a interesting approach with multiple delivery companies that exchange packages for deliveries using a network of parcel lockers. The authors decompose the problem formulation into two levels: the upper problem deals with goods and parcel locker assignment, while the bottom problem is a more traditional multi-depot CVRP. GA method is used for the upper level, while the Lin-Kernighan heuristic solves the lower level. The problem instances considered in the numerical experiment were small, partially due to the two-level nature of the formulation.

Although many authors consider only static parcel lockers, a number of papers research the concept of mobile parcel lockers (MPLs), which increases the complexity of the overall problem. An example of such paper is the work by Yan *et al.* [28]. The authors used real-life data from Taoyuan City with up to 800 and 15 parcel lockers. Similarly, Peppel *et al.* [18] researched the influence of MPLs on costs and CO_2 emissions. The study uses real-life data on a large scale, covering 15 cities over 3 weeks with over 700 000 orders in total (though per day and city this number is reduced to around 2000 orders on average). The results suggest that MLPs lead to 5–9% savings in costs and CO_2 emissions.

Aside from typical solving methods such as MIP, CPLEX, or metaheuristics, machine learning methods are also used for PLBD system optimization. For example, Liu *et al.* [13] consider PLBD system with MLPs, time windows, MLP capacity and changing customer demand patterns. The authors formulate a MIP model and design a Q-learning algorithm to solve. The results indicated that the algorithm outperforms the Gurobi solver for larger instances, while also having shorter running time. This is one of the approaches that considers a stochastic approach to PLBD instead of a deterministic one. Another stochastic PLBD was shown by Wang *et al.* [27], who considered a routing problem with MLPs with capacity, time windows, three types of uncertain demands, and two types of failures. A TS algorithm was proposed to solve the problem with consideration of recourse costs, showing a significant improvement of service reliability with little cost.

The summary of the literature review for the most relevant papers is shown in Table 1. Smaller checkmarks indicate that a feature is either only partially present or that it differs significantly from our variant. Overall, the presented literature overview indicates that PLBD is an important, active and contemporary research topic with several articles that consider optimization of delivery costs or distance, and customer convenience. However, while some authors consider alternative locations, locker capacity, order sizes, and unloading times, there are very few articles that consider all of these constraints. Existing papers that consider alternative locations work by considering a set of viable alternative points and applying a penalty to each one. As such, there is no research that considers alternative locations as a problem constraint (i.e. requiring the order to be delivered as close to the intended location as possible). Moreover, there is also a lack of papers that consider the influence of traffic conditions on travel times for PLBD systems, despite such constraint being applied to more general VRP approaches (see, for example, papers [5, 11]). Existing papers also rarely consider problem instances of more than 2000 orders and 250 locations. Finally, seemingly all existing approaches treat orders individually, despite the fact that multiple orders are often intended for the same parcel locker. Due to this, we propose an approach to

Table 1: Summary of the characteristics of the most relevant literature papers in comparison to our paper

Papers	Alternative locations	Pickups	Vehicle capacity	Locker capacity	Locker size	Park/service time	Road conditions	Goal function	Real-life data	Large instances
[24, 25]	✓	✓	✓	✓	✓			✓		✓
[17]	✓			✓	✓	✓		✓		✓
[4]		✓	✓					✓		
[7]		✓			✓			✓	✓	✓
[10]			✓					✓		
[29, 30]			✓	✓				✓		
[22]			✓	✓						✓
This paper	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

PLBD that considers all of the above problem constraints (alternative locations, locker capacity, order size, variable travel times, parking/serving time) while also presents the effects of grouping orders together and showcasing the result for real-life-based instances of up to 7000 orders and 900 locations.

3. Problem formulation

In this section, we will formulate the variant of the PLBD optimization problem under consideration in this paper. Due to a number of constraints we will refer to this problem simply as "PLBDP" (for "Parcel Locker-Based Delivery Problem") for the sake of convenience. We will start by describing the delivery process, then move onto specifying the problem input, solution, the goal function, and constraints. A full MILP formulation for the considered problem is unwieldy and impractical, especially as the sizes of real-life problems are too large to tackle with exact methods. Moreover, the formulation is more difficult than for typical VRP due to (1) locker sizes, (2) limited locker numbers, and (3) the decisions required due to the existence of alternative parcel locations. Thus, here we will provide an event-based formulation (one that is not suitable for, e.g., MILP solvers like Gurobi), which is easily extendable to other constraints. This formulation will be used in conjunction with solution representations proposed later in the paper.

3.1. Delivery process

In PLBDP, the parcel goes through several stages. In the first stage, customers submit parcels to the system. This can be done by bringing the parcel directly to a regional company hub of the delivery company, by using more traditional postal service or sending points, but also by leaving a parcel directly in one of the parcel lockers available in the city. Every day, all parcels placed in the lockers by customers are collected by couriers and transported to the local hub at the end of the day. In the second stage, the parcels are transported to the central in the evening and sorted there. In the third stage, sorted parcels are transported to the correct local hub, depending on the destination overnight. In the fourth stage, the parcels are ready to be delivered to the appropriate parcel lockers. Thus, in many countries, for each parcel the inter-country delivery process takes two days, regardless of the sending and destination points.

Based on the above, we focus on a single day delivery process for a single local hub. This means that we have a set of parcels in the hub that have to be delivered on this day and a set of parcels that we need to collect from the lockers and bring back to the hub (for them to be delivered on the next day). Each parcel has a designated locker location for either delivery or pickup. Parcels have weight and size. The weight determines how many parcels can be carried by a single vehicle and can be physically interpreted as mass or volume. The size determines what lockers the parcel can be placed in. Namely, each considered delivery/pickup locations there is a number of lockers of each size. A "small" size parcel can be placed in a "medium" size locker, but not the other way around. We assume, a parcel should be placed in the smallest available locker at the time of delivery.

However, even with that strategy, in practice it may happen that the target location does not have enough free lockers to accommodate all parcels intended for it. Since we assume that all parcels must be delivered, an alternate locker location has to be found. Namely, the courier should travel to the nearest location and try to do a delivery there. If there are no free lockers there either, the process repeats, visiting subsequent "candidate" locations, as close to the original intended location as possible. This approach minimizes the distance required for customers to travel from their preferred parcel retrieval location, thus maximizing their satisfaction. Such a strategy is thus very desirable for companies, but has not been formulated as a part of an optimization problem before.

As PLBDP takes place in a city environment, we assume that arriving and departing from a location takes non-zero time. Similarly, the time needed to deliver or pickup a parcel takes non-zero time (parcel needs to be located, brought to the locker, locker needs to be opened, parcel needs to be placed in the locker, delivery has to be registered, etc.). As mentioned earlier, drivers can leave the

hub only after all the parcels for that day have arrived from the central sorting hub. As such, drivers leave for delivery only after a certain time in the morning. The travel time is also affected by traffic conditions. We also assume that drivers visit the locations on their list without unnecessary breaks.

In such a delivery system, there are several quality measures that one would want to optimize. In our case, we consider two such measures, one from the perspective of the delivery company and one from the perspective of the customers. The first is the minimization of the total distance traveled by all vehicles, minimizing the direct costs incurred by the delivery company. The second is the time when all parcels have been delivered, thus allowing customers to collect them as soon as possible. It is important to note that this measure considers only delivery parcels. This is because the pick-up parcels can be picked up at any time throughout the day — they will be delivered the next day regardless. Delivery parcels are different, as delivery done at 5 PM instead of 1 PM makes it harder for customers to collect their parcels.

The basic outline of the process is shown in Figure 1, including lockers with difficult locations and sizes, pick-up orders, and different road connections. Please note that the locker at location 7 does not have any delivery or pickup

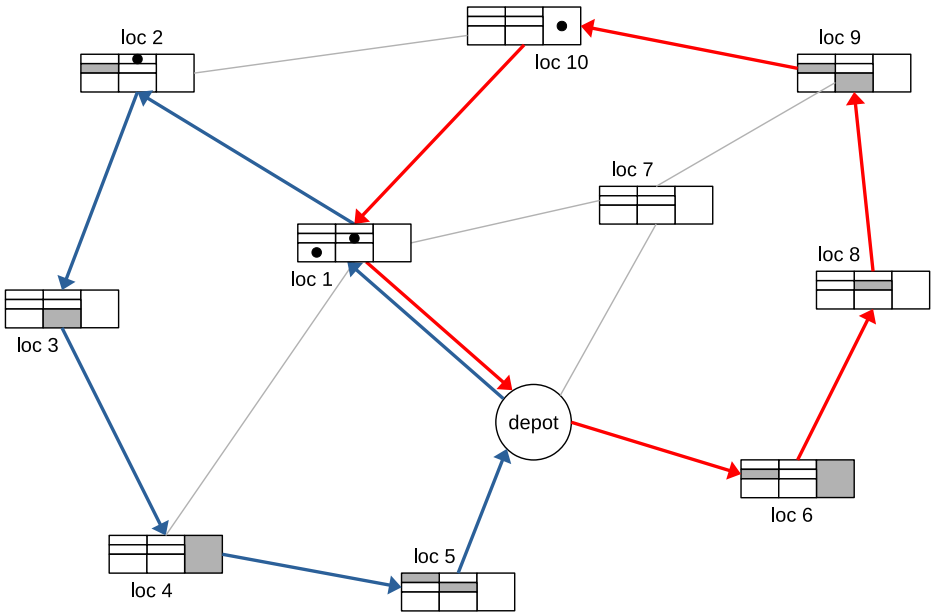


Figure 1: Illustration of the delivery process setting with 10 locker locations and roadway connections. Grey indicates unavailable (non-empty) lockers. Black dots indicates pickup orders. Delivery orders are not shown since they all originate at the depot. An example of routes are shown in red and blue

orders planned for it, but it cannot be removed from the setting as it might serve as an alternative location for some order originally intended for another location.

3.2. Discrete Event Simulation

The above delivery process is complex, especially due to the existence of alternative locations. This is especially true since whether a vehicle needs to visit an alternate location is determined by the lockers available at the time, and this depends on other vehicles. This effect of vehicles interacting with each other introduces another level of complexity. Due to this, we will not present a full formal mathematical formulation. Instead, we will describe the system in terms of events and the corresponding system state and propose a Discrete Event Simulation (DES) to construct a feasible solution. DES is a well-known framework used in many fields of science, including optimization [2]. DES will allow us to process and generate events in an efficient way (it considers only times when events occur and skips the in-between times), while also being easily extendable in the future.

It is important to note that DES is not aimed at solving the full problem. Rather, its goal is to construct a feasible solution (or reporting infeasibility) and to provide the value of the goal function. The input for DES is two-fold: (1) the actual problem input data (i.e. orders, location distances, etc.) and (2) a simplified solution representation provided by another algorithm. The use of a simplified representation is to reduce the size of the solution space (especially the number of infeasible solutions). Moreover, we model our system such that some decisions (i.e. choosing a locker size and whether to visit an alternative location) are performed by couriers on-site. Thus, DES also needs to have a mechanism (called a strategy agent) to make those decisions. In other wods, we view our problem as decomposed into three separate tiers:

1. On the top tier, a solution representation is constructed. This can be done via a number of algorithms, such as Branch and Bound, metaheuristics etc.
2. On the middle tier, DES is invoked to construct the (full) solution, establish its feasibility, and calculate the goal function value.
3. On the bottom tier an strategy agent is invoked when a decision regarding locker size and alternative location is needed.

The outline of this concept is shown in Figure 2.

The input data, (direct) solution, constraints, and goal function will be described further in this section. The basic solution representation and the implementation of the DES will be described in Section 4. The summary of most important symbols and notations is shown in Table 2.

Table 2: Summary of most important notation

Symbol	Meaning
	Indexes
i, j, a, b	General indexes (especially for orders)
k, l	Indexes for locations
r, s	Indexes for vehicles
	Problem input
$\mathcal{N}, n$	Set of orders, number of orders
$\mathcal{M}, \mathcal{M}_+, m$	Set of locations, set of locations without the depot, number of locations
$\mathcal{V}, v$	Set of vehicles, number of vehicles
$\mathcal{Z}, z$	Set of locker sizes, number of locker sizes
s_i, w_i	Size and weight of order i
p_i, u_i	Intended location and type (delivery/pickup) of order i
$a_{k,i}$	Available number of lockers of size i at location k
$d_{k,l}$	Travel distance from location k to location l
$t_{k,l,i}$	Travel time from location k to location l at time i
P, S, B	Vehicle parking/departing time, order service time, delivery start time
C	Vehicle capacity
	Primary solution and Discrete Event Simulation state
E, E_r, e	Solution, sequence of events of vehicle r, event
$T(e), L(e)$	Start time of event e, vehicle load after event e
$attr(e)$	attribute $attr$ (e.g. duration, delivered order weight etc.) of event e
$A_{k,i,t}$	Number of remaining lockers of size i at location k at time t
$A_{k,i}$	Current number of remaining lockers of size i at location k
$order[r], next[r]$	Next order to be processed by vehicle r, index of that order in π_r
$load[r], loc[r]$	Current load of vehicle r, current location of vehicle r
$visited[r]$	List of alternative locations visited by vehicle r since last successful order delivery
D_Σ, T_{max}	Total traveled distance, maximum delivery time
	Solution representations
π	Solution for Order representation
σ	Solution for other representations (mainly Grouped Order representation)
	Miscellaneous
$\pi(i)$, len(π)	i-th element of sequence π, length of sequence π

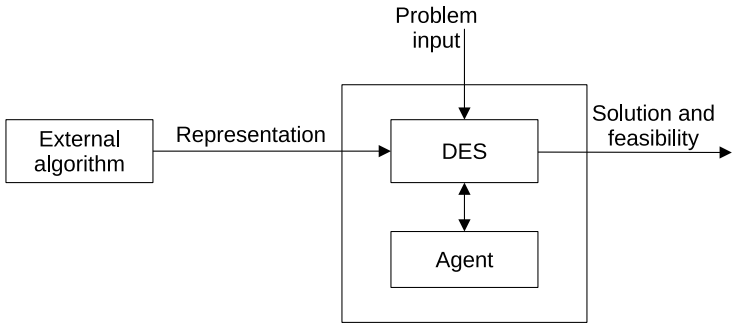


Figure 2: DES in the context of the three-tier problem formulation

3.3. Input data

The input data of the problem are as follows. Let $\mathbb{N}_+$ denote the set of positive integers. Let $\mathcal{N} = \{1, 2, \ldots, n\}$ be the set of $n \in \mathbb{N}_+$ orders. Similarly, let $\mathcal{M} = \{0, 1, 2, \ldots, m\}$ be a set of $m + 1$ locations, $m \in \mathbb{N}_+$. The locations 1 through m represent delivery and pick-up points (parcel locker locations), while location 0 represents the depot. For the sake of brevity, the set of locations without the depot (i.e. $\mathcal{M} \setminus \{0\}$) will be denoted as $\mathcal{M}_+$. Next, let $\mathcal{V} = \{1, 2, \ldots, v\}$ be a set of $v \in \mathbb{N}_+$ vehicles, $v \leqslant n$. Finally, let $\mathcal{Z} = \{1, 2, \ldots, z\}$ be a set of parcel locker sizes.

