Formulating and solving integrated order batching and routing in multi-depot AGV-assisted mixed-shelves warehouses

Lin Xie[a,*], Hanyi Li[b], Laurin Luttmann[a]

[a] *Leuphana University of Lüneburg, Universitätallee 1, 21335 Lüneburg*
[b] *Beijing Hanning Tech Co., Ltd*

Abstract

Different retail and e-commerce companies are facing the challenge of assembling large numbers of time-critical picking orders that include both single-line and multi-line orders. To reduce unproductive picker working time as in traditional picker-to-parts warehousing systems, different solutions are proposed in the literature and in practice. For example, in a mixed-shelves storage policy, items of the same stock keeping unit are spread over several shelves in a warehouse; or automated guided vehicles (AGVs) are used to transport the picked items from the storage area to packing stations instead of human pickers. This is the first paper to combine both solutions, creating what we call *AGV-assisted mixed-shelves picking systems*. We model the new integrated order batching and routing problem in such systems as an extended multi-depot vehicle routing problem with both three-index and two-commodity network flow formulations. Due to the complexity of the integrated problem, we develop a novel variable neighborhood search algorithm to solve the integrated problem more efficiently. We test our methods with different sizes of instances, and conclude that the mixed-shelves storage policy is more suitable than the usual storage policy in AGV-assisted mixed-shelves systems for both single-line and multi-line orders (saving up to 67% on driving distances for AGVs). Our variable neighborhood search algorithm provides close-to-optimal solutions within an acceptable computational time.

Keywords: Logistics, Order batching, Routing, Mixed-shelves storage, AGV-assisted picking

*Lin Xie
Email address: `xie@leuphana.de` (Lin Xie)
URL: `http://www.leuphana.de/or/` (Lin Xie)

1. Introduction

The most important and time-consuming task in a warehouse is the collection of items from their storage locations to fufill customer orders. This process is called *order picking*, which may constitute about 50–65% of operating costs. Therefore order picking is considered the highest-priority area for productivity improvements (see De Koster et al. (2007)). In a traditional manual order picking system (also called a *picker-to-parts system*), the pickers spend 50% of their working time on the task of walking (see Tompkins (2010); for an overview of manual order picking systems see De Koster et al. (2007)). The unproductive working times require the picker-to-parts system to have a large workforce, especially for companies which have millions of small-sized items in large warehouses, such as the retailers Amazon, Alibaba, Zara, Zalando and Walmart. Many of them provide both brick-and-mortar stores and online shops to create a seamless shopping experience for customers (omnichannel flexibility). Due to the diversity of online shops, we concentrate on both single-line and multi-line small-sized orders. Especially during the COVID-19 pandemic, online grocery sales are growing threefold faster (see Fabric (2020)). There are increasing demands for alternative warehousing systems to increase the efficiency of order picking, for example, robot-based compact storage and retrieval systems and robotic mobile fulfillment systems (see more details in Azadeh et al. (2019)). Here we consider a relatively new warehousing concept that does not use expensive fixed hardware and can be easily and quickly implemented, called *AGV-assisted picking* (see Boysen et al. (2019), Azadeh et al. (2019)).

1.1. AGV-assisted pick systems

As described in Azadeh et al. (2019), there are different variants of this type of system. In this paper, we concentrate on the one that has been marketed to the warehousing industry, such as Locus Robotics, 6 River Systems and Fetch Robotics (see Figure 1).

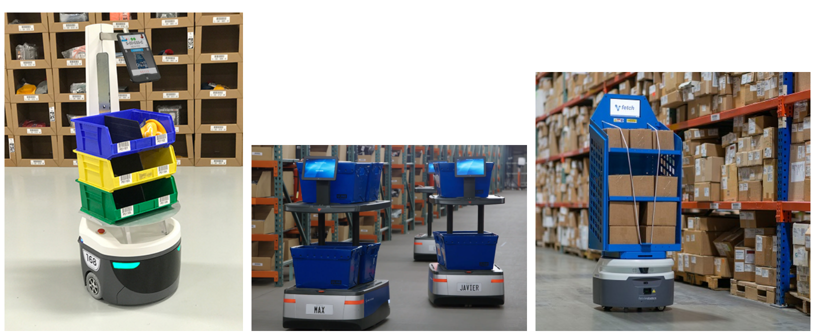

Figure 1: Companies who own an AGV-assisted picking system (from left to right: Locus Robotics, 6 River Systems, Fetch Robotics).