For each order $i \in \mathcal{N}$, we define its size $s_i \in \mathcal{Z}$, weight $w_i \in \mathbb{N}_+$, delivery/pickup location (place) $p_i \in \mathcal{M}_+$ as well as type $u_i \in \{0, 1\}$. Type $u_i = 0$ indicates a delivery order and $u_i = 1$ indicates a pickup order. For example, if $s_i = 2$ then order i can be placed in a parcel locker of size 2 or 3, but not of size 1. For a pickup order its size indicates the size of the parcel locker it is picked from, i.e. servicing a pickup order i will increase the number of the currently available parcel lockers of size s_i.

For each non-depot location $k \in \mathcal{M}_+$ and locker size $i \in \mathcal{Z}$, we define the number of lockers of that size available at that location as $a_{k,i} \in \mathbb{N}_+$. Moreover, for each pair of locations $k, l \in \mathcal{M}$ we define the shortest vehicle travel distance from k to l as $d_{k,l}$. It should be noted that locations and their distances do not form a metric space as the symmetry is not guaranteed i.e. it is possible that for some k and l:

$$d_{k,l} \neq d_{l,k} \,. \tag{1}$$

The above might be caused by one-way streets, such that the route k to l is simple, but the route l to k must go through other locations. However, the rest of the metric space properties are guaranteed, i.e.:

$$d_{k,k} = 0, \qquad \forall k \in \mathcal{M}, \tag{2}$$

$$k \neq l \implies d_{k,l} > 0, \qquad \forall k, l \in \mathcal{M}, \tag{3}$$

$$d_{k,l} \leqslant d_{k,a} + d_{a,l} \qquad \forall a, k, l \in \mathcal{M}. \tag{4}$$

Similarly to distance, we want to define the travel time between the locations k and l. The travel time is naturally a function of distance, however, unlike distance it also depends on road conditions, most notably traffic. Thus, we define the travel time from location k to l as $t_{k,l,i}$, where i is the time at which the travel takes place.

Next, let $S \in \mathbb{N}_+$ be the time taken to service (deliver/pick-up) an order after arriving at the location. Similarly, let $P \in \mathbb{N}_+$ indicate the time it takes for the vehicle to park when it arrives at the location. We assume that the departure time when leaving the location is also P. Let $B \in \mathbb{N}_+$ be the earliest time vehicles can leave the depot. Finally, let $C \in \mathbb{N}_+$ be the vehicle capacity. We assume that $w_i \leqslant C$ for all $i \in \mathcal{N}$.

3.4. Solution

The (direct) solution for our delivery process can be defined as a sequence of events (actions) to carry out for each vehicle (courier). This approach allows to more easily model the hard constraint that requires the couriers to decide on alternative locations on-site, while also being extensible to future modifications (especially creating a full stochastic model). Two types of events are possible (1) travel to another location, (2) service (delivery or pickup) of an order. Thus, we can define the solution as a sequence $E = (E_1, E_2, \ldots, E_v)$ of v elements (v-tuple). The element E_r, $r \in \mathcal{V}$ is a sequence of actions for vehicle r. The length of E_r and its i-th element are denoted by $len(E_r)$ and $E_r(i)$, respectively. Thus, vehicle r has $len(E_r)$ events. The travel event e must store the locations between which the travel occurs, which will be denoted by $from(e)$ and $to(e)$. The service event stores the order to be serviced denoted $order(e)$. For delivery orders, we also need to store the locker size in which the order will be placed, denoted $size(e)$. For pickup orders, this value does not matter and can be anything. Moreover, each event e stores its type denoted by $type(e)$, which is 0 for travel events and 1 for order service events.

As an example, let us consider 3 vehicles. Vehicle 1 travels from location 0 (the depot) to location 1 to service orders 5 and 2 and then travels to location 2 to service order 4 and then travels back to the depot. Vehicle 2 does not travel or service any orders. Vehicle 3 travels from 0 to 4 to service orders 8, 3 and 6, then travels to 3 to service 7 and 1 and then travels back to 0. For simplicity, let us also assume that vehicle 1 placed all of its orders in lockers of size 1 and vehicle

2 placed all orders in lockers of size 3. In this case, the solution is:

$$E = (E_1, E_2, E_3), \tag{5}$$

$$E_1 = \big(E_1(1), E_1(2), E_1(3), E_1(4), E_1(5), E_1(6)\big), \tag{6}$$

$$E_2 = (), \tag{7}$$

$$E_3 = \big(E_3(1), E_3(2), E_3(3), E_3(4), E_3(5), E_3(6), E_3(7), E_3(8)\big). \tag{8}$$

The values stored for each event are summarized in Table 3. As such, to specify the solution we specify E, including the number of events for each vehicle and the data each event stores. Note that due to the possibility of alternative delivery locations, the total number of events could be as high as $n(m+1)$.

Table 3: Events data for the example solution

Event e	$type(e)$	$from(e)$	$to(e)$	$order(e)$	$size(e)$
$E_1(1)$	0	0	1		
$E_1(2)$	1			5	1
$E_1(3)$	1			2	1
$E_1(4)$	0	1	2		
$E_1(5)$	1			4	1
$E_1(6)$	0	2	0		
$E_3(1)$	0	0	4		
$E_3(2)$	1			8	3
$E_3(3)$	1			3	3
$E_3(4)$	1			6	3
$E_3(5)$	0	4	3		
$E_3(6)$	1			7	3
$E_3(7)$	1			1	3
$E_3(8)$	0	3	0		

3.5. Constraints

In this subsection, we will establish the constraints that the solution E must satisfy for E to be feasible. This means that constraints are restrictions on how DES is allowed to evolve (i.e. generate and process events). Thus, DES needs to be implemented such that for a given representation it either produces a feasible E or reports infeasibility. Due to their complexity, the constraints are described mainly for the sake of completing the formulation, and their actual enforcement will be shown with the details of DES in Section 4.

First, each order has to be assigned to exactly one vehicle. In other words, for each order $i \in \mathcal{N}$ there must be exactly one $r \in \mathcal{V}$ and $j \in \{1, 2, \ldots, len(E_r)\}$ such that

$$type\big(E_r(j)\big) = 1 \ \wedge \ order\big(E_r(j)\big) = i. \tag{9}$$

Thus, order i is serviced during event $E_r(j)$. The effect is similar to partition of a set, except we are dealing with sequences instead of sets and E_r are allowed to be empty and allowed to contain other elements than service events (i.e. travel events).

Next, each non-empty vehicle r has to start and end its tour at the depot and had to visit at least one non-depot location. The constraints for this are as follows:

$$type\big(E_r(1)\big) = 0, \ \ from\big(E_r(1)\big) = 0, \ \ to\big(E_r(1)\big) \neq 0, \quad \forall r \in \{a \in \mathcal{V} : len(E_a) > 0\}, \tag{10}$$

$$type\big(E_r(*)\big) = 0, \ \ from\big(E_r(*)\big) \neq 0, \ \ to\big(E_r(*)\big) = 0, \quad \forall r \in \{a \in \mathcal{V} : len(E_a) > 0\}. \tag{11}$$

where $E_r(*)$ is a shorthand for $E_r\big(len(E_r)\big)$ i.e. the last element of E_r.

For any vehicle $r \in \mathcal{V}$, a travel event has to start at the location where the previous travel event ended. Thus, if $E_r(i)$ and $E_r(j)$, $j > i$ are subsequent travel events (meaning that events $E_r(a)$, $i < a < j$ are not travel events), then the constraint is:

$$from\big(E_r(j)\big) = to\big(E_r(i)\big). \tag{12}$$

Similarly, for every order servicing event e we have to ensure that the vehicle is at the correct location when servicing it. Let us consider this for pickup and delivery orders separately. For a pickup order (i.e. $type(e) = 1$ and $u_{order(e)} = 1$) the constraint is straightforward, as pickup is always done at a predetermined location. Thus, if e is a pickup servicing event and the last travel event prior to e is e' then the constraint becomes:

$$to(e') = p_{order(e)}. \tag{13}$$

For delivery orders (i.e. $type(e) = 1$ and $u_{order(e)} = 0$), the issue is more complicated. Delivery must first be attempted at the location $p_{order(e)}$ and only if there are no lockers available there at the time. Then alternative locations are considered in increasing order of distance from $p_{order(e)}$ until a location with available lockers is found. To formulate this, we need an additional notation. Let $T(e)$ be the time when the event e occurs, and $A_{k,i,t}$ be the number of lockers of size i available at the location k at time t. With this, for an order delivery event e_b by vehicle r to be feasible, there must exist in E_r a sequence of *subsequent*

events $e_1, e_2, \ldots, e_a$ such that:

$$to(e_1) = p_{order(e)}, \tag{14}$$

$$T(e_a) \leqslant T(e_b), \tag{15}$$

$$type(e_i) = 0, \qquad \forall i \in \{1, 2, \ldots, a\}, \tag{16}$$

$$type(e_i) = 1, \qquad \forall i \in \{a+1, a+2, \ldots, b\}, \tag{17}$$

$$A_{to(e_i),j,T(e_i)} = 0, \qquad \forall i \in \{1, 2, \ldots, a-1\}, j \in \{s_{order(e)}, \ldots, z\}, \tag{18}$$

$$d_{p_{order(e)},to(e_i)} \leqslant d_{p_{order(e)},to(e_{i+1})}, \qquad \forall i \in \{1, 2, \ldots, a-1\}. \tag{19}$$

In other words, in e_1 we travel to $p_{order(e)}$, e_a is the last travel event before the delivery event e_b, all events from e_1 to e_a are travel events, all locations visited in e_1 through e_{a-1} have no available lockers of any matching size ($s_{order(e)}$ or greater) and locations from e_1 through e_a are visited in order of non-decreasing distance from $p_{order(e)}$.

We will now consider the weights and vehicle capacity. At no time can the vehicle load exceed its capacity. By $L\big(E_r(i)\big)$ we denote the load of vehicle r after event $E_r(i)$. The initial load $L\big(E_r(0)\big)$ of vehicle r is thus a sum of all orders r has to deliver, which is:

$$L\big(E_r(0)\big) = \sum_{i=1}^{len(E_r)} weightDelivery\big(E_r(i)\big), \tag{20}$$

where $weightDelivery(e)$ for the event e is defined:

$$weightDelivery(e) = \begin{cases} w_{order(e)} & \text{if } type(e) = 1 \text{ and } u_{order(e)} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

Thus, $weightDelivery(e)$ counts only the weights of the delivery orders. Similarly, we can count the current load after each event:

$$L\big(E_r(i)\big) = L\big(E_r(i-1)\big) + \sum_{i=1}^{len(E_r)} weightOrder\big(E_r(i)\big), \tag{22}$$

where $weightOrder(e)$ is defined to appropriately count weights for all orders:

$$weightOrder(e) = \begin{cases} -w_{order(e)} & \text{if } type(e) = 1 \text{ and } u_{order(e)} = 0, \\ w_{order(e)} & \text{if } type(e) = 1 \text{ and } u_{order(e)} = 1, \\ 0 & \text{if } type(e) = 0. \end{cases} \tag{23}$$

Thus, this will correctly decrease the vehicle load during deliveries and increase it during pickups. Then the overall load constraint is in the form:

$$L_r(i) \leqslant C, \qquad \forall r \in \mathcal{V}, i \in \{1, 2, \ldots, len(E_r)\}. \tag{24}$$

We will now move onto time constraints. Each event takes time based on travel, parking, departure, and order service. The first event (which must be a travel event as said previously) must start at time B, when the vehicle leaves the depot:

$$T\big(E_r(1)\big) = B, \qquad \forall r \in \mathcal{V}. \tag{25}$$

Subsequent events start when the previous event ends, based on that event duration:

$$T\big(E_r(i)\big) = T\big(E_r(i-1)\big) + duration\big(E_r(i-1)\big), \quad \forall r \in \mathcal{V}, i \in \{2, 3, \ldots, len(E_r)\}, \tag{26}$$

where $duration(e)$ is:

$$duration(e) = \begin{cases} t_{from(e),to(e),T(e))} + P & \text{if } type(e) = 1 \ \wedge \ from(e) = 0, \\ t_{from(e),to(e),T(e))} + 2P & \text{if } type(e) = 1 \ \wedge \ from(e) \neq 0, \\ S & \text{if } type(e) = 0. \end{cases} \tag{27}$$

Thus, for servicing events, the time advances by S and for travel events, the time advances by departing time P (except when leaving the depot), travel time $t_{from(e),to(e),T(e)}$ and parking time P.

Finally, we consider locker numbers and sizes. First, for each location and locker size, the initial number of free lockers at time B is defined by the input of the problem:

$$A_{k,i,B} = a_{k,i}, \qquad \forall k \in \mathcal{M}_+, i \in \mathcal{Z}. \tag{28}$$

Next, when delivering an order, there has to be a free locker of appropriate size at the location of delivery. Thus, if $type(e) = 1$ and $u_{order(e)} = 0$, then we demand that:

$$A_{k,size(e),T(e)} \geqslant 1, \tag{29}$$

where k is the location where the vehicle is when performing e.

Next, when delivering an order in event e, the chosen locker size must be the smallest possible. Thus, if $type(e) = 1$ and $u_{order(e)} = 0$ then:

$$size(e) = \min\{a \in \mathcal{Z} : a \geqslant s_{order(e)} \wedge A_{k,a,T(e)} \geqslant 1\}, \tag{30}$$

where k is once again the location where the vehicle is when performing e. In other words, the chosen locker size is the first size counting from 1 that both (1) is of sufficient size of the order and (2) has free lockers available at the time.

Finally, order events have to change the current number of available lockers. Assume that e is a servicing event ($type(e) = 1$) that delivers or picks up an order at some location k from/to a locker of size i. Also, assume that the next

delivery/pickup event for this location and locker size is e'. Then it must hold that:

$$A_{k,i,T(e')} = \begin{cases} A_{k,i,T(e)} - 1 & \text{if } u_{order(e)} = 0, \\ A_{k,i,T(e)} + 1 & \text{if } u_{order(e)} = 1. \end{cases} \tag{31}$$

Thus, a delivery (pickup) order decreases (increases) the number of lockers available of that size at that location.

3.6. Goal function

The goal function can be formulated as follows. The first criterion is the sum of the distances traveled by all vehicles. Let D_r be the total distance traveled by vehicle r, which is calculated as follows:

$$D_r = \sum_{i=1}^{len(E_r)} distance\big(E_r(i)\big), \tag{32}$$

where $distance(e)$ is:

$$distance(e) = \begin{cases} d_{from(e),to(e)} & \text{if } type(e) = 0, \\ 0 & \text{if } type(e) = 1. \end{cases} \tag{33}$$

In other words, we sum all distances of travel events for the vehicle r. The total distance traveled $\Sigma D(E)$ for the solution E is thus a sum over all vehicles:

$$\Sigma D(E) = \sum_{r=1}^{v} D_r. \tag{34}$$

The second criterion is the completion time of all delivery orders, i.e. the completion time of the last delivery event (pickup events do not count) to be completed. This value, denoted $T_{max}(E)$, for solution E, is calculated as follows:

$$T_{max}(E) = \max_{r \in \mathcal{V}} \max_{i \in \{1,2,\ldots,len(E_r)\}} completed\big(E_r(i)\big), \tag{35}$$

where $completed(e)$ is:

$$completed(e) = \begin{cases} T(e) + S & \text{if } type(e) = 1 \ \wedge \ u_{order(e)} = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

The value of the goal function for the solution E is thus a pair $(\Sigma D(E), T_{max}(E))$. Since we are considering a multi-criteria optimization, we will use the dominance relation to compare solutions. The solution E is said to dominate the solution E',

denoted $E \prec E'$ if and only if

$$\Sigma D(E) \leqslant \Sigma D(E') \ \wedge \ T_{max}(E) \leqslant T_{max}(E'), \tag{37}$$

$$\Sigma D(E) < \Sigma D(E') \ \vee \ T_{max}(E) < T_{max}(E'). \tag{38}$$

In other words, E dominates E' if and only if E is not worse on either criterion than E' and is better on at least one criterion. The solution that dominates is always better (thus the use of $\prec$, which is analogous to $<$). Please note that for a given pair of solutions E and E' their relation is always one of the three: (1) $E \prec E'$, (2) $E' \prec E$, or (3) neither dominates the other, which means that the solutions are Pareto-equivalent, denoted $E \equiv E'$.