In such systems, each human *picker* works at his/her working area in the *storage area* (containing shelves with items). For example, in Figure 2, picker

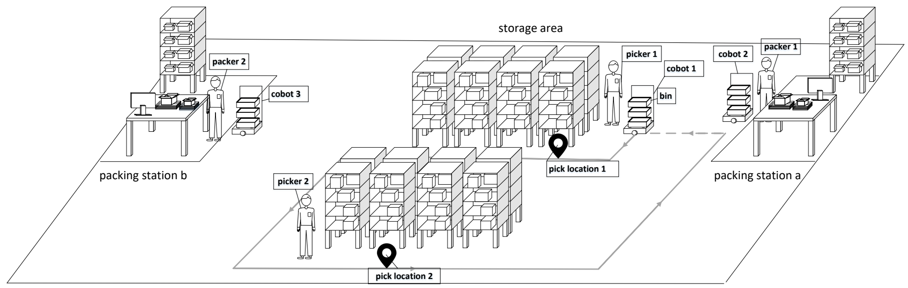

Figure 2: An example of a AGV-assisted picking system (with two packing stations).

1 works in the upper area of the storage area while picker 2 works in the lower area. Each mobile robot (called a *cobot*) leaves from its packing station (also called a *depot*) and goes automatically to a pick location in the storage area and waits for a human picker to arrive. The picker identifies and picks up the ordered items from shelves and puts them into bins carried by the cobot (one *bin* is used to temporarily store an order); after that, the cobot goes to its next picking location. The load of each cobot is limited. After collecting, each cobot goes back to its packing station to drop off the loaded bins, and the human *packer* at that station packs the collected items. Each cobot has a dedicated packing station and it is charged there. In the example in Figure 2, cobot 1 leaves packing station a and goes firstly to pick location 1 to wait for picker 1 to load items; after that, it goes to pick locatin 2 to wait for picker 2 to load items and returns to packing station a. If there is another cobot in front of station a, then it should wait in a *queue* in front of the station. The explanation of all components in such a picking system can be found in Table A.8. In different real-world warehouses, these depots might vary slightly as shown in Figure 2. For example, the depots might be drop-off locations, where the bins are put onto a conveyor system to transport them to packing stations. But each cobot still has a dedicated depot, which is its charging location. In the rest of the paper, the terms *depot* and *packing station* are synonyms. This system is called a *follow-pick* system in Lee & Murray (2019), and Boysen et al. (2019) call this model of operation the *free-floating* policy. Although AGV-based picking systems are often applied for heavy and bulky items, there are also applications for small-sized items (see Boysen et al. (2019)). The latter applications are also a focus of this paper.

In practice, AGV-assisted pick systems are becoming increasingly popular, since adding AGVs to an existing picker-to-parts system barely alters the basic fulfillment processes (easy and quick implementation). More than 80% of all warehouses in Western Europe still follow the traditional picker-to-parts setup (see De Koster et al. (2012)). The other automated systems require, for example, expensive fixed hardware, such as automated storage systems in robot-based compact storage and retrieval systems. An AGV-assisted system is also easily scalable by adding or reducing pickers and AGVs to adapt to varying workloads, for example due to seasonal changes in customers' orders, such as end-of-season sales.

As described in Azadeh et al. (2019) and Boysen et al. (2019), AGV-assisted picking is a relatively new concept. The only publication about the free-floating system is in Lee & Murray (2019). They modeled the routing problem to support the cooperation of picker robots with transport robots and compared this with the cooperation of human pickers with transport robots (follow-pick). They concluded that human pickers are more suitable for identifying and picking up items. They also analyzed the relationship between layout configuration and robot system parameters. In contrast to that paper, we concentrate on integrated batching and routing in a mixed-shelves follow-pick system.

1.2. Mixed-shelves picking systems

As described in Boysen et al. (2019) and Weidinger et al. (2019), mixed-shelves storage has been applied by many large-sized facilities in e-commerce companies, for example in the distribution centers of Amazon Europe and fashion retailer Zalando. In such storage, items of the same stock keeping unit (SKU) are spread over the shelves, so there are always some items of that SKU close by for picking. The main advantage of such storage is that it reduces unproductive working time. That is supported by Weidinger et al. (2019) for traditional manual systems and Boysen et al. (2017) for robotic mobile fulfillment systems. However, the literature shows that this storage is more suitable for online companies, which usually have many single-line orders (or small order-line demands). For cases with a large demand per SKU or multiple order lines per order, several storage locations might need to be visited by pickers to finish one order.

We apply mixed-shelves storage in an AGV-assisted pick system (called a *multi-depot AGV-assisted mixed-shelves system*) for picking both single-line and multi-line small-sized orders and compare this with the reference storage policy, called *dedicated storage*. In the dedicated storage policy, items of the same SKU are stored in only one shelf.

1.3. Operational decision problems and literature review

The problems that occur in a multi-depot AGV-assisted mixed-shelves system are mostly similar to the problems in traditional manual order picking systems, including strategic (such as layout planning), tactical (such as zoning and storage assignment) and operational problems (batching, routing and job assignment) (see overview in Van Gils et al. (2018) for the traditional manual order picking systems). In addition, we have several depots in our system, and each cobot has its dedicated depot, i.e. it leaves from its depot and goes back to that depot. Therefore, we also need to decide in which depot an order will be handled to minimize driving distances while ensuring balanced workloads among all depots (this will be discussed further in Section 2).