The goal is to find a set of solutions $\{E_1^*, E_2^*, \ldots, E_h^*\} \subseteq E_{feas}$, such that:

$$E \nprec E_i^*, \qquad \forall E \in E_{feas}, \ i \in \{1, 2, \ldots, h\}, \tag{39}$$

where E_{feas} is a set of feasible solutions i.e. solutions meeting constraints from Section 3.5. In other words, we want a set of solutions such that there does not exist a feasible solution E that would dominate any of the solution in our set. The set $\{E_1^*, E_2^*, \ldots, E_h^*\}$ is called the Pareto front. The size of the front h depends on the problem instance.

4. Order representation

The solution presented in Section 3 is very impractical due to the number of constraints that are not easy to formulate in tools like Gurobi. Thus, we will introduce a different solution representation to simplify the problem and reduce both the solution size and the number of infeasible solutions. This solution representation will be used as part of the input to DES, serving as a guideline on which DES has to construct the full feasible solution E or report the infeasibility if it is impossible to construct a feasible E from this representation.

Since this representation considers individual orders, we will call it Order Representation or O-representation for short. Solutions based on O-representation will be called O-solutions. We define the O-solution $\pi = (\pi_1, \pi_2, \ldots, \pi_v)$ as a v-tuple, where π_r is a sequence of orders to be serviced by vehicle r. π Let us consider the example solution E from (5) and Table 3. For this solution, the corresponding O-representation π would be:

$$\pi = \big((5, 2, 4), (), (8, 3, 6, 7, 1)\big). \tag{40}$$

This is similar to the Giant Tour Representation (GTR) commonly seen in VRP, except that it is formulated as a sequence of sequences instead of a single sequence with delimiting "zeroes". We have chosen it this way because vehicles can affect

each other and such a representation is simpler to decode. The number of possible solutions is equal to the n-permutation of $n + v - 1$ denoted by $P(n + v - 1, n)$:

$$P(n + v - 1, n) = \frac{(n + v - 1)!}{(v - 1)!} \in O\big((n + v)!\big). \tag{41}$$

This can be shown through a simple transformation to GTR, which results in:

$$(5, 2, 4, 0, 0, 8, 3, 6, 7, 1), \tag{42}$$

with zeroes indicating separation between vehicles. The number of solutions is the number of permutations of the above sequence (which is $(n + v - 1)!$), but since zeroes are indistinguishable, we remove redundancy by dividing this number by $(v - 1)!$.

We will now define a *well-formed* O-solution. π is well-formed if and only if:

$$len(\pi) = v, \tag{43}$$

$$\sum_{r=1}^{v} len(\pi_r) = n, \tag{44}$$

$$\pi_r(i) \in \mathcal{N}, \qquad \forall r \in \mathcal{V}, i \in \{1, 2, \ldots, len(\pi_r)\}, \tag{45}$$

$$r \neq s \ \vee \ i \neq j \implies \pi_r(i) \neq \pi_s(j), \quad \forall r, s \in \mathcal{V}, i \in \{1, 2, \ldots, len(\pi_r)\}, \ j \in \{1, 2, \ldots, len(\pi_s)\}. \tag{46}$$

This is equivalent to constraint (9). Thus, if we ensure that we only consider well-formed O-solutions, we are ensuring those constraints. However, π by itself does not ensure that the remaining constraints are met.

4.1. DES-based solution decoding

We will now describe the details of the DES system mentioned earlier. DES will take a well-formed O-solution π and problem data as input and will either construct a feasible schedule $E(\pi)$ corresponding to π and compute the value of the goal function $(\Sigma D(\pi), T_{max}(\pi))$, or report infeasibility. The outline of the DES system is shown in Figure 3. The system consists of a few parts. First, the system clock and state are initialized and the first events are generated. Then the system enters a loop, advancing time, fetching the next event, and processing it. During processing, the necessary decisions are performed by the agent and the system state is updated. New events are also added if necessary. After all events have been processed, the simulation stops and the goal function value is returned.

DES will store events using a priority queue Q. As explained in Section 3, event e is described by values such as $type(e)$, $from(e)$ etc. In addition, we will add the attributes $vehicle(e)$ and $time(e)$ to describe the vehicle that performs

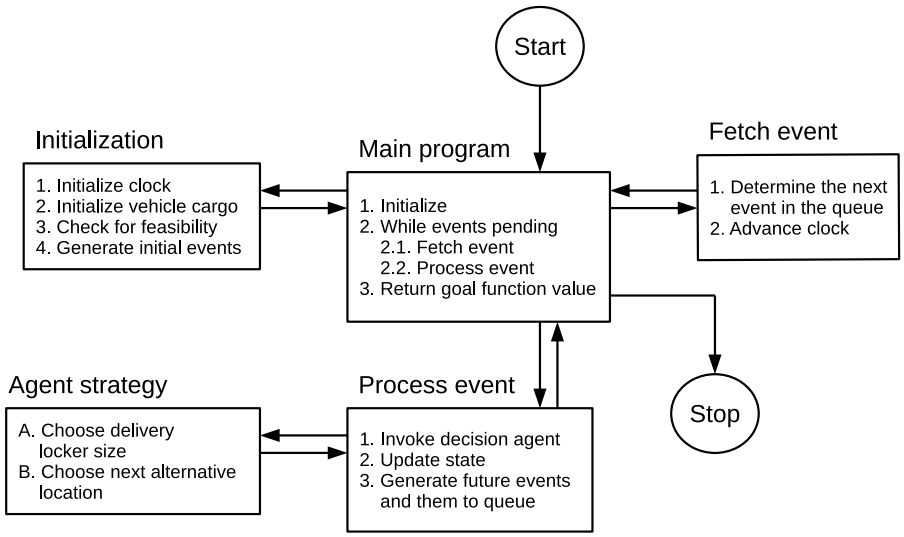


Figure 3: Outline of a Discrete Event Simulation

the event and the time when the events start. The latter will also be the priority of Q, i.e. events are processed in the order of increasing event times. Adding a new event e to Q depends on the type of e. For travel event, it will take the form:

$$Q.\text{pushTravelEvent}(time(e), from(e), to(e), vehicle(e)), \tag{47}$$

while for servicing event it is:

$$Q.\text{pushServicingEvent}(time(e), order(e), size(e), vehicle(e)). \tag{48}$$

Finally, the procedure must have access to the current state of each vehicle. Namely, the arrays *next*, *order*, *load*, *loc*, and *visited* hold the index of the current order, the current order number, the current load, the current location, and the list of alternative locations visited (since the last successful delivery), respectively. Each of these arrays will contain v elements. Thus, for example, $load[r]$ is the current load of the vehicle r. Likewise, the current number of free lockers of size i at location k will be denoted by $A_{k,i}$ (this is the "current" variant of $A_{k,i,t}$).

The outline of the decoding algorithm is shown in Algorithm 1. This code simply initializes the system and processes all events until the event queue is empty, at which point the value of the goal function is returned. The initialization procedure is shown in Algorithm 2. In lines 1–3, we set the current time to B, reset the goal function values to 0 and create an empty queue Q. In lines 4–6, the current number of $A_{k,i}$ available lockers is set for each location and size based on $a_{k,i}$. In lines 7–17, we prepare the initial state for all non-empty vehicles. Namely,

in lines 9–11, we compute the initial load. If this load exceeds vehicle capacity, infeasibility is detected in lines 12–13. Next, in lines 14–16, we set the index in π_r of the next order, the order itself, and we clear the set of visited alternative locations. Finally, in line 17 we add the first event for vehicle r, which is a travel event at time T from depot (0) to the intended location of the first order ($p_{order[r]}$).

Algorithm 1: DES procedure

Input: Problem instance, O-solution π
Output: Solution and goal function values or `INFEASIBLE`

```
1 initialize();
2 while Q.size() > 0 do
3     processEvent(Q.pop());
4 return E, (ΣD, T_max);
```

Algorithm 2: DES initialize() subprocedure

```
1  T ← B;                                              // set initial time
2  ΣD ← 0;  T_max ← 0;                          // reset goal function values
3  Q ← ∅;                                          // initialize event queue
4  for k ← 1 to m do                      // initialize locker availability
5      for i ← 1 to z do
6          A_{k,i} ← a_{k,i};
7  for r ← 1 to v do                                          // for each...
8      if len(π_r) > 0 then                          // ...non-empty vehicle
9          load[r] ← 0;
10         for i ← 1 to len(π_r) do                  // compute initial load
11             load[r] ← load[r] + w_{π_r(i)}(1 − u_{π_r(i)});
12         if load[r] > C then   // infeasible if initial load exceeds
               capacity
13             return INFEASIBLE;
14         next[r] ← 1;          // mark the first order in vehicle...
15         order[r] ← π_r(next[r]);          // ...as the next to service
16         visited[r] ← {};             // clear the list of alternative
               locations
17         Q.pushTravelEvent(T, 0, p_{order[r]}, r);  // add vehicle first event
               to queue
```

The subprocedure for processing events is shown in Algorithm 3. In lines 1–3, we add the event to E, retrieve the time and vehicle for the event. In lines 4–8, we process travel events. First, in line 5, we add the travel distance to the goal function. If the vehicle tour continues (i.e. target location is not the depot), we

change the current location in line 6. Next, we would like to service the order, but we do not know which locker size to choose. Moreover, we do not know whether an alternative location needs to be visited. Thus, the agent is invoked to apply the required strategy in line 8.

Algorithm 3: DES *processEvent*() subprocedure

```
   Input: Event e
1  E_vehicle(e).add(e);                          // add event to solution E
2  T ← time(e);                                  // set current time to event time
3  r ← vehicle(e);                               // set shorthand for current vehicle
4  if type(e) = 0 then                                                 // travel event
5      ΣD ← ΣD + d_from(e),to(e);                // update total travel distance
6      if to(e) ≠ 0 then                                  // if not completing tour
7          loc[r] ← to(e);                               // update vehicle location
8          decideByAgent(T + duration(e), r);            // invoke strategy agent
9  else                                                                // service event
10     i ← order(e);                                 // set current order shorthand
11     j ← size(e);                                  // set order locker size shorthand
12     load[r] ← load[r] + weightOrder(e);           // update vehicle load
13     A_loc[r],j ← A_loc[r],j + (2u_i − 1);         // update free lockers
14     if load[r] > C then                           // infeasible if load exceeds capacity
15         return INFEASIBLE;
16     if u_i = 0 and T + duration(e) > T_max then   // update maximum...
17         T_max ← T + duration(e);                  // ...delivery time
18     next[r] ← next[r] + 1;                        // proceed to the next order
19     visited[r] ← {};                              // clear list of visited alternative
        locations
20     if next[r] > len(π_r) then                    // tour ends, add travel back to the
        depot
21         Q.pushTravelEvent(T + duration(e), loc[r], 0, r);
22     else                                                            // tour continues
23         order[r] ← π_r(next[r]);
24         if loc[r] ≠ p_order[r] then               // mismatched location, create
            travel event
25             Q.pushTravelEvent(T + duration(e), loc[r], p_order[r], r);
26         else                                      // correct location, invoke strategy agent
27             decideByAgent(T + duration(e), r);
```

In lines 9–27 servicing events are processed. First, we retrieve the order and locker size to use (lines 10–11). In lines 12–13, we update the vehicle load and the number of lockers available. Due to the term $2u_i - 1$, this single part works

both for delivery and pickup orders. Next, if the vehicle load exceeds capacity, the solution is infeasible (lines 14–15). For delivery orders, we also update the maximum delivery time (lines 16–17). In lines 18–19, we update the index $next[r]$ to point to the next order in π_r and reset the set of visited alternative locations to empty (since order servicing just succeeded). If all orders for this vehicle were serviced, we schedule the last travel event to the depot (lines 20–21). Otherwise (there are still orders to service for this vehicle), we update the current order (line 23) and check if we are at the correct location. If not, we schedule a travel event (lines 24–25). Otherwise, we invoke the agent to decide on how the order should be handled (lines 26–27).

Finally, the agent strategy procedure is shown in Algorithm 4. In lines 2–17 we consider delivery orders. First, we check if there are lockers available, starting with the smallest size possible for this order (lines 3–6). If an appropriate size is

Algorithm 4: DES decideByAgent() subprocedure

```
   Input: Time t, vehicle r
1  i ← order[r];                                  // set current order shorthand
2  if u_i = 0 then                                                 // delivery decision
3      for j ← s_i to z do          // try to find smallest fitting locker size...
4          if A_{k,j} > 0 then      // ...if found, schedule delivery event and return
5              Q.pushServicingEvent(t, i, j, r);
6              return;
7      visited[r] ← visited[r] ∪ loc[r];          // locker not found, update visited locations
8      bestDist ← ∞;                  // find closest non-visited location
9      bestLoc ← 0;
10     for l ← 1 to m do
11         if l ∉ visited[r] and d_{p_i,l} < bestDist then
12             bestDist ← d_{p_i,l};
13             bestLoc ← l;
14     l ← bestLoc;
15     if l = ∞ then  // if all locations were visited, solution is infeasible
16         return INFEASIBLE;
17     Q.pushTravelEvent(t, loc[r], l, r);        // schedule travel event to alternative location
18 else                                                              // pickup decision
19     Q.pushServicingEvent(t, i, s_i, r);          // schedule pickup event
```

found, we schedule a servicing event at time time t (passed to the procedure) for vehicle r, order i and locker size j and exit the procedure. If no locker is available, we need to find an alternative location, which is done in lines 7–17. We first mark the current location as visited and search through all non-visited locations to find the one closest to the originally intended location p_i. If no such location is found, the solution is infeasible (lines 15–16). Otherwise, a travel event is scheduled (line 17). Finally, in lines 18–19 we consider pickup orders, for which decision is obvious and a servicing event is scheduled.

It is easy to see that for a well-formed O-solutions, such a decoding procedure will result in a feasible solution, except if there is not enough lockers in the system to fit all deliveries or if the vehicle capacity is exceeded. Regarding the time computational complexity, the function to compute the goal function in worst-case runs in time $O(nm(m \log m + z \log v))$. This is under the assumption that the priority queue Q is implemented with a heap and *visited* is implemented with a binary search tree. Compared to a standard VRP this is a high complexity, especially with regards to the number of locations m. However, the worst-case scenario is unrealistic. In practice, the number of different locker sizes is a small constant and it is very unlikely that more than a constant number of alternate locations will have to be checked per order. In such a typical-case scenario the running time complexity is $O(nm + n \log v + mz)$.

5. Alternative representations

The O-representation shown in the earlier section seems natural and allows to consider a wide range of potential solutions. However, such a representation may be impractical. In particular, it is easy to notice that in many cases a vehicle transports multiple orders that are meant for the same location. It seems logical that such orders should be grouped together as this avoids having the same vehicle visit the same location twice, saving up distance and time. In this section, we propose three alternative representations based on the idea of grouping orders. We will show that such representations can vastly reduce the size of the solution space but do not guarantee optimality for the considered problem.