We consider in this paper two integrated operational problems in a multi-depot AGV-assisted mixed-shelves system: customer orders are combined in several pick rounds (batches) in *order batching* by deciding from which depot a batch is leaving, while the sequence of storage locations that cobots should visit to load all retrieved items from human pickers in each batch is determined in *routing*.

In the following, the literature of picker routing in manual mixed-shelves warehouse and integrated order batching and routing will be described.

Picker routing in mixed-shelves picking

As shown in Weidinger et al. (2019), picker routing in a mixed-shelves warehouse is much more complex than in traditional environments: Multiple orders are concurrently picked by each picker (in our case: cobot), many alternative depots (in our case: packing stations) are available, and items of the same SKU are available in multiple shelves. Daniels et al. (1998) was the first publication to integrate the selection of alternative storage positions into picker routing. The same problem was also addressed in Weidinger (2018) for rectangular warehouses, and integrating alternative pick locations was proven to make this problem strongly NP-hard.

The mixed-shelves storage in Weidinger et al. (2019) is similar to the environment we consider, including:

- Items of the same SKU are available in multiple shelves.
- The cobot / picking cart is able to carry multiple bins for different orders concurrently.

However, we do not have the same meaning of *depots* as in Weidinger et al. (2019). They define depots as access points to the conveyor system, not the leaving and return locations of cobots. So in our case we have dedicated assignment of cobots to depots. Each cobot begins from its depot and ends at the same depot. This property makes our formulation similar to the vehicle routing problem with multiple depots. An overview of multi-depot vehicle routing problems can be found in Montoya-Torres et al. (2015). It is known that multi-depot vehicle routing problems are proven to be NP-hard (Garey & Johnson (1979)). Furthermore, we should ensure the fair distribution of workload for the packers in different depots.

And we consider multiple pickers (in our case: cobots), rather than one picker as in Weidinger et al. (2019). The following problems are mentioned in Weidinger et al. (2019) for considering multiple pickers.

- They block each other in front of shelves (in this paper we assume that the aisles are wide enough for at least two (even three) cobots operating side by side (see Lee & Murray (2019) for the same assumption))
- They block each other in front of depots (there is a queue in front of each depot; furthermore, due to the short drop-off time by removing bins from cobots, we assume that the blocking is minimal).
- They influence each other's inventory levels (this will be formulated in our model, and that might increase the complexity of the model).

Integrated batching and routing

Due to the similar time horizon of the batching and routing decisions and the strong relationship between them, the integration of them is extensively studied in the literature; see an overview in Van Gils et al. (2019). The efficiency of the integration compared with the sequential approach to both of these problems is already shown in previous papers, such as in Van Gils et al. (2019). Therefore, we do not focus on such comparison in this paper.

The differences in our problem from the existing integrated batching and routing problems for the traditional manual picking systems are summarized as follows.

- Not all locations (shelves) should be visited by each picker (cobot). It is similar to the Steiner TSP (see Letchford et al. (2013)), but the visited locations for each picker are not determined. They are decided by the routing itself. The items of an SKU are available in several locations (shelves) (mixed storage, see third column of Table 1).
- Multiple depots (packing stations) (see fourth column of Table 1).
- Multiple pickers (cobots) (see fifth column of Table 1).

In the first integrated model from Won & Olafsson (2005), they formulate it as the integrated bin packing and traveling salesman problem. The three-index formulations with sub-tour elimination constraints are widely used in the literature (see Table 1). It is worth mentioning that Chen et al. (2015), Scholz et al. (2017) and Van Gils et al. (2019) also use three-index formulation to integrate batching and routing with another operational problem.

	Formu-lation	Mixed storage	Multiple depots	Multiple pickers	Min. dist.	Heuristic
Won & Olafsson (2005)	1					FCFS+2-Opt
Tsai et al. (2008)	1				X	GA
Ene & Öztürk (2012)	1					GA
Kulak et al. (2012)	1				X	Cluster-based TS
Matusiak et al. (2014)	1				X	SA+A*
Cheng et al. (2015)	1				X	PSO+ACO
Lin et al. (2016)	1				X	PSO
Li et al. (2017)	1				X	Similarity+ACO
Valle et al. (2017)	1				X	-
Briant et al. (2020)						CG
this paper	1, 2	X	X	X	X	VNS

Table 1: The literature about integrated batching and routing. 1: three-index formulation; 2: two-commodity network flow formulation; FCFS: first come, first serve; GA: genetic algorithm; TS: tabu search; SA: simulated annealing; PSO: particle swarm optimization; ACO: ant colony optimization; CG: column generation; VNS: nariable neighborhood search.