5.1. Grouped Order representation

Let us start with the alternative representation that we will call the Grouped Order representation or GO-representation for short. GO solution is a tuple $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_v)$, where $\sigma_r = (\sigma_r^1, \sigma_r^2, \ldots, \sigma_r^g)$ represents the vehicle r containing g groups. Each group σ_r^i is a sequence of orders with the same location. Orders intended for the same location can still be spread over many vehicles, but a single vehicle r contains at most one group intended for that location. For

example, let us consider a problem instance with $n = 3$, $v = 3$ just like the one shown in (40), but with $p_1 = p_2 = 1$, $p_3 = p_4 = p_5 = 2$, $p_6 = p_7 = p_8 = 3$. A GO solution for this instance could be

$$\sigma = (\sigma_1, \sigma_2, \sigma_3) = \big((\sigma_1^1, \sigma_1^2), (), (\sigma_3^1, \sigma_3^2, \sigma_3^3)\big) \tag{49}$$

$$= \Big(\big((4, 5), (2)\big), (), \big((6, 7, 8), (3), (1)\big)\Big). \tag{50}$$

A GO-solution σ can be transformed into an O-solution π by joining the groups together for each vehicle. For example, the GO solution (49) can be transformed into the O solution:

$$\pi = \big((4, 5, 2), (), (6, 7, 8, 3, 1)\big). \tag{51}$$

A reverse transformation from π to σ is also possible by appropriately rearranging orders in π_r such that all orders with the same location are grouped together. However, an issue arises: each group in an O-solution is a sequence, and thus needs to be sorted in some way. There are many possibilities, each with their own advantages and disadvantages. Those possibilities are, among others:

1. Preserve the order from π, which is the easiest and fastest (does not require sorting).
2. Shuffle each group at random.
3. Sort each group according to order indexes in $\mathcal{N}$.
4. Sort each group such that delivery orders appear first.
5. Sort each group such that pickup orders appear first.

Options 4 and 5 are interesting. Option 4 works better at minimizing order completion times (since pickups do not affect this criterion, only deliveries). On the other hand, option 5 allows to first empty the parcel lockers before inserting orders into them, potentially generating more feasible solutions. For now, we assume that the sorting within a group is arbitrary unless otherwise stated.

The decoding procedure for GO-solutions is the same as for O-solutions, except that we first transform GO-solution into a corresponding O-solution. The resulting procedure is slightly slower than for O-solutions, but the computational complexity remains the same, since transforming σ into π takes time $O(n + v)$.

It is easy to see that for a given instance there are no more GO-solutions than O-solutions. For example, assume that $\pi = \big((1, 2, 3, 4, 5, 6)\big)$, where orders 1, 2 and 3 belong to one location, while orders 4, 5 and 6 belong to another. For this, there are $6! = 720$ possible O-solutions, while there are only two GO-solutions, assuming a single in-group sorting method. Even if we consider all possible sortings, there are only 72 GO-solutions.

In practice, we expect GO-representation to do better than O-representation on average due to a smaller solution space and the positive effect of orders being grouped together. First, we will show some clear advantages of the GO-representation over the O-representation. We will now show a simple result of orders being grouped in the absence of alternative locations (i.e. order i is always delivered to location p_i).

Property 1. *Let π be an O-solution and σ be a GO-solution corresponding to π such that no alternative location is visited while decoding either of them. Then $\Sigma D(\sigma) \leqslant \Sigma D(\pi)$.*

Proof. Consider any vehicle r. In both π and σ this vehicle contains the same set of orders (just possibly arranged differently), which we can denote $\mathcal{N}_r \subseteq \mathcal{N}$. Similarly, the set of intended locations for this vehicle is the same for both solutions:

$$\bigcup_{i=1}^{len(\pi_r)} \{p_{\pi_r(i)}\} = \mathcal{M}_r \subseteq \mathcal{M}_+. \tag{52}$$

For each vehicle r three cases are possible. The first is that $|\mathcal{N}_r| = |\mathcal{M}_r|$, meaning that each order is for a different location. Thus, vehicle r has the same route in both σ and π, which means that $D_r(\sigma) = D_r(\pi)$.

For the second and third cases, we have $|\mathcal{N}_r| > |\mathcal{M}_r|$. The second case is when the orders in π are arranged incidentally such that they are fully grouped. While incidental, it still results in the same route for vehicle r in both solutions and thus $D_r(\sigma) = D_r(\pi)$.

The last case is when $|\mathcal{N}_r| > |\mathcal{M}_r|$, but orders in π are not fully grouped, i.e., at least two orders in π_r for location k are separated by a sequence of orders with locations different from k. In this case, σ_r has the same tour as in the earlier cases i.e. the tour has the form of:

$$0 \to k_1 \to k_2 \to \ldots \to k_{|\mathcal{M}_r|} \to 0, \tag{53}$$

where k_1 to $k_{|\mathcal{M}_r|}$ denote the subsequent locations. However, due to Dirichlet's box principle, π_r has to repeat at least one location, creating tours in the form:

$$\begin{aligned} 0 \to k_1 \to k_2 \to \cdots \to k_{a-1} \to k_a \to k_b \to k_{a+1} \to k_{a+2} \\ \to \cdots \to k_{|\mathcal{M}_r|} \to 0, \end{aligned} \tag{54}$$

where $k_b \in \mathcal{M}_r$ is the repeated location inserted between locations k_a and k_{a+1}. Thus, the subroute $k_a \to k_{a+1}$ is replaced by $k_a \to k_b \to k_{a+1}$. Due to (4), we know that the former tour cannot be longer than the latter and thus $D_r(\sigma) \leqslant D_r(\pi)$. In all of the above cases, we get $D_r(\sigma) \leqslant D_r(\pi)$. Since ΣD is the sum of D_r over all vehicles, we ultimately get $D_r(\sigma) \leqslant D_r(\pi)$.

The above reasoning works if a single location is visited twice. However, when there are more locations that are repeated (or are repeated more than once), then the reasoning can be applied multiple times, each time showing that the "reduced" tour is not longer than the "unreduced" tour. □

We can expect similar results regarding the maximum order delivery time, at least under some typical conditions.

Property 2. *Let π be an O-solution and σ be a GO-solution corresponding to π such that no alternative location is visited while decoding either of them. Also, let $t_{k,l,i} = c \cdot d_{k,l}$, where $c > 0$ is some constant and $u_i = 0$ (i.e. all orders are deliveries). Then $T_{max}(\sigma) \leqslant T_{max}(\pi)$.*

Proof. The reasoning is similar to Property 1. If all orders in vehicle r are for different locations or the arrangement of orders in π_r is incidentally the same as in σ_r (cases 1 and 2 from Property 1), then the vehicle performs the same in both solutions. Due to this and travel times being directly tied to distance ($t_{k,l,i} = c \cdot d_{k,l}$), we have $T_{max}(\sigma) = T_{max}(\pi)$.

For the last remaining case, we want to consider the time the last order is delivered in π_r and σ_r. In both solutions that time is a sum of:

1. Servicing times S. Since both π_r and σ_r serve the same number of orders, this values is the same for both.
2. Parking/departure times P. Since grouping might decrease the number of travels between locations and never increases it, then the sum of P for σ_r is not higher than for π_r.
3. Travel times $c \cdot d_{k,l}$. From Property 1 we know that this sum for σ_r is always at most as large as the sum for π_r.

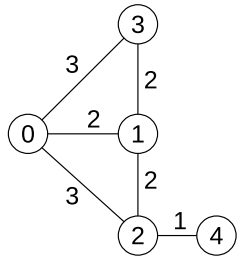
Thus, for vehicle r, the delivery time of the last order is never higher in σ_r than in π_r, which means that $T_{max}(\sigma) \leqslant T_{max}(\pi)$. □

The above properties together with reduced number of solutions indicate that under specific conditions (no alternate location visits, travel time proportional to distance), GO-solutions are superior to O-solutions. We expect this behavior to also be true in similar conditions (some alternate locations visits, some influence on road conditions on travel time). However, while GO-solutions might be better in specific cases and in the average case, there are not better in all cases. We will show this surprising fact through several properties, starting with the total distance time in the presence of alternative locations.

Property 3. *There exists PLBDP instance I and O-solution π and GO-solution σ for I such that σ corresponds to π and $\Sigma D(\sigma) > \Sigma D(\pi)$.*

Proof. The proof is based on counter-example. We will construct π and σ such that in both of them the same alternative location will have to be visited. However, in π that location would have been visited directly afterwards anyway, not affecting total distance traveled. Conversely, σ will be affected.

We construct I as follows. Let $n = 9$, $m = 4$, $v = 2$. For the orders, we set $p_1 = p_2 = p_3 = p_4 = p_5 = 1$, $p_6 = p_8 = 2$, $p_7 = 3$ and $p_9 = 4$. We set $s_i = 1$, $u_i = 0$, $w_i = 1$ for all i. For locations, we set $a_{2,1} = 1$ and $a_{k,1} = 6$ for $k \neq 2$. For $d_{k,l}$, we assume the distances shown in Figure 4. We also assume $t_{k,l,i} = d_{k,l}$ for all i. We also assume $C = 6$, $S = 1$, and $P = B = 0$, although I can be constructed for non-zero values as well.



Source \ Destination	0	1	2	3	4
0	0	2	3	3	4
1	2	0	2	2	3
2	3	2	0	4	1
3	3	2	4	0	5
4	4	3	1	5	0

Figure 4: Location distances for instance in Property 3

Next, let our σ and π be as follows:

$$\pi = \big((1, 6, 2, 3, 4, 5), (7, 8, 9)\big), \tag{55}$$

$$\sigma = \Big(\big((1, 2, 3, 4, 5), (6)\big), \big((7), (8), (9)\big)\Big). \tag{56}$$

Both vehicles have to arrive at location 2. Since $a_{2,1} = 1$, one of them vehicles will have to visit an alternate location 4 (since it is closest to 2). Vehicle 2 will visit location 4 after location 2 in both cases, so its total distance D_2 will not change. However, vehicle 1 will visit location 4 only if there is no lockers available at location 2 (which will be different for π and σ), which will lengthen its travel distance D_1.

For better understanding, we will show the time the vehicles spent on each event, allowing us to see which vehicle reaches location 2 later, meaning that it will have to visit the alternative location. Underlined numbers indicate time spent servicing orders and remaining non-underlined numbers indicate time spent traveling. The double vertical line indicates the moment the vehicles arrive at location 2. Time an event starts is obtained by summing the numbers prior to it.

Let us start with σ:

$$\text{Vehicle 1}: \quad 2 \quad \underline{5} \quad 2 \;||\; 1 \quad \underline{1} \quad 4, \tag{57}$$

$$\text{Vehicle 2}: \quad 3 \quad \underline{1} \quad 4 \;||\; \underline{1} \quad 1 \quad \underline{1} \quad 4. \tag{58}$$

Vehicle 2 goes to location 3, services an order there and goes to location 2, arriving there at time 8. Vehicle 2 leaves its order there, then goes to location 4, services its last order, and goes back to the depot. On the other hand, vehicle 1 first travels to location 1 and services 5 orders there, before going to location 2 and arriving there at time 9. Since it arrived at this location after vehicle 2 and this location had only 1 locker available, the vehicle cannot service its order there. Instead, it moves to location 4 services its order there and goes back to the depot. Since $d_{k,l} = t_{k,l,i}$, we can compute $D_1 = 9$ and $D_2 = 12$ by summing all non-underlined numbers. Thus, $\Sigma D(\sigma) = D_1 + D_2 = 21$.

We now perform the same process for π:

$$\text{Vehicle 1}: \quad 2 \quad \underline{1} \quad 2 \;||\; \underline{1} \quad 2 \quad \underline{4} \quad 2, \tag{59}$$

$$\text{Vehicle 2}: \quad 3 \quad \underline{1} \quad 4 \;||\; 1 \quad \underline{2} \quad 4. \tag{60}$$

The result is $D_1 = 8$ and $D_2 = 12$, so $\Sigma D(\pi) = 20$. □

A similar result can be shown for the maximal delivery time with the following property.

Property 4. *There exists PLBDP instance I and O-solution π and GO-solution σ for I such that σ corresponds to π and $T_{max}(\sigma) > T_{max}(\pi)$.*

Proof. The idea is that grouping orders might push pickup orders (which do not affect T_{max}) earlier, while pushing delivery orders later, negatively affecting T_{max}. With this idea, the instance I can be constructed trivially with $n = 3$, $m = 2$, $v = 1$, $p_1 = p_2 = 1$, $p_3 = 2$, $u_1 = u_3 = 0$, $u_2 = 1$ and $S = 1$. We also assume $t_{k,l,i} = d_{k,l}$ for all i. Next, ass π and σ we choose:

$$\pi = \big((1, 3, 2)\big), \tag{61}$$

$$\sigma = \Big(\big((1, 2)\big), \big((3)\big)\Big). \tag{62}$$

Only delivery orders affect T_{max}, meaning orders 1 and 3. Order 1 is delivered at the same time in both σ and π, however order 3 is delayed in σ compared to π by servicing of a pickup order 2 which takes time. Thus, $T_{max}(\sigma) > T_{max}(\pi)$. □

The above properties show that when each criterion is considered alone, we can sometimes find GO-solutions that are worse than their corresponding O-solutions. However, it is easy to notice that the decrease in $T_{max}(\pi)$ in the example from Property 4 comes at a price of increased $\Sigma D(\pi)$. Thus, there is a possibility that when both criteria are considered, σ is never worse than the corresponding π, that is, $\pi \not\prec \sigma$. This is not the case, as we show below.

Property 5. *There exists PLBDP instance I and O-solution π for that instance, such that $\pi \prec \sigma$, where σ corresponds to π according to some in-group sorting.*

Proof. We construct I as follows. We have $n = 3$ orders, $m = 2$ locations (not counting the depot), $v = 1$ vehicles, and $z = 1$ parcel sizes. For orders, we assume $s_1 = s_2 =_3= 1$, $w_1 = w_2 = w_3 = 1$, $p_1 = p_2 = 1$, $p_3 = 2$, $u_1 = u_3 = 0$, $u_2 = 1$. Considering locations, we assume $a_{1,1} = a_{2,1} = 2$, $d_{0,1} = d_{1,0} = d_{1,2} = d_{2,1} = 1$, $d_{0,2} = d_{2,0} = 2$. For vehicles, we assume $S = 1$, $P = 1$, $B = 0$, $C = 3$ and $r_i = 1$ (for all possible i).

Next, we choose $\pi = \big((1, 3, 2)\big)$. Similarly, as the corresponding GO-solution, we choose $\sigma = \Big(\big((1, 2), (3)\big)\Big)$, i.e. the sorting is preserved from π. We will now show that $\pi \prec \sigma$.

We start with the total travel distance. For π we first travel from location 0 (the depot) to 2, then to location 1 and then back to 0. The total distance is thus $d_{0,2} + d_{2,1} + d_{1,0} = 4$. Similarly, for σ, we have $d_{0,1} + d_{1,2} + d_{2,0} = 4$. Since the total distance is the same for both solutions, it is now sufficient to show that the maximum order delivery time for π is shorter than for σ.