The two-commodity network flow formulation for solving the traveling salesman problem was first introduced by Finke (1984). This formulation was proposed by Baldacci et al. (2004) to solve the capacitated vehicle routing problem and was extended by Ramos et al. (2020) to solve the multi-depot vehicle routing problem, and it was shown that the flow formulation provides better performance to achieve lower gaps within a smaller computational time. In our work, we extend the formulation in Ramos et al. (2020) for integrating vehicle routing with order batching in a mixed-shelves warehouse. In other words, we allow each location to be visited more than once and the demand in each location is unknown (but the maximum is given) and enable batching of orders. More about this formulation can be found in Subsection 3.2.

Due to the complexity of the integration, the problem is solved in the literature mostly using metaheuristics (see seventh column of Table 1). Li et al. (2017) mentioned several problems of existing applied metaheuristics for the integrated batching and routing problem, such as the bad performance caused by not using the full capacity of the vehicle in the batching process, or the long computational time. Instead, we propose in this work a new variable neighborhood search algorithm to solve our integrated problem. The main reason for that is that variable neighborhood search has been successfully applied to solve vehicle routing problems, such as large scale capacitated vehicle routing in Kytöjoki et al. (2007).

1.4. Contributions and paper structure

Based on the literature review in the previous subsection, we summarize the main contributions of this paper as follows:

- We formulate the first integrated batching and routing problem in a multi-depot AGV-assisted mixed-shelves warehouse with a new two-commodity network flow model and compare it with the classical three-index model.
- Due to the high complexity of the integration, a novel variable neighborhood search is developed to provide good solutions within shorter computational times.

The remainder of this paper is organized as follows. Section 2 describes the assumptions we used, while Sections 3 and 4 describe the different mathematical formulations and implemented metaheuristic for our integrated order batching and robot routing respectively. Computational results are described in Section 5. Finally, Section 6 presents conclusions and opportunities for future research.

2. Assumption for the mathematical models

The following assumptions are made for the formulation of the integrated problem in Section 3.

Objective. The objective of minimizing distances is widely used in the literature for the integrated batching and routing in the traditional manual systems (see sixth column of Table 1), and it assumes that minimizing distances has the same meaning as minimizing total picking time (Li et al. (2017), Tsai et al. (2008) and Won & Olafsson (2005)). This is supported by the assertion in De Koster et al. (2007) that the travel time is an increasing function of the travel distance, so that minimizing the latter is considered a primary objective in warehouse design and optimization. So we also use minimizing distances as our objective.

A given set of orders. The single-line and multi-line orders of customers are given as input for the models. No new incoming orders are considered.

Demand nodes. In order to formulate the problem as a vehicle routing problem, we need to define a set of demand nodes. So it is possible to define each item as a node. But there are a large number of nodes, and all items on one shelf share the same distances to items on another shelf. Therefore we use a shelf as an aggregated node for all items on it.

Picking list of an order. Each customer's order includes several SKUs with their given ordered quantities. Based on the property of mixed-shelves storage, items of the same SKU might be picked by different shelves. To simplify the modeling, we divide customer orders into a picking list. For example, we have order 1, including two units of SKU a, one unit of SKU b and three units of SKU c: $o_1 = (a, 2), (b, 1), (c, 3)$ (SKU, ordered quantity); then we have a picking list for o_1: $a_1, a_2, b_1, c_1, c_2, c_3$ in our models. In this way, two items of SKU a can be picked at different shelves.

No charging. We assume that all cobots we send out have enough remaining percentage of battery life for one tour. When a cobot visits its depot, it will be re-charged at a rate per unit of time.

Aisle congestion. We assume that the aisles are large enough for two or even three cobots moving side by side (the same assumption is common in the literature, such as in Lee & Murray (2019)), so the aisle congestion is minimal, since one cobot can go around the boundaries (such as: picker, another cobot).

Load capacity of batches. In integrated batching and routing, we can determine a tour for a cobot for each batch. So the load capacity of each batch is equal to the load capacity of a cobot. We assume that the payloads of all cobots are the same.

Same number of cobots in each depot. We assume that the same number of cobots are available in each depot.

A bin for each order. Each cobot can carry several bins, and each bin is for one order. This reduces the sorting work at the depots. Furthermore, a bin can be separated into several small sub-bins for small-sized items (see for example Locus Robotics in Figure 1). Therefore, we assume that the batching is limited by the load capacity of a cobot, not the number of orders/bins such as in Valle et al. (2017).

Pickers in the storage area. We don't analyze the number of pickers and size of their corresponding working areas (zones). To simplify the model, we assume that there is at least one picker near each picking location.

Workloads of different depots. In order to make the workload even for packers in different depots, we need to balance the number of batches in different depots. Assuming the processing time for an order in each depot is the same, minimized traveled distances and the balanced workload of each depot can lead to a good makespan for a given set of orders.

No refill in parallel to picking. We assume that our optimization begins after refill operations and the inventory has enough items for picking.

3. Mathematical model for integrated order batching and cobot routing

Our integrated order batching and cobot routing can be reduced to the capacitated vehicle routing problem if there is only one depot, with each shelf including only items of one SKU, single-line orders (each with a distinct SKU) and given number of batches. We assume each shelf corresponds to a customer and each batch corresponds to a vehicle, so each customer is assigned to exactly one vehicle. As the capacitated vehicle routing problem is known to be NP-hard (see, for example, Baldacci et al. (2007)), so is our problem.

In this section, we firstly model the integrated order batching and routing problem as a three-index formulation in Subsection 3.1, and then as a two-commodity network flow formulation in Subsection 3.2.

3.1. Three-index formulation

The following model is based on the integrated batching and routing in Cheng et al. (2015). In contrast to that, we consider mixed-shelves storage and multiple depots. Furthermore, we use one more index for cobots for the following reasons. First, we have multiple depots, so we need to know at which depot an order is packed. Therefore we assume that we have exactly one artificial cobot leaving from each depot and run all batches for that cobot beginning at that depot. Second, this model can be easily extended to support further integration with job assignment to determine the sequence of batches (see Van Gils et al. (2019)).

The sets, parameters and variables are as follows:

Sets:

$\mathcal{V}$	Set of nodes, including shelves $\mathcal{V}^{\mathrm{S}}$ (including at least one ordered item) and depots $\mathcal{V}^{\mathrm{D}}$ ($\mathcal{V} = \mathcal{V}^{\mathrm{S}} \cup \mathcal{V}^{\mathrm{D}}$)
$\mathcal{E}$	Set of edges $(i,j) \in \mathcal{E} \;\; \forall i,j \in \mathcal{V}$
$\mathcal{P}^{\mathrm{C}}$	Set of physical items for picking
$\mathcal{V}_p^{\mathrm{SC}}$	Set of shelves including item $p \in \mathcal{P}^{\mathrm{C}}$
$\mathcal{O}$	Set of orders
$\mathcal{P}_o^{\mathrm{C}}$	Set of physical items in the picking list for order o
$\mathcal{R}^{\mathrm{C}}$	Set of cobots for collecting items and transporting to depots
$\mathcal{B}_r$	Set of batches for cobot $r \in \mathcal{R}^{\mathrm{C}}$
$\mathcal{R}_i^{\mathrm{C}} \subset \mathcal{R}^{\mathrm{C}}$	Set of cobots leaving depot $i \in \mathcal{V}^{\mathrm{D}}$

Parameters:

d_{ij}	Distance between two nodes $i, j \in \mathcal{V}$
c	Maximum payload of a cobot
w_p	Weight of item $p \in \mathcal{P}^{\mathrm{C}}$
n_{ps}	Number of items sharing the same SKU as the physical item p at shelf node $s \in \mathcal{V}^{\mathrm{S}}$

Decision variables:

$$x_{ijrb} \quad \begin{cases} 1, & \text{Cobot } r \in \mathcal{R}^{\mathrm{C}} \text{ has visited node } j \in \mathcal{V} \text{ after node } i \in \mathcal{V} \text{ in batch } b \in \mathcal{B}_r \\ 0, & \text{else} \end{cases}$$

$$z_{psrb} \quad \begin{cases} 1, & \text{Cobot } r \in \mathcal{R}^{\mathrm{C}} \text{ has collected item } p \text{ at node } s \in \mathcal{V}_p^{\mathrm{SC}} \text{ in batch } b \in \mathcal{B}_r \\ 0, & \text{else} \end{cases}$$

$$y_{rb} \quad \begin{cases} 1, & \text{Cobot } r \in \mathcal{R}^{\mathrm{C}} \text{ is used in batch } b \in \mathcal{B}_r \\ 0, & \text{else} \end{cases}$$

$$y_{r} \quad \begin{cases} 1, & \text{Cobot } r \in \mathcal{R}^{\mathrm{C}} \text{ is used} \\ 0, & \text{else} \end{cases}$$

$$\omega_{orb} \quad \begin{cases} 1, & \text{Order } o \in \mathcal{O} \text{ is picked by cobot } r \in \mathcal{R}^{\mathrm{C}} \text{ in batch } b \in \mathcal{B}_r \\ 0, & \text{else} \end{cases}$$