We start with π. We travel from 0 to 1, park there, service (deliver) order 1 and depart. Then we travel to location 2, park, service (deliver) order 3 and depart. Then we travel back to location 1, park, service (pickup) order 2, depart. Finally, we travel to the depot. Thus, the times orders 1 and 3 (order 2 does not matter as it is a pickup) are serviced are:

$$d_{0,1} + P + S = 3, \tag{63}$$

$$d_{0,1} + P + S + P + d_{1,2} + P + S = 7. \tag{64}$$

Thus the completion time of all delivery orders for π is $\max\{3, 7\} = 7$.

We perform a similar reasoning for σ. We first visit location 1, servicing orders first order 1 and then order 2. Then we move to location 2, servicing order 3. Then we return to the depot. The delivery times are:

$$d_{0,1} + P + S = 3, \tag{65}$$

$$d_{0,1} + P + S + S + P + d_{1,2} + P + S = 8. \tag{66}$$

The delivery completion time for σ is $\max\{3, 8\} = 8$. Thus, $\pi \prec \sigma$. □

The above property shows that sometimes it might be beneficial to visit a parcel location twice. Moreover, this is regardless of the in-group sorting used to create σ as shown below.

Property 6. *There exists PLBDP instance I and O-solution π for that instance, such that $\pi \prec \sigma$ for any σ corresponding to π for any in-group sorting.*

Proof. We will use the same instance I and π as for Property 5. Since we already used $\sigma = \Big(\big((1,2),(3)\big)\Big)$, the only other sorting option we have is $\sigma = \Big(\big((2,1),(3)\big)\Big)$.

The total travel distance is 4 as before (since we visit the same locations in the same order), so once again we focus on maximal delivery time. We have:

$$d_{0,1} + P + S + S = 4, \tag{67}$$

$$d_{0,1} + P + S + S + P + d_{1,2} + P + S = 8. \tag{68}$$

Thus, the delivery completion time for this σ is also 8. □

We showed that some σ can be worse than their corresponding π regardless of the in-group sorting. However, it might still be possible that for every π there exists σ (which corresponds to some O-solution, not necessarily π) such that $\pi \not\prec \sigma$. This would imply that $\pi^* \not\prec \sigma^*$ and would make GO-solutions "safe" to use. We will now show that this is not the case and that GO-solutions are not guaranteed to be optimal.

Property 7. *There exists PLBDP instance I for which $\pi^* \prec \sigma^*$.*

Proof. We will once again use the same instance as in Property 5. For this instance there are 6 possible O-solutions, so we can obtain π^* by evaluating all of them. Below we enumerate those O-solutions and their corresponding goal function values in the form $\pi \to \big(\Sigma D(\pi), T_{max}(\pi)\big)$:

$$\big((1,2,3)\big) \to (4,8), \tag{69}$$

$$\big((1,3,2)\big) \to (4,7), \tag{70}$$

$$\big((2,1,3)\big) \to (4,8), \tag{71}$$

$$\big((2,3,1)\big) \to (4,11), \tag{72}$$

$$\big((3,1,2)\big) \to (4,8), \tag{73}$$

$$\big((3,2,1)\big) \to (4,9). \tag{74}$$

Thus, we obtain $\pi^* = \big((1,3,2)\big)$. Similarly, we have 4 possible GO-solutions (2 solutions times 2 possible in-groups sortings) and we enumerate them as well:

$$\Big(\big((1,2),(3)\big)\Big) \to (4,8), \tag{75}$$

$$\Big(\big((2,1),(3)\big)\Big) \to (4,8), \tag{76}$$

$$\Big(\big((3),(1,2)\big)\Big) \to (4,8), \tag{77}$$

$$\Big(\big((3),(2,1)\big)\Big) \to (4,9). \tag{78}$$

Thus, σ^* is any of the above GO-solutions except $\Big(\big((3),(2,1)\big)\Big)$. Since $(4,7) \prec (4,8)$, we conclude that $\pi^* \prec \sigma^*$. □

The above properties indicate that GO-solutions might be suboptimal in the worst case, however, the resulting solution space is still much smaller. As such, GO-representation might still obtain better solutions in the average case.

5.2. Batch representation

The idea showcased by GO-presentation might be brought further. In particular, all orders intended for a given location might be grouped together. The resulting could be called Location representation or L-representation for short. The L-solution ρ would be defined similarly to the GO-solution σ, except there is an additional restriction that all orders for a given location are placed in one group (and thus in one vehicle). As with GO-representation, the immediate potential benefit of L-representation is that the number of possible solutions decreases and cannot be higher than:

$$P(m+v-1,m) = \frac{(m+v-1)!}{(v-1)!} \in O\big((m+v)!\big). \tag{79}$$

For example, assume $n = 15$, $m = 6$, $v = 5$. Then from (41) the number of O-solutions is greater than $5.06 \cdot 10^{15}$, while the number of L-solutions from (79) is not greater than 30240. That means approximately $1.56 \cdot 10^{11}$ O-solutions per one L-solution.

However, the L-representation has one drawback: there might be instances for which there are no feasible L-solutions at all. Such instances can be trivially constructed by making the total weight of all orders higher than the total capacity of all vehicles:

$$\sum_{i=1}^{n} w_i > vC. \tag{80}$$

However, for such instances, no feasible O- or GO-solutions exist either. We will now show instances for which feasible O- and GO-solutions exist but feasible L-solutions do not.

Property 8. *There exists PLBDP instance I for which there exist feasible O-solutions and easible GO-solutions, but for which there are no feasible L-solutions.*

Proof. Instance I is constructed trivially. For example, assume $n = 2$, $m = 1$, $v = 2$, $w_1 = w_2 = 1$, $p_1 = p_2 = 1$, $C = 1$. Since groups in L-solutions cannot be split between different vehicles and have to include all orders for a given location,

the only possible L-solutions are:

$$\Big(\big((1,2)\big),\big(()\big)\Big), \tag{81}$$

$$\Big(\big((2,1)\big),\big(()\big)\Big), \tag{82}$$

$$\Big(\big(()\big),\big((1,2)\big)\Big), \tag{83}$$

$$\Big(\big(()\big),\big((2,1)\big)\Big). \tag{84}$$

All of those L-solutions are infeasible. Meanwhile the possible O-solutions are:

$$\big((1,2),()\big), \tag{85}$$

$$\big((2,1),()\big), \tag{86}$$

$$\big((),(1,2)\big), \tag{87}$$

$$\big((),(2,1)\big), \tag{88}$$

$$\big((1),(2)\big), \tag{89}$$

$$\big((2),(1)\big). \tag{90}$$

Out of those O-solutions only the first four are infeasible. The construction of the corresponding GO-solutions is left for the reader as exercise. □

The showcased drawback can be corrected by introducing a limit on the maximal group size. If the sum of weights of the orders belonging to a single location does not exceed C, then they make a single group. If the sum exceeds C, then the orders for that location are split into multiple groups. To distinguish those groups from those in the GO-representation, we will call the resulting groups "batches". Thus, the resulting representation will be called Batch representation or B-representation for short.

The first emerging issue is how to divide groups that are too large to fit into one vehicle into multiple batches. The maximal number of batches to create for a given location is easy enough to reason out. If the total weight of orders for locations k is W, then we need at least $\lceil W/C \rceil$ batches. The way of assigning orders to these batches is more difficult. An easy idea is to take orders in arbitrary order and put them into a batch as long as the total weight of the batch does not exceed C. As soon as the weight exceeds C a new batch is created. Different variants are possible, for example, limiting batch weight to some percentage of C. Since we expect the problem of too big groups to rarely occur in practice, we will use the "do not exceed" C strategy unless stated otherwise.

It is easy to prove that all B-solutions are GO-solutions. Due to this, all results from Section 5.1 apply to B-solutions, including the modified decoding procedure, as well as Properties 5, 6, and 7.

5.3. i-Limited Batch representations

As we have seen, B-representation has much smaller solution space, but due to this, it might exclude some of the high-quality solutions (including optimal solutions). On the other hand, O-representation has an unreasonably large solution space in practice. To solve this dilemma we propose a modification of the B-representation that limits the maximal number of orders in the batch to some value i. This results in a i-Limited Batch representation or i-LB-representation, for short. This idea is similar to using some percentage of C to limit the batch size as before, except the limit is directly tied to the number of orders n. The value i might be a constant or a percentage of n. This allows to exercise a limited but more precise control of the size of the solution space.

As before, it is easy to prove that all i-LB-solutions are GO-solutions and the results from Section 5.1 regarding the decoding procedure and Properties 5, 6, and 7 apply to them. It should also be noted that each batch in an i-LB-solution still contains only orders intended for one location. Thus if a given location has less than i orders, then that particular batch will be smaller. If each location has at most j orders intended for it and $j < i$, then the i-LB-representation effectively becomes the j-LB-representation.

6. Solving methods

In this section, we will describe the algorithms proposed to solve the considered problem. It should be noted that those algorithms are primarily meant to highlight the properties of the problem and the representations proposed in Section 5. However, since no other algorithms for this problem were shown before, they will also serve as state-of-the-art algorithms. For each algorithm four different variants were developed, one variant for each of the proposed representations. Both algorithms are iterative and will invoke the DES system multiple times.

6.1. Greedy heuristic

The outline of the proposed greedy heuristic for all variants is shown in Figure 5. The algorithm is iterative and works as follows.

In the first phase, the sequence of elements (orders, order groups, batches, etc.) is constructed. In the second phase, we proceed iteratively, starting from empty vehicles and trying to insert the next element of the sequence into one of

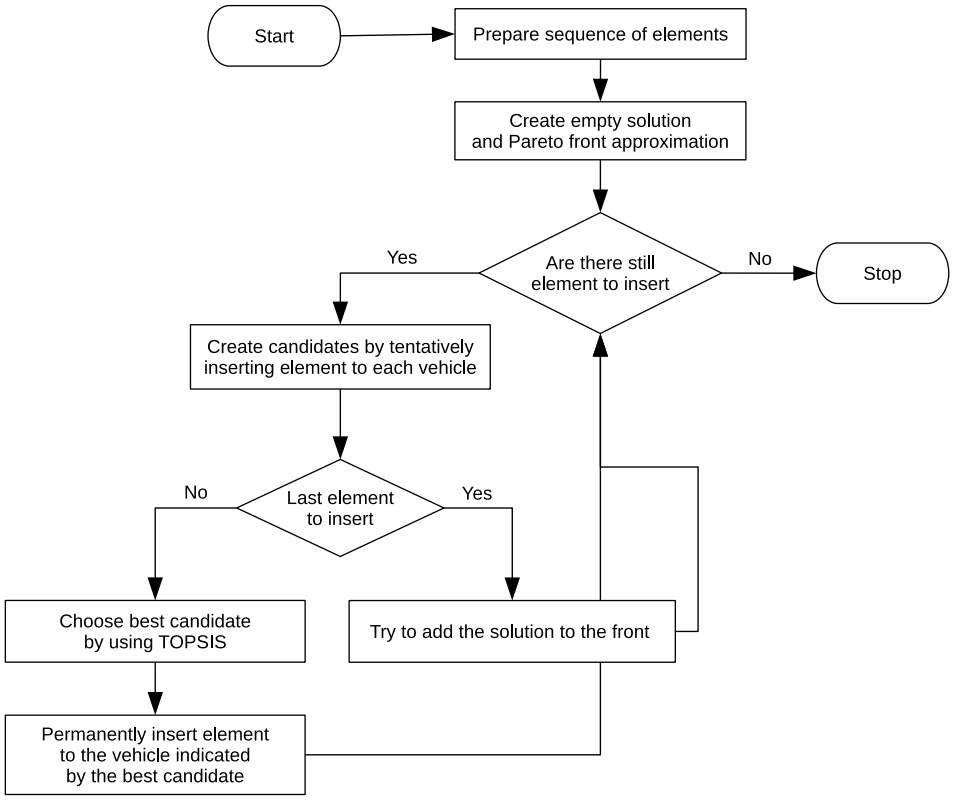


Figure 5: Outline of the greedy algorithm

the vehicles. For each vehicle, we evaluate the resulting solution and add it to the candidates if it is feasible.

During each iteration (except the last), we choose one vehicle from all candidate partial solutions. In single-criterion optimization this is straightforward. We can just compute the goal function value for each vehicle and choose the vehicle with the smallest value. However, for two-criterion optimization it is more complicated. Using domination relation is not fully preferable, as it might turn out that all candidate solutions are Pareto-equivalent. Instead, we use the TOPSIS method [6], which sorts all candidate solutions according to their distance from the ideal positive and ideal negative solutions.

The last iteration is performed differently. Instead of using TOPSIS to pick one solution from the candidates, we just remove the candidates which are dominated by at least one other candidate. The remaining candidates do not dominate each other and thus are the approximation of the Pareto front.

The greedy heuristic employing O-solution will be denoted O-greedy. For this algorithm, elements are orders. The orders are inserted to the end of the chosen vehicle. For GO-greedy elements are also orders. The order is placed at the end of the group intended for its location. If the vehicle does not have a group for this location, such a group is created and added as the last group. For the B-greedy algorithm, the elements are batches, prepared as shown in Section 5. Insertion adds the batch to the end of the batch list. We also make sure that a single vehicle has at most one batch for a given location. Finally, for i-LB-greedy we proceed in a similar manner. The orders are divided into i-limited batches. When inserting, a limited batch is added as the last batch in the vehicle.

6.2. Genetic algorithm

The Genetic Algorithm (GA) is a well-known metaheuristic based on the evolutionary processes observed in biological evolution. Similarly, to the greedy method, a different variant of GA (i.e. O-GA, GO-GA, B-GA etc.) was implemented for each of the considered representations. Unlike the greedy method, GA runs until a given time has elapsed. The general outline of GA is shown in Figure 6.

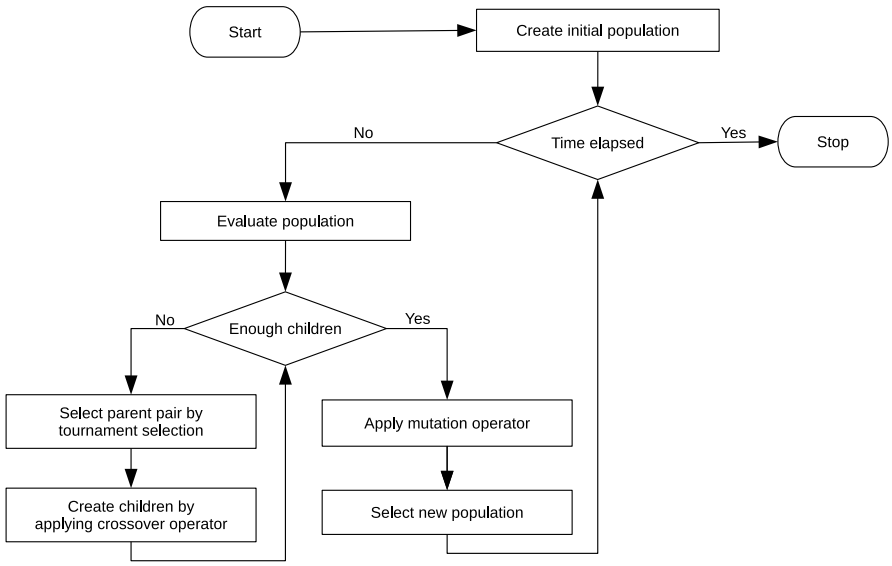


Figure 6: Outline of the Genetic Algorithm

First, an initial population of size *popSize* has to be created, which is done similarly for all variants. We start by running the greedy algorithm, obtaining a Pareto front approximation of k solutions. Then each of the *popSize* specimens

is created as a copy of a random greedy solution. However, leaving it at this would create low genetic diversity (especially if the greedy algorithm found a very small front). Thus, for each specimen, we additionally diversify it by applying a number of random insertion moves, in the same way as in the case of the mutation operator described further in this section. Once the initial population is established, we evaluate each solution in it and try to add them to the global Pareto front approximation.