Goal: Minimizing traveled distances of all batches

$$F(3.1) \quad z(F(3.1)) = \text{Min} \sum_{r \in \mathcal{R}^{\mathrm{C}}} \sum_{b \in \mathcal{B}} \sum_{(i,j) \in \mathcal{E}} d_{ij} * x_{ijrb} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{V}} x_{ijrb} = \sum_{j \in \mathcal{V}} x_{jirb} \leq y_{rb}, \ \forall r \in \mathcal{R}^{\mathrm{C}}, i \in \mathcal{V}, b \in \mathcal{B}_r \tag{2}$$

$$\sum_{j \in \mathcal{V}} x_{ijrb} = \sum_{j \in \mathcal{V}} x_{jirb} = y_{rb}, \ \forall r \in \mathcal{R}_i^{\mathrm{C}}, i \in \mathcal{V}^{\mathrm{D}}, b \in \mathcal{B}_r \tag{3}$$

$$\sum_{p \in \mathcal{P}^{\mathrm{C}}} \sum_{s \in \mathcal{V}_p^{\mathrm{SC}}} z_{psrb} * w_p \leq c, \ \forall r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{4}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijrb} \leq |S| - 1, \ \forall S \subset \mathcal{V}^{\mathrm{S}}, |S| \geq 0, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{5}$$

$$\sum_{s \in \mathcal{V}_p^{\mathrm{SC}}} \sum_{r \in \mathcal{R}^{\mathrm{C}}} \sum_{b \in \mathcal{B}_r} z_{psrb} = 1, \ \forall p \in \mathcal{P}^{\mathrm{C}} \tag{6}$$

$$\sum_{p \in \mathcal{P}^{\mathrm{C}}} \sum_{s \in \mathcal{V}_p^{\mathrm{SC}}} z_{psrb} = \omega_{orb} * |\mathcal{P}_o^{\mathrm{C}}|, \ \forall o \in \mathcal{O}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{7}$$

$$z_{psrb} \leq \sum_{j \in \mathcal{V}} x_{sjrb}, \ p \in \mathcal{P}^{\mathrm{C}}, s \in \mathcal{V}_p^{\mathrm{SC}}, \forall r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{8}$$

$$\sum_{r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r} z_{psrb} \leq n_{ps}, \ \forall p \in \mathcal{P}^{\mathrm{C}}, s \in \mathcal{V}_p^{\mathrm{SC}} \tag{9}$$

$$\sum_{p \in \mathcal{P}^{\mathrm{C}}} \sum_{s \in \mathcal{V}_p^{\mathrm{SC}}} z_{psrb} \leq M * y_{rb}, \ \forall r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{10}$$

$$y_{rb+1} \leq y_{rb}, \ \forall b \in \{1, 2, ..., |\mathcal{B}_r| - 1\}, r \in \mathcal{R}^{\mathrm{C}} \tag{11}$$

$$\sum_{b\in\mathcal{B}_r} y_{rb} \leq |\mathcal{B}_r| * y_r,\ \forall r \in \mathcal{R}^{\mathrm{C}} \tag{12}$$

$$y_{rb} \in \{0,1\},\ \forall r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{13}$$

$$y_r \in \{0,1\},\ \forall r \in \mathcal{R}^{\mathrm{C}} \tag{14}$$

$$x_{ijrb} \in \{0,1\},\ \forall (i,j) \in \mathcal{E}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{15}$$

$$z_{psrb} \in \{0,1\},\ \forall p \in \mathcal{P}, r \in \mathcal{R}^{\mathrm{C}}, s \in \mathcal{V}_p^{\mathrm{SC}}, b \in \mathcal{B}_r \tag{16}$$

$$\omega_{orb} \in \{0,1\},\ \forall r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{17}$$

A. Same constraints as in classical multi-depot vehicle routing problems

Constraint set (2) indicates the flow conservation, while constraint set (3) ensures that each batch leaves its depot once at the most, and if it leaves its depot, it should come back to that depot at the end of the tour. Furthermore, the capacity of batches is limited in constraint set (4). Constraint set (5) eliminates subtours. Similar constraints can be found in, for example, Ramos et al. (2020) for multi-depot vehicle routing problems.

B. Constraints about order batching

Each item must be assigned to a batch. Each item in the picking list should be collected exactly once in one batch in constraint set (6).

No splitting. Constraint set (7) ensures that all items for an order are collected in precisely one batch assigned to a cobot.

Set of constraints due to mixed-shelves storage. Constraint set (8) ensures that if one item p in shelf s is collected in batch b, shelf s should be visited in that batch. Constraints (9) ensure that the available stock of items in a shelf is not exceeded in all batches.

C. Constraints related to determine minimum number of batches for each cobot

Alternative I. A batch assigned to a cobot is only used if it has collected items in constraint set (10). Constraints (11) ensure that if a batch $b+1$ assigned to cobot r is used, batch b should be used as well. So the selection of batch is from 1 to $|\mathcal{B}_r|$. In this case, $|\mathcal{B}_r|$ is set to a constant, which is large enough. A cobot is only used if at least one of its batches is used (constraint set (12)). In this model, we set y_r to be 1, since we have one artificial cobot at each depot to ensure that there is at least one batch leaving that depot. The definitions of variables y_{rb} and y_r are given in (13) and (14). Using this alternative, we need to first run the model using objective (18). After that the optimal values of y_{rb} and y_r are determined and we run the model using objective (1) to find the optimal traveled distances.

$$\text{Min} \quad \sum_{r\in\mathcal{R}^{\mathrm{C}}} \sum_{b\in\mathcal{B}_r} y_{rb} + y_r \tag{18}$$

Alternative II. As an alternative to using this set of constraints in Alternative I (constraint sets from (10) to (14)), we can calculate $|\mathcal{B}_r^*|$ optimally (according to the balanced workload for each depot) as follows:

$$|\mathcal{B}_r^*| = \left\lceil \frac{\sum_{p \in \mathcal{P}^{\mathrm{C}}} w_p}{|\mathcal{R}^{\mathrm{C}}| * c} \right\rceil \tag{19}$$

D. Definition of variables

The binary definitions of variables are given in constraint sets (15)–(17).