Next, in each iteration (generation) of the algorithm we create a certain number of children. This number is based on the *elitism* factor described later in this section. For each pair of selected parents, we attempt to create a pair of children. However, due to the possibility of infeasible solutions, one or both children might be infeasible. Thus, the procedure for selecting parents and creating children is repeated until a sufficient number of children are created. A parent pair is selected through a tournament in the following way. Two tournaments are held, and their winners become the selected parent pair. A single tournament is held by randomly choosing a number of specimens from the population and applying the TOPSIS method to choose the best specimen, which becomes the winner.

For each pair of parents (P_1, P_2) the first child C_1 is created by applying the crossover operator i.e. $C_1 = crossover(P_1, P_2)$. The second child C_2 is created in a similar way, except that the parents swap places, i.e. $C_2 = crossover(P_2, P_1)$. The basic idea of the crossover operator is similar for all representations. First, we treat the sequence of vehicles as linear and randomly choose two crossover points. An example of this idea for the O-representation is shown in Figure 7. For the O-representation, the crossover operator then proceeds as follows. The orders from P_1 that are between the crossover point are inserted into C_1 into the same vehicles they were originally in P_1. Next, we go through P_2 and try to insert all the remaining orders into C_1. We first try to insert the order into the same vehicle in C_1 that it was originally in P_2. If the resulting solution is infeasible, we try to insert the order into further vehicles. If there are orders that were not inserted into any vehicle, the child is infeasible and is not created.

For B-representation and i-LB-representation the crossover procedure is similar, except that entire batches are inserted instead of individual orders. For

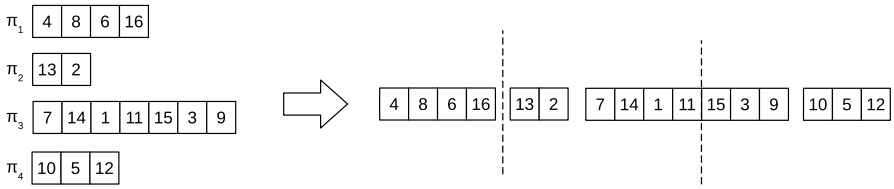


Figure 7: Illustration of crossover point for O-solution

GO-representation individual orders are inserted and we ensure that if a group for a given location already exists in that vehicle, the order is inserted into that group.

When a sufficient number of children is created, we proceed to mutation. The mutation operator is applied to each child with probability *mutProb*. For O-representation the mutation is performed by removing a random order from a random vehicle and the re-inserting it somewhere else, such that the order might end up in any vehicle and in any available position in the chosen vehicle. If the resulting solution is feasible, it replaces the original child. If it is infeasible, the original child is kept.

The mutation operator works similarly for B-representation and i-LB representation, except entire batches are removed and re-inserted. For GO-representation the process is more complex. Suppose that we want to remove from vehicle r and insert into vehicle s. If $r \neq s$, then we first remove a random order from r. If s has no group matching the location for this particular order, then we create a new group for it and insert that group in a random position among the other groups. Finally, we add the order at the end of its group. If $r = s$, then moving the individual order inside the vehicle has little effect as it would still be tied to its group. Thus, in such a case we remove an entire random group from r and re-insert it into r at random position.

As the last step in each generation, we create a new population based on the old population and the children. First, we use the elitism mechanism, where a certain percentage of best solutions from the old population are put into the new population. The best solution are, once again, determined by the TOPSIS method. The remaining specimens in the new population are children created in this generation. Thus, in each generation, we create $(1 - elitism)popSize$ children.

Regarding the parameters, we have chosen $popSize = 100$. The number of random insertions performed on the greedy solution was set to $n/10$. The size of the tournament was chosen to be equal to $\sqrt{popSize}$, rounded to the nearest integer. The parameter *mutProb* was set to 8% (slightly higher than our typical GA approach. This is done to account for some mutations creating infeasible solutions). We have chosen the value of *elitism* to be 5%. The time limit for GA was set to $n/10$ seconds.

7. Instance generation

In this section, we will describe the procedure for generating instances of the considered problem. The instance generation is partially based on real-life data. We also use the presented instance generator to propose a instance dataset for the

problem proposed in this paper. The dataset, and the generation procedure are available on Github [21].

7.1. Real-life data and general assumptions

First, eight cities in Poland were chosen as the basis for the instances. The summary of the data for each city is shown in Table 4. The population was established based on data from the Poland government agency Statistics Poland [26]. The locations of parcel lockers for a specific delivering company were collected from the OpenStreetMap database using the Overpass AP tool [16]. The number and locations of the depots for each city were retrieved from the company website. Please note that for some cities the depot is located in another city. This is also expressed in the final column, where *LL* is defined as the average distance between locker locations and *LD* is the average distance between lockers and depots. Thus, $\frac{LD}{LL} = 1.75$ means that, on average, the distance from a depot to a locker is 75% higher than distance from a locker to another locker.

Table 4: Summary of data on 8 cities used in instance generation

Id	City name	Population	Locations	Depots (city)	LD/LL
0	Warszawa	1794166	949	6 (Warszawa)	2.14
1	Wrocław	641928	375	3 (Wrocław)	1.47
2	Łódź	672185	300	2 (Łódź)	1.26
3	Częstochowa	217530	39	1 (Częstochowa)	1.38
4	Radom	209296	81	1 (Radom)	1.75
5	Inowrocław	71674	21	1 (Bydgoszcz)	17.22
6	Ostrów Wlkp.	71560	46	1 (Kalisz)	6.22
7	Suwałki	69639	29	1 (Ełk)	19.93

Distances $d_{k,l}$ were calculated using real-life data using OpenStreetMap [14]. Regarding travel times tk, l, i, we wanted to introduce the influence of traffic conditions on travel time between locations. Unfortunately, as it is not easy to obtain reliable data on traffic conditions, especially for specific cities, we have used the data for average in-day traffic speed in Beijing shown in [9], which, after averaging for both types of roads, are shown in Table 5. Thus, for our dataset, we assume that:

$$t_{k,l,i} = \frac{d_{k,l}}{V_i}, \tag{91}$$

where V_i is the average speed corresponding to time i.

Table 5: Average travel speed depending on in-day hour [km/h], based on [9]

Hour	12 AM	1 AM	2 AM	3 AM	4 AM	5 AM	6 AM	7 AM
Speed	38.9	39.5	40.2	40.9	41.0	40.0	35.6	30.9
Hour	8 AM	9 AM	10 AM	11 AM	12 PM	1 PM	2 PM	3 PM
Speed	30.2	30.8	31.1	31.7	32.4	32.1	31.2	30.9
Hour	4 PM	5 PM	6 PM	7 PM	8 PM	9 PM	10 PM	11 PM
Speed	30.2	28.4	28.4	31.1	32.5	33.6	37.0	38.0

Based on the commonly used type of parcel locker, we have assumed that there are 3 different parcel locker sizes: small, medium, and large. The default number of lockers per location is 32 (small), 29 (medium), and 18 (large).

Regarding order weights, we have assumed that $w_i \in \{1, 2, \ldots, 25\}$, in kilograms. However, we felt that uniform distribution is unrealistic and assumed that heavier orders are less probable. Thus, we assumed an exponential progression, where order weight $i + 1$ is 10% less probable than order weight i. The resulting probabilities are shown in Table 6.

Table 6: Order weight probabilities

w_i	1	2	3	4	5	6	7	8	9
$P(w_i)$	0.1000	0.0910	0.0828	0.0752	0.684	0.0622	0.0565	0.0514	0.0467
w_i	10	11	12	13	14	15	16	17	18
$P(w_i)$	0.0425	0.0386	0.0351	0.0319	0.0290	0.0264	0.0240	0.0218	0.0198
w_i	19	20	21	22	23	24	25		
$P(w_i)$	0.0180	0.0164	0.0149	0.0135	0.0123	0.0112	0.0102		

7.2. Generation procedure

Specific instances are generated through a procedure that accepts 10 parameters: *seed* (for the pseudorandom number generator), *city* (city number from Table 4), *orderRatio* (numbers of orders per city resident), *pickupRatio* (percentage of pickups among orders), *vehicleRatio* (numbers of vehicles per order), *occuppancyRatio* (probability of a locker being occupied), *vehicleCapacity*, *parkTime*, *serviceTime* and *startTime*. Then the instance is generated as follows.

Let $\mathcal{U}\{a, b\}$ and $\mathcal{U}(a, b)$ denote a discrete and continuous uniform distribution from a to b, respectively. First, we set:

$$n = round(orderRatio \cdot population(city)), \tag{92}$$

$$m = locations(city) - depots(city), \tag{93}$$
$$v = \max\{1, round(vehicleRatio \cdot orderRatio \cdot population(city))\}, \tag{94}$$
$$z = 3, \tag{95}$$
$$P = parkTime, \tag{96}$$
$$S = serviceTime, \tag{97}$$
$$B = startTime, \tag{98}$$
$$C = vehicleCapacity. \tag{99}$$

where *population*, *locations* and *depots* are taken from columns 3, 4 and 5 from Table 4 for that city. For example, for $orderRatio = 0.001$ and $city = 0$ we have $n = 1794$ orders.

Regarding locations and parcel lockers, we first need to choose our depot location. For cities 3 through 7 there is only one possibility. For cities 0 through 2 we choose the depot at random. Then the appropriate distances $d_{k,l}$ are taken from the distance matrix for the appropriate city. The values $t_{k,l,i}$ are dynamically calculated based on (91). Finally, for values $a_{k,i}$, we start with $a_{k,i} = 0$ and try to add all possible lockers (32, 29, 18, depending on the size of the locker), but each increment is made with probability *occuppancyRatio*. Thus, if $occuppancyRatio = 0.25$, then $a_{k,0}$ would be 24 instead of 32 on average.

The order weights w_i are drawn from the distribution shown in Table 6. Order sizes are drawn as $s_i \sim \mathcal{U}\{1, 3\}$. The first $round((1 - pickupRatio)n)$ orders are delivery orders, i.e. $u_i = 0$. The remaining orders are pickup orders, i.e. $u_i = 1$. Finally, order locations p_i are generated differently for delivery and pickup orders. For delivery orders, it is simply $p_i \sim \mathcal{U}\{1, m\}$. However, for pickup orders, we also have to check if the resulting location has lockers available for size s_i. If not, we check larger sizes for that location. If the pickup order fits in one, we keep the location, but change s_i accordingly. If no appropriate sizes are available for this location, we repeat the procedure for another random location until an appropriate location is found. Naturally, each generated pickup order decreases the appropriate $a_{k,i}$ value by one.

7.3. Instance dataset

The basic instance dataset was generated as follows. For each of the 8 cities we consider four values of *orderRatio*: 0.001, 0.002, 0.003, 0.004 (meaning from 0.001 to 0.004 orders per citizen). This results in problem instances with the number of orders from $n = 70$ to $n = 7177$. For each of the 32 such instance groups, we generate 10 instances, for a total dataset size of 320 instances. We assume that pickup orders constitute 25% of all orders and that 10% of lockers are unavailable (occupied). For vehicle capacity, we assume $C = 700$ kg, in line

with smaller or electric delivery cars, which are more economic and environment-friendly in city traffic. We assume that each vehicle will be 70% full on average, allowing for some flexibility. This, together with the expected value of w_i around 8, leads to 1 vehicle per 62.5 orders, i.e. *vehicleRatio* = 0.016. We have assumed that parking/departing and order servicing take 60 and 30 seconds, respectively. Finally, we assume that vehicles leave the depot at 9 am, which means that $B = 32400$ seconds. *seed* increases by 100 with each instance. The overall procedure is shown in Algorithm 5.

Algorithm 5: Basic instance dataset generation

```
1 seed ← 100;
2 for city ← 0 to 7 do
3     for orderRatio ← 0.001 to 0.004 by 0.001 do
4         for i ← 1 to 10 do
5             generate(seed, city, orderRatio, 0.25, 0.016, 0.1, 700, 60, 30, 32400);
6             seed ← seed + 100;
7     23 and 40 respectively 0.012
```

8. Computer experiments

In this section, we show the results of computer experiments researching the practical properties of the problem and the solution representations proposed in Sections 4 and 5 as well as the effectiveness of the solving methods described in Section 6. The experiments will be performed using the base instance set described in Section 7 and its derivatives.

8.1. Equipment and measures of quality

All algorithms were coded in C++ 9.4.0 and their source code is available in the GitHub repository [21]. All experiments were performed on a DGX machine with AMD EPYC 7742 3.2 GHz processor with 64 physical (128 logical) cores and 503 GiB of RAM running under Ubuntu Linux 5.4. Since GA is a probabilistic solving method, in all experiments it was run 10 times for each instance and representation.

Due to the multi-criteria nature of our problem, a single solving method provides a Pareto front approximation that may contain multiple solutions. To compare fronts, we will use the concept of a Hyper-Volume Indicator (HVI) [31]. Assume that $\mathcal{F}_1$ and $\mathcal{F}_2$ are two Pareto front approximations. Then the HVI of the front $\mathcal{F}_i$ is the area of the figure constrained by points from $\mathcal{F}_i$ and a reference (or "nadir") point Z. The coordinates of point Z are constructed from the worst

values of both considered fronts and then multiplied by some constant. In our case, we multiply them by 1.2, which is a common value. Thus, the point Z represents the "anti-ideal" solution, and the higher the HVI value (the larger the area of the figure), the better. This idea can be used for any number of front approximations and any number of optimization criteria. The concept of HVI is illustrated in Figure 8. Although HVI will be our primary measure of quality, we will also report the average values of both criteria ΣD and T_{max}.

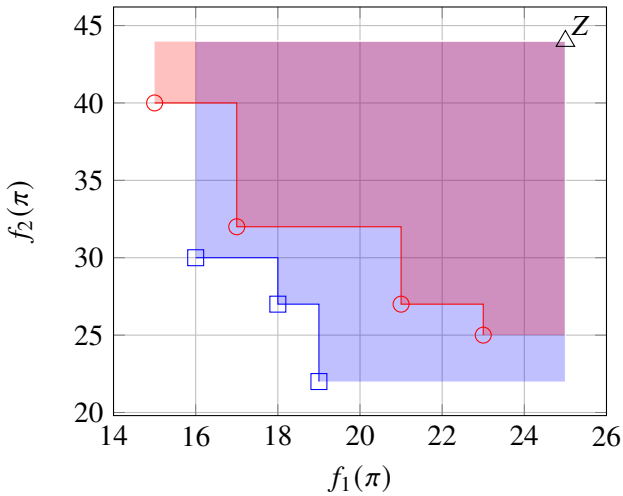


Figure 8: Illustration of HVI for two Pareto front approximations. The first front (red circles) has 4 solutions and the second one (blue squares) has 3 solutions. The colored areas represent HVI of respectively fronts. The front indicated with blue squares has higher HVI and is thus perceived as better. Point Z has coordinates $(25, 44)$ based on the worst values on each criterion (which are 23 and 40 respectively)

8.2. i-LB representation quality

In this section, we will research the effect of batch size on the quality of solutions for i-LB-representation. This is the only one of the proposed representations that requires one to choose the value of an additional parameter i. The purpose of this research is to establish both (1) the general relationship between batch size and (2) which values of i are best.

We have conducted research on standard instances using different i-LB-representations. We have used relative i values, where 100% corresponds to the maximal reasonable batch size (i.e., there are no locations with that many orders). The overall results for GA in the form of boxplots are shown in Figure 9. Please note that the HVI values are relative, i.e. HVI = 1.0 indicates the worst HVI obtained among all batch sizes for a given instance.

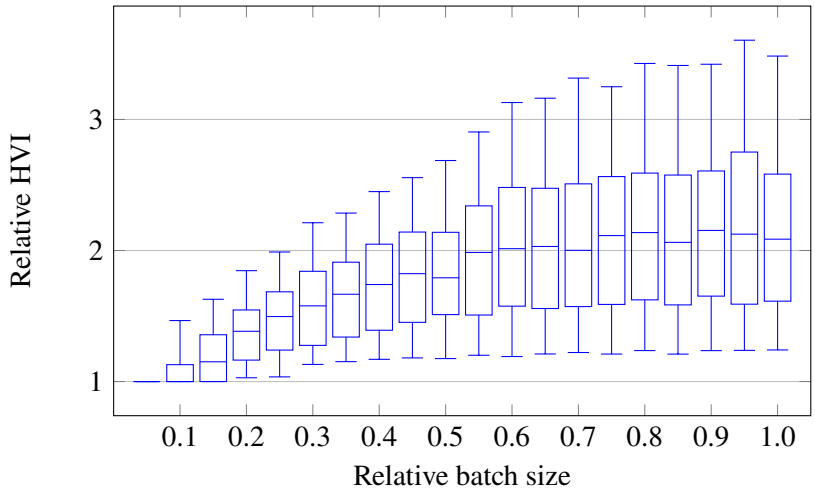


Figure 9: HVI boxplots for various relative batch sizes (boxplots are constructed from the 5th, 25th, 50th, 75th and 95th percentiles respectively)

As expected, an increase in batch size generally increases the value of HVI. The increase is significant, with the median HVI increasing more than twice when the batch size increases from 5% to 100%. One notable exception is the batch size of 95%, which seems to provide better values than 100%. A similar, though less pronounced effect was observed for the greedy algorithm. These observations are further illustrated by the average values of HVI, ΣD and T_{max} shown in Table 7.

Table 7: Aggregated results of LB-greedy and LB-GA algorithms for various batch sizes

Batch size	GA			Greedy		
	HVI	ΣD	T_{max}	HVI	ΣD	T_{max}
80%	2.16410	0.81229	0.92692	1.37161	0.94982	0.97893
85%	2.12899	0.81631	0.92877	1.33897	0.95509	0.98173
90%	2.19320	0.80968	0.92579	1.35706	0.95301	0.97928
95%	2.24152	0.80845	0.92928	1.44975	0.94166	0.97427
100%	2.15598	0.81061	0.92575	1.30659	0.96012	0.98381

With regard to the research goal, we conclude that quality increases with the percentage of the maximum batch size, but the growth stabilizes around 75%. Moreover, the i-LB-representation is the best for i equal to 95% of a maximal batch size. The differences compared to the 90% and 100% options are small,

but it should be noted that even small differences can be important as PLBD systems are periodic. As such, a small daily saving becomes significant over a longer horizon, e.g. a year.

8.3. Representation and algorithm comparison

In this section, we provide the main results of comparing GA and greedy solving methods for all proposed solution representations. The purpose of this research is to establish (1) differences between various proposed representations, (2) the effect of the number of orders on solution quality, and (3) the improvement of the GA method over the greedy algorithm.

We will start with HVI. The average results for various n ranges are shown in Table 8. Please note that $\leqslant 500$ means $250 < n \leqslant 500$ and that all HVI are relative to the worst HVI observed for a given instance. Representation-wise, we notice that O-representation performs the worst in all considered cases. Its relative quality also decreases slightly as n increases. The second-worst is GO-representation, which already provides several times higher HVIs. Similarly to O-representation, its relative quality drops significantly as n increases. Finally, B- and i-LB-representation (where i equals 95% of the maximum batch size, as shown in 8.2) consistently perform the best. Moreover, while for smaller instances their quality is similar to GO-representation, it quickly increases, providing 8 times higher HVI for the largest instances. We conclude that despite their theoretical drawbacks shown in Section 5 the alternative representations perform much better in practice. Finally, the results indicate that the i-LB-representation with 95% batch size outperforms the B-representation, although the effect is negligible.

Table 8: Aggregated relative HVI results of GA and greedy algorithms for various representation types and numbers of orders n

n	GA				Greedy			
	i-LB	B	GO	O	i-LB	B	GO	O
$\leqslant 250$	7.608	7.598	7.014	1.460	5.952	5.952	4.794	1.000
$\leqslant 500$	7.588	7.597	5.861	1.117	6.871	6.871	4.344	1.000
$\leqslant 1000$	11.772	11.706	6.451	1.039	10.823	10.822	5.068	1.000
$\leqslant 2000$	13.605	13.551	3.688	1.002	12.697	12.697	3.592	1.000
$\leqslant 4000$	18.614	18.469	3.868	1.000	17.663	17.657	3.817	1.000
$\leqslant 8000$	20.493	20.477	2.523	1.000	20.223	20.222	2.503	1.000
All	11.316	11.276	5.639	1.184	10.208	10.207	4.359	1.000

Algorithm-wise, the GA metaheuristic generally outperforms the greedy method, from 10 to 30% on average, depending on the representation. The improvement is especially visible for smaller problems. However, as problem size increases, the differences between GA and greedy algorithm become smaller and for the largest instances the GA fails to significantly improve the results provided by the greedy method. This might be partially due to the greedy algorithm having $O(n^3)$ time complexity. Since GA has a running time limit of $O(n)$, for a higher n GA has relatively less time to improve the solution provided by the greedy method, while also dealing with a larger solution space.

Since HVI does not allow us to see the values of each specific criterion, we also provide the average values for ΣD and T_{max} that are shown in Table 9. The general behavior of the representations and algorithms is similar to that in Table 8, but there are some differences. Most importantly, both criteria are improved at different rates. For O- and GO-representations, the relative values of both criteria look and improve roughly the same. This is not the case for i-LB- and B-representations, for which not only T_{max} improves significantly slower than ΣD, but its relative value is twice as high. This might be an indication of T_{max} being harder to optimize than ΣD.

Table 9: Aggregated relative ΣD and T_{max} results of GA and greedy algorithms for various representation types and numbers of orders n

n	GA				Greedy			
	i-LB	B	GO	O	i-LB	B	GO	O
				ΣD				
$\leqslant 250$	0.293	0.294	0.322	0.883	0.385	0.385	0.482	0.908
$\leqslant 500$	0.183	0.183	0.284	0.863	0.207	0.207	0.395	0.832
$\leqslant 1000$	0.173	0.176	0.429	0.939	0.203	0.203	0.489	0.925
$\leqslant 2000$	0.269	0.270	0.665	0.979	0.293	0.293	0.668	0.976
$\leqslant 4000$	0.164	0.167	0.679	0.994	0.186	0.186	0.681	0.993
$\leqslant 8000$	0.170	0.170	0.794	1.000	0.175	0.175	0.795	1.000
All	0.226	0.228	0.453	0.924	0.274	0.274	0.537	0.924
				T_{max}				
$\leqslant 250$	0.555	0.556	0.588	0.885	0.621	0.621	0.681	0.912
$\leqslant 500$	0.457	0.457	0.547	0.880	0.493	0.493	0.631	0.891
$\leqslant 1000$	0.480	0.482	0.681	0.940	0.514	0.514	0.744	0.941
$\leqslant 2000$	0.545	0.546	0.893	0.984	0.568	0.568	0.893	0.984
$\leqslant 4000$	0.454	0.458	0.900	0.995	0.476	0.476	0.899	0.995
$\leqslant 8000$	0.381	0.381	0.952	0.990	0.387	0.388	0.960	0.990
All	0.501	0.502	0.699	0.927	0.542	0.542	0.756	0.939

8.4. Vehicle ratio

In this and subsequent subsections, we will research how parameters such as *vehicleRatio*, *pickupRatio*, and *occupancyRatio* affect the quality of solutions. In this subsection in particular, our aim is to provide results that will help practitioners decide the number of vehicles for a given number of orders to ensure (1) sufficient high solution quality, (2) sufficient low chance of infeasible solutions.

To this end, we will use a derived instance dataset. It is an extension of the basic dataset by considering 21 different values of the *vehicleRatio* parameter for each of the 320 instances, essentially meaning that we consider 6720 instances. Since the relationship between different representations was established in the previous subsection, we will restrict ourselves only to B-representation. Moreover, in addition to HVI, ΣD, and T_{max}, we will also show the percentage of feasible solutions among 320 instances for each value of *vehicleRatio*. The results are shown in Table 10.

Table 10: Aggregated relative HVI, ΣD and T_{max} results of GA and greedy algorithms for B-representation and various values of *vehicleRatio*

Ratio	Feasible	B-GA			B-Greedy		
	%	HVI	ΣD	T_{max}	HVI	ΣD	T_{max}
0.0060	9.4	3.935	0.712	0.900	1.000	1.000	1.000
0.0065	9.4	3.934	0.711	0.900	1.000	1.000	1.000
0.0070	16.3	3.744	0.660	0.902	1.003	0.955	1.000
0.0075	20.9	3.560	0.657	0.906	1.002	0.938	1.000
0.0080	20.9	3.563	0.657	0.906	1.002	0.938	1.000
0.0085	27.2	3.554	0.670	0.906	1.001	0.949	0.999
0.0090	50.0	3.037	0.717	0.918	1.001	0.940	0.997
0.0095	74.7	2.606	0.780	0.926	1.008	0.958	0.988
0.0100	82.2	2.641	0.782	0.925	1.016	0.962	0.987
0.0105	92.5	2.797	0.779	0.918	1.021	0.966	0.986
0.0110	95.6	2.785	0.783	0.917	1.027	0.968	0.985
0.0115	96.3	2.802	0.784	0.917	1.028	0.969	0.985
0.0120	99.7	2.794	0.796	0.917	1.030	0.973	0.984
0.0125	100.0	2.792	0.800	0.916	1.030	0.973	0.984
0.0130	100.0	2.787	0.803	0.916	1.030	0.973	0.984
0.0135	100.0	2.779	0.804	0.917	1.028	0.973	0.985
0.0140	100.0	2.778	0.804	0.916	1.028	0.973	0.985
0.0145	100.0	2.778	0.805	0.916	1.028	0.973	0.985
0.0150	100.0	2.776	0.805	0.917	1.028	0.973	0.985
0.0155	100.0	2.773	0.805	0.917	1.028	0.973	0.985
0.0160	100.0	2.768	0.806	0.917	1.029	0.973	0.984

Regarding solution feasibility, we see that every instance for *vehicleRatio* $\geqslant$ 0.012 is solved to feasibility, but for smaller values the feasibility quickly drops to 10% for *vehicleRatio* = 0.006. We conclude that its enough to have one vehicle per 80 orders to assure the feasibility of solutions. Next, we move on to the analysis of the HVI results. For the greedy algorithm, the HVI increases as *vehicleRatio* increases, meaning more vehicles available lead to better solutions. However, this increase is very small, providing only 2–3% improvement. On the other hand, this effect is completely reversed for the GA method, which obtained the best HVI for a small number of vehicles. This might seem surprising, but is most likely the combination of GA being able to better utilize the vehicles and the fact that the fewer vehicles we use, the less times we need to add the depot-locker distances, which are relatively high compared to locker-locker distances. Moreover, it should be noted that even for *vehicleRatio* = 0.016 the GA method vastly outperforms the greedy method. One can also notice that the values obtained for HVI are different from those of Table 8. This is caused by HVI being a relative measure and is sensitive to which fronts it compares. In the previous section, each instance had 8 results (4 representations times 2 algorithms). In this section, each instance has 42 results (21 *vehicleRatio* values times 2 algorithms). As such, absolute HVI values cannot be compared between various instances, and relative HVI values cannot be compared between various experiments.

We will now move on to the analysis of the criteria ΣD and T_{max}, for which the results are shown in Table 10. Let us start with T_{max}. We see that both algorithms struggle with optimizing this criterion. Moreover, the greedy method obtains the best T_{max} values for the highest *vehicleRatio*, while the GA method has the best results for the smallest *vehicleRatio*. The explanation might be that with more vehicles, the greedy method simply has more solutions to check, causing its accuracy to drop. We also notice that for the greedy method T_{max} drops consistently with increasing *vehicleRatio*. However, for the GA method, the worst values are actually obtained around *vehicleRatio* = 0.0095. Most likely for a small number of vehicles the GA method can find a good solution due to a smaller size of the solution space, while with a high number of vehicles, it is easier to deliver orders faster by using more vehicles. The middle range has both too many vehicles to search the solution space efficiently and too few vehicles to spread the orders properly. The ΣD is also interesting, as both algorithms obtain the best values of it for *vehicleRatio* around 0.008. Thus, fewer vehicles are not always better for optimizing the total distance traveled.

It should also be noted that the GA method consistently obtains better ΣD and T_{max}, although the difference changes with *vehicleRatio*. For ΣD, this difference ranges from 6% to nearly 45%. The differences for T_{max} are smaller, ranging between 7% and 10%.

8.5. Delivery to pickup ratio

We have performed similar research for 11 different values of the *pickupRatio* parameter. We have chosen values from 0.0 (i.e. all orders are delivery orders) to 1.0 (i.e. all orders are pickup orders). This means 11 variants of each one of the basic 320 instances for 3520 instances in total. The aim of this research is to help practitioners understand how different delivery-pickup ratio affects the observed solution quality and feasibility. The aggregated results are shown in Table 11.

Table 11: Aggregated relative HVI, ΣD and T_{max} results of GA and greedy algorithms for B-representation and various values of *pickupRatio*

Ratio	Feasible	B-GA			B-Greedy		
	%	HVI	ΣD	T_{max}	HVI	ΣD	T_{max}
0.0	98.8	2.813	0.804	0.805	1.285	0.972	0.854
0.1	100.0	2.815	0.791	0.820	1.284	0.957	0.875
0.2	100.0	2.883	0.767	0.839	1.320	0.930	0.901
0.3	100.0	2.937	0.742	0.857	1.372	0.896	0.924
0.4	100.0	2.975	0.724	0.875	1.325	0.878	0.950
0.5	100.0	3.003	0.708	0.891	1.339	0.856	0.967
0.6	100.0	3.041	0.726	0.864	1.391	0.881	0.935
0.7	100.0	3.080	0.750	0.831	1.503	0.900	0.891
0.8	100.0	3.176	0.773	0.796	1.561	0.924	0.853
0.9	100.0	3.313	0.800	0.752	1.714	0.938	0.808
1.0	98.8	8.480	0.794	0.000	4.385	0.982	0.000

First, we notice that different delivery-to-pickup order ratio does not affect feasibility, except for the most extreme cases. The most likely reason is that when the orders are mixed, the pickup orders being moved from lockers into the vehicle can fit due to delivery orders being moved from the vehicle into the lockers. However, when all orders are deliveries (or pickups), then all orders have to fit into the vehicle, which is less likely. The infeasibility frequency is 1.2% at most. We conclude that *pickupRatio* has a negligible effect on solution feasibility.