3.2. Two-commodity network flow formulation

The model about the two-commodity network flow formulation is based on the formulation presented by Ramos et al. (2020) for solving the multi-depot vehicle routing problem. Constraint sets (2)–(3) and the constraint sets about order batching ((6)–(17)) are extended with a set of copy depots and new flow variables. We will show you an example to illustrate the idea of the flow formulation. First we need some addition definitions of sets and variables.

Additional set:

$\mathcal{V}^{\mathrm{D'}}$	Set of copy of depots ($\mathcal{V} = \mathcal{V}^{\mathrm{S}} \cup \mathcal{V}^{\mathrm{D}} \cup \mathcal{V}^{\mathrm{D'}}$)
$\mathcal{P}_s$	Set of all items in shelf $s \in \mathcal{V}^{\mathrm{S}}$

Additional decision variables:

y_{ijrb}	A flow variable representing the cobot load when cobot $r \in \mathcal{R}^{\mathrm{C}}$ travels from i to j in the b-th batch. The flow y_{jirb} represents the current empty space of cobot $r \in \mathcal{R}^{\mathrm{C}}$ in the b-th batch.

Example. Figure 3 shows an example of batch I limited by the batch capacity $c_I = 1$ (in other words: cobot 1 in depot a has the capacity of 1). Three orders are assigned to depot a. They include different items, whose sizes are equal (0.3). The items sharing the same SKU are illustrated with the same color and shape. For example, the items indicated by a green rectangle are included in orders 2 and 5, and they are stored in shelves 1 and 3. In the flow formulation, the route for batch I is defined by two paths: one from the real depot a to the copy depot a' ($a \to 1 \to 3 \to a'$), defined by y_{ijrb} (representing the vehicle load); while the reserve path from the copy depot to the real depot ($a' \to 3 \to 1 \to a$) is defined by y_{jirb} (representing the empty space in the vehicle). For example, the flow y_{a11I} indicates that cobot 1 leaves the depot in batch I with a load equal to the total weights of the three picking items, 0.9; while the flow $y_{a'31I} = 1$ means that the batch arrives empty at the depot after dropping off items. For each edge (i, j) of the route within a batch b of a cobot r, we have $y_{ijrb} + y_{jirb} = c$, for example, $y_{a11I} + y_{1a1I} = 0.9 + 0.1 = 1$.

Goal: Minimizing traveled distances for all batches

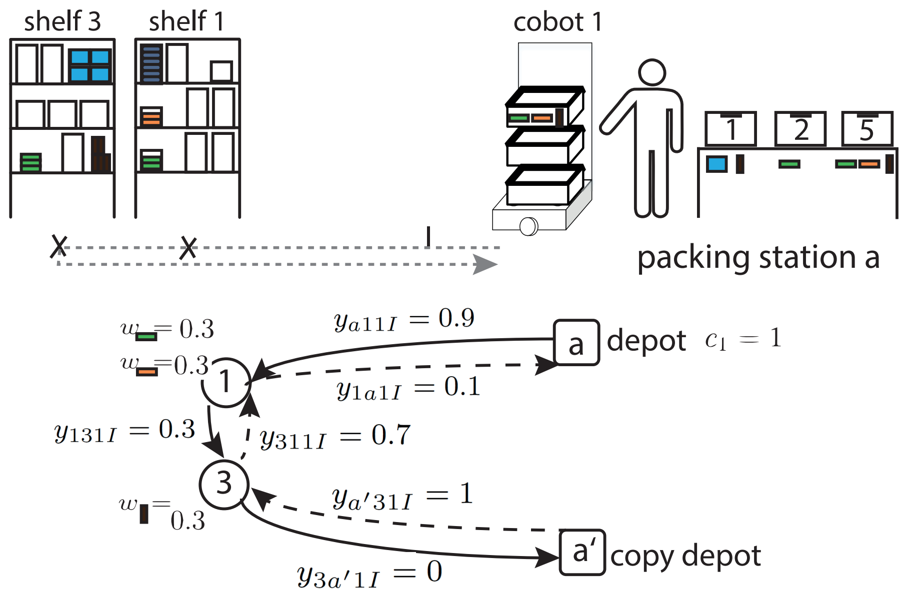

Figure 3: Flow paths for batch I for picking three items.