Considering HVI, its value increases with *pickupRatio*, which means that better solutions are obtained where more orders are pickups instead of deliveries. Moreover, the relative HVI values for the GA method are consistently twice as high or higher than those for the greedy method. Considering T_{max}, we observe that for *pickupRatio* = 1 its value is 0, significantly increasing the HVI. This is expected, as with no delivery orders, T_{max} is always zero by definition. However, leaving that extreme case aside, we observe an interesting result. The worst values of T_{max} are when *pickupRation* is around 0.5. The explanation is that

with mixed orders the algorithms struggle between saving the pickup orders for later (completing delivery orders early to minimize T_{max}, but potentially causes a vehicle to visit some locations twice) and completing them early (thus leading to higher T_{max}, but saving distance to minimize ΣD). Interestingly, the results are the opposite for ΣD, where the best results are for $pickupRation = 0.5$. This is harder to explain, but could be caused by the fact mentioned earlier – that mixing "remove from vehicle" orders with "put into vehicle" orders causes better utilization of vehicle capacity. Thus, the solution requires fewer vehicles or creates shorter routes for some vehicles, decreasing ΣD.

In summary, the effect of the delivery-to-pickup ratio on solution feasiblity is negligible. However, different ratios significantly affect solution quality, indicating that practitioners should account for this when comparing profits or costs for different ratios.

8.6. Locker occupancy ratio

Finally, we have researched the influence of *occupancyRatio* on the quality of the solutions. The aim of this research is similar to the one in the previous subsection (to allow practitioners to see how different locker occupancy conditions) affect the quality and feasibility of solutions. This is especially important because, unlike the vehicle ratio, locker occupancy cannot be directly controlled by the company.

Similarly, to the previous section we wanted to try 11 values from 0.0 to 1.0. However, the value of 1.0 does not make sense (i.e., all lockers would be unavailable and no solution would be feasible). Moreover, a peculiar outcome was observed when trying to test the instances for $occupancyRatio = 0.9$. For some instances, this was infeasible, but for some other instances, the algorithms, which were set to a time limit of 720 seconds at most, were running for over an hour. This was likely caused by vehicles having to visit many alternative locations on average, which moved the running time of the DES from $O(m)$ closer to the worst-case scenario of $O(m^2 \log m)$. As such, we have stopped at $occupancyRatio = 0.8$. The aggregated results are shown in Table 12.

First, we observe that *occupancyRatio* starts to affect the feasibility of the solution only when around 70% of all lockers become unavailable. Moreover, even at $occupancyRatio = 0.8$ more than 5/6 of all instances are still solved to feasibility. Thus, we conclude that the proposed algorithms are fairly robust. Regarding the relative HVI, we observe that up to $occupancyRatio = 0.3$ the decrease in quality is minimal. As occupancy grows further, the HVI starts to drop faster. However, it is only past $occupancyRatio = 0.6$, where HVI drops more than 10% for both GA and greedy methods. Moreover, the relative HVI for the GA method is 60% to more than 260% higher than for the greedy method.

Table 12: Aggregated relative HVI, ΣD and T_{max} results of GA and greedy algorithms for B-representation and various values of *occupancyRatio*

Ratio	Feasible	B-GA			B-Greedy		
	%	HVI	ΣD	T_{max}	HVI	ΣD	T_{max}
0.0	100.0	4.901	0.697	0.821	3.022	0.835	0.882
0.1	100.0	4.900	0.697	0.821	3.022	0.835	0.882
0.2	100.0	4.900	0.697	0.821	3.020	0.835	0.882
0.3	100.0	4.893	0.697	0.821	3.016	0.836	0.882
0.4	100.0	4.867	0.699	0.822	2.985	0.837	0.883
0.5	100.0	4.793	0.702	0.823	2.923	0.840	0.884
0.6	100.0	4.539	0.717	0.831	2.681	0.853	0.894
0.7	98.4	3.944	0.752	0.849	2.086	0.891	0.919
0.8	84.1	2.678	0.841	0.903	1.013	0.979	0.988

Interestingly, the higher *occupancyRatio* is, the smaller the advantage of GA over the greedy method becomes.

Regarding ΣD and T_{max}, their values behave similar to HVI, becoming progressively worse as *occupancyRatio* exceeds 0.3 and 0.6. Despite that, for *occupancyRatio* = 0.8 the GA method still obtains better ΣD values than the greedy method for *occupancyRatio* = 0.0. Similarly, the average T_{max} for the GA method is almost always better than the best T_{max} obtained by the greedy method, except for *occupancyRatio* = 0.8. In general, the GA method obtains 7% to 20% better values of both criteria.

In summary, the PLBD system remains fairly robust even when locker occupancy approaches 60%, which means that companies do not have to worry about solution feasibility or a drop in quality due to unavailable lockers. This also leaves companies with enough of a leeway to install new lockers when the average occupancy starts to increase.

9. Conclusions and future work

In this paper we considered a Parcel Locker-Based Delivery system formulated as an optimization problem related to the well-known Vehicle Routing Problem. We considered several practical problem constraints including: (1) the possibility of delivering packages to alternative locker locations (as long as the location is as close to the originally intended location as possible), (2) both delivery and pickups, (3) locker capacities, (4) vehicle capacities, (5) order sizes and weights, and (6) influence of in-day time on travel times to account for road conditions. To account for both delivery costs and customer satisfaction, we considered a multi-

criteria goal function to minimize both: (1) total distance traveled by all vehicles and (2) delivery time of all deliveries.

We formulate the problem and then, due to its complexity (mainly caused by the possibility of multiple alternative delivery locations) and large problem sizes encountered in practice, propose an indirect solution representation. For this representation, we have presented a Discrete Event Simulation-based procedure to both compute the value of the goal function and assert the feasibility of the solution. Despite our approach being deterministic, the procedure is compatible with stochastic and dynamic approaches, as it uses an agent to make dynamic decisions regarding which alternative locations to use.

As multiple orders being intended for the same location is common, we have proposed three additional solution representations based on different levels of grouping orders together. We have presented several theoretical properties, showing that grouping orders might not be optimal but in practice leads to a very significant reduction of the solution space size. We have proposed two solving methods — a greedy heuristic and Genetic Algorithm metaheuristic — with variants for each of the proposed solution representations.

We have prepared an extensive instance dataset for the considered problem using real-life data from 8 Polish cities and one real-life delivery company. This data included parcel locker properties and locations, real-life location distances, and the approximate effect of the in-day time on road conditions. Our dataset contains instances of up to more than 7000 orders and 900 locations, which is larger and more practical than many existing studies.

During computer experiments, we have shown that the proposed order-grouping representations allows to obtain much higher quality of solutions as measured by the Hyper-volume Indicator and direct values of both optimization criteria. We have also established how parameters of the problem such as the number of orders, delivery-to-pickup ratio, vehicle ratio, and parcel locker occupancy probability affect the quality and feasibility of the solution.

We consider several directions and extensions for future research. First, we want to provide a better way to approximate the effects of road conditions on travel time. Second, we would like to turn the problem into stochastic Parcel Locker-Based Delivery to better model the challenges encountered by the delivery company in practice. Third, we would like to research more advanced properties to group the orders together more efficiently.

References

[1] K. Akdoğan and E. Özceylan: Parcel lockers location and routing problem: A bibliometric, descriptive, and content analysis of existing studies. *International Journal of Industrial Engineering*, **29**(5), (2022). DOI: 10.23055/ijietap.2022.29.5.8319

[2] M. Amorim-Lopes, L. Guimarães, J. Alves and B. Almada-Lobo: Improving picking performance at a large retailer warehouse by combining probabilistic simulation, optimization, and discrete-event simulation. *International Transactions in Operational Research*, **28**(2), (2021), 687–715. DOI: 10.1111/itor.12852

[3] P. Carotenuto, R. Ceccato, M. Gastaldi, S. Giordani, R. Rossi and A. Salvatore: Comparing home and parcel lockers' delivery systems: A math-heuristic approach. *Transportation Research Procedia*, **62** (2022), 91–98. 24th Euro Working Group on Transportation Meeting. DOI: 10.1016/j.trpro.2022.02.012

[4] W. Chen, Y. Liu and J. Liu: The multi-objective vehicle routing problems with parcel lockers for simultaneous pick-up and delivery. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, (2024), 1–8. DOI: 10.1109/CEC60901.2024.10611977

[5] A. Haghani and S. Jung: A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, **32**(11), (2005), 2959–2986. DOI: 10.1016/j.cor.2004.04.013

[6] C-L. Hwang and K. Yoon: *Multiple Attribute Decision Making: Methods and Applications A State-of-the-Art Survey*. Springer Berlin, Heidelberg, 1981. DOI: 10.1007/978-3-642-48318-9

[7] R. Idzikowski, J. Rudy and M. Jaroszczuk: Multi-criteria vehicle routing problem for a real-life parcel locker-based delivery. *International Journal of Electronics and Telecommunications*, **71**(3), (2025), 1–9. DOI: 10.24425/ijet.2025.153624

[8] S. Iwan, K. Kijewska and J. Lemke: Analysis of parcel lockers' efficiency as the last mile delivery solution – the results of the research in Poland. *Transportation Research Procedia*, **12** (2016), 644-655. Tenth International Conference on City Logistics 17-19 June 2015, Tenerife, Spain. DOI: 10.1016/j.trpro.2016.02.018

[9] B. Jing, L. Wu, H. Mao, S. Gong, J. He, C. Zou, G. Song, X. Li and Z. Wu: Development of a vehicle emission inventory with high temporal–spatial resolution based on nrt traffic data and its impact on air pollution in beijing–part 1: Development and evaluation of vehicle emission inventory. *Atmospheric Chemistry and Physics*, **16**(5), (2016), 3161–3170. DOI: 10.5194/acp-16-3161-2016

[10] G. Kim: Optimization for vehicle routing problem with locations of parcel lockers. *Journal of Korean Society of Industrial and Systems Engineering*, **45**(4), (2022), 134–141. DOI: 10.11627/jksie.2022.45.4.134

[11] G. Kim, Y.S. Ong, T. Cheong and P.S. Tan: Solving the dynamic vehicle routing problem under traffic congestion. *IEEE Transactions on Intelligent Transportation Systems*, **17**(8), (2016), 2367–2380. DOI: 10.1109/TITS.2016.2521779

[12] J. Lemke, S. Iwan and J. Korczak: Usability of the parcel lockers from the customer perspective – the research in polish cities. *Transportation Research Procedia*, **16** (2016), 272–287. The 2nd International Conference "Green Cities – Green Logistics for Greener Cities", 2-3 March 2016, Szczecin, Poland. DOI: 10.1016/j.trpro.2016.11.027

[13] Y. Liu, Q. Ye, J. Escribano-Macias, Y. Feng, E. Candela and P. Angeloudis: Route planning for last-mile deliveries using mobile parcel lockers: A hybrid q-learning network approach. *Transportation Research Part E: Logistics and Transportation Review*, **177** (2023), 103234. DOI: 10.1016/j.tre.2023.103234

[14] D. Luxen and Ch. Vetter: Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, (2011), 513–516, New York, NY, USA. DOI: 10.1145/2093973.2094062

[15] N. Moradi, F. Mafakheri and Ch. Wang: Covering routing problem with robots and parcel lockers: a sustainable last-mile delivery approach. In *Proceedings of the IISE Annual Conference*, (2024), 1-6. Institute of Industrial and Systems Engineers (IISE). DOI: 10.21872/2024IISE_7899

[16] Openstreetmap, 2021. [accessed on: 26.02.2025]. URL: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap

[17] I. Orenstein, T. Raviv and E. Sadan: Flexible parcel delivery to automated parcel lockers: models, solution methods and analysis. *EURO Journal on Transportation and Logistics*, **8**(5), (2019), 683–711. DOI: 10.1007/s13676-019-00144-7

[18] M. Peppel, S. Spinler and M. Winkenbach: Integrating mobile parcel lockers into last-mile delivery networks: An operational design for home delivery, stationary, and mobile parcel lockers. *International Journal of Physical Distribution & Logistics Management*, **54**(4), (2024), 418–447. DOI: 10.1108/IJPDLM-01-2023-0055

[19] M. Prandtstetter, C. Seragiotto, J. Braith, S. Eitler, B. Ennser, G. Hauger, N. Hohenecker, R. Schodl and M. Steinbauer: On the impact of open parcel lockers on traffic. *Sustainability*, **13**(2), (2021). DOI: 10.3390/su13020755

[20] A. Ranjbari, C. Diehl, G. Dalla Chiara and A. Goodchild: Do parcel lockers reduce delivery times? Evidence from the field. *Transportation Research Part E: Logistics and Transportation Review*, **172** (2023), 103070. DOI: 10.1016/j.tre.2023.103070

[21] J. Rudy: Github repository. https://github.com/jaroslaw-rudy/parcel-locker-based-delivery-optimization-2025, 2025. [accessed on 10.03.2025].

[22] A. Saker, A. Eltawil and I. Ali: Adaptive large neighborhood search metaheuristic for the capacitated vehicle routing problem with parcel lockers. *Logistics*, **7**(4), (2023). DOI: 10.3390/logistics7040072

[23] R.G. Thompson, S. Pan, L. Zhang and H. Ghaderi: A parcel network flow approach for joint delivery networks using parcel lockers. *International Journal of Production Research*, **59**(7), (2021), 2090–2115. DOI: 10.1080/00207543.2020.1856440

[24] P. Sitek, J. Wikarek, K. Rutczyńska-Wdowiak, G. Bocewicz and Z. Banaszak: Optimization of capacitated vehicle routing problem with alternative delivery, pick-up and time windows: A modified hybrid approach. *Neurocomputing*, **423** (2021), 670–678. DOI: 10.1016/j.neucom.2020.02.126

[25] P. Sitek and J. Wikarek: Capacitated vehicle routing problem with pick-up and alternative delivery (CVRPPAD): model and implementation using hybrid approach. *Annals of Operations Research*, **273**(1), (2019), 257–277. DOI: 10.1007/s10479-017-2722-x

[26] Statistics Poland. Area and population in the territorial profile in 2021. (2021). [accessed on: 26.02.2025]. URL: https://stat.gov.pl/en/topics/population/population/area-and-population-in-the-territorial-profile-in-2021,4,15.html

[27] Y. Wang, M. Bi, J. Lai, Ch. Wang, Y. Chen and J. Holguín-Veras: Recourse strategy for the routing problem of mobile parcel lockers with time windows under uncertain

demands. *European Journal of Operational Research*, **316**(3), (2024), 942–957. DOI: 10.1016/j.ejor.2024.02.034

[28] S. Yan, Ch-S. Sun and Y-Y. Chen: Optimal routing and scheduling for a mobile parcel locker delivery system. *Research in Transportation Business & Management*, **59** (2025), 101318. DOI: 10.1016/j.rtbm.2025.101318

[29] V.F. Yu, P.T. Anh and Y-W. Chen: The electric vehicle routing problem with time windows, partial recharges, and parcel lockers. *Applied Sciences*, **13**(16), (2023). DOI: 10.3390/app13169190

[30] V.F. Yu, H. Susanto, P. Jodiawan, T-W. Ho, S-W. Lin and Y-T. Huang: A simulated annealing algorithm for the vehicle routing problem with parcel lockers. *IEEE Access*, **10** (2022), 20764–20782. DOI: 10.1109/ACCESS.2022.3152062

[31] E. Zitzler and L. Thiele: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, **3**(4), (1999), 257–271. DOI: 10.1109/4235.797969