$$(F(3.2)) \quad z(F(3.2)) = \text{Min} \quad \sum_{r\in\mathcal{R}^{\mathrm{C}}} \sum_{b\in\mathcal{B}} \sum_{(i,j)\in\mathcal{E}} d_{ij} * x_{ijrb} \tag{1}$$

$$\text{s.t.} \quad (2)-(3),(6)-(17)$$

$$\sum_{j\in\mathcal{V}} (y_{jsrb} - y_{sjrb}) = 2 * \sum_{p\in\mathcal{P}_s^{\mathrm{C}}} (w_p * z_{psrb}), \ \forall s \in \mathcal{V}^{\mathrm{S}}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{20}$$

$$\sum_{i\in\mathcal{V}^{\mathrm{D}}} \sum_{j\in\mathcal{V}^{\mathrm{S}}} \sum_{r\in\mathcal{R}^{\mathrm{C}}} \sum_{b\in\mathcal{B}_r} y_{ijrb} = \sum_{p\in\mathcal{P}^{\mathrm{C}}} w_p \tag{21}$$

$$\sum_{i\in\mathcal{V}^{\mathrm{D}}} \sum_{j\in\mathcal{V}^{\mathrm{S}}} \sum_{r\in\mathcal{R}^{\mathrm{C}}} \sum_{b\in\mathcal{B}_r} y_{jirb} \leq \sum_{r\in\mathcal{R}^{\mathrm{C}}} \sum_{b\in\mathcal{B}_r} c * y_{rb} - \sum_{p\in\mathcal{P}^{\mathrm{C}}} w_p \tag{22}$$

$$\sum_{j\in\mathcal{V}^{\mathrm{S}}} y_{ijrb} \leq c, \ \forall i \in \mathcal{V}^{\mathrm{D'}}, r \in \mathcal{R}_i^{\mathrm{C}}, b \in \mathcal{B}_r \tag{23}$$

$$y_{ijrb} + y_{jirb} = c * (x_{ijrb} + x_{jirb}), \ \forall i,j \in \mathcal{V}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{24}$$

$$y_{sjrb} \leq M * \sum_{p\in\mathcal{P}_s} z_{psrb}, \ \forall s \in \mathcal{V}^{\mathrm{S}}, j \in \mathcal{V}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \tag{25}$$

$$\sum_{j\in\mathcal{V}^{\mathrm{S}}} x_{ijrb} = 0, \ \forall i \in \mathcal{V}^{\mathrm{D'}}, r \notin \mathcal{R}_i^{\mathrm{C}}, b \in \mathcal{B}_r \tag{26}$$

$$\sum_{j\in\mathcal{V}^{\mathrm{S}}} x_{jirb} = 0, \ \forall i \in \mathcal{V}^{\mathrm{D}}, r \notin \mathcal{R}_i^{\mathrm{C}}, b \in \mathcal{B}_r \tag{27}$$

$$y_{isrb} \geq \sum_{p \in \mathcal{P}_s} w_p * z_{psrb} - c * (1 - x_{isrb}),\ \forall s \in \mathcal{V}^{\mathrm{S}}, i \in \mathcal{V}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \quad (28)$$

$$y_{ijrb} \geq 0,\ \forall i, j \in \mathcal{V}, r \in \mathcal{R}^{\mathrm{C}}, b \in \mathcal{B}_r \quad (29)$$

The objective function (1) is the same as in F(3.1), while some constraint sets are retained as in F(3.1), including (2)–(3) and (6)–(17). For each batch assigned to a cobot, the inflow minus outflow at each shelf node is equal to twice the weights of picked items on that shelf (see constraint set (20)). The total outflow from the real depots for all batches assigned to cobots indicates the total load, which should be equal to the weight of all items (see constraint set (21)), while the total inflow to the real depots should be smaller than or equal to the residual capacity of the used batches (see constraint set (22)). It is smaller in the case of unused batches. Constraint set (23) ensures that the outflow of each batch of a cobot leaving a copy depot i is less than or equal to that batch's capacity. Constraint set (24) ensures that the inflow plus outflow of each node equals the capacity of the batch of the cobot that visits that node. Constraint set (25) guarantees that the flow variable is set to zero; if not, a single item i is picked in batch b, so b does not need to visit shelf s. Finally, constraint sets (26) and (27) jointly ensure that a batch assigned to a cobot cannot leave and return to a copy/real depot other than its home depot. Constraint set (28) defines valid inequalities to tighten the formulation by reducing the search space. These constraints ensure that the remaining cobot load before reaching shelf s should be larger than or equal to the weights of all picked items at shelf s. Constraint set (29) defines new linear flow variables.

4. Variable neighborhood search algorithm for integrated order batching and cobot routing

As described before, integrated batching and routing is very complex, so it is solved in the literature mostly using metaheuristics to get a good solution within a reasonable computational time (see Table 1, the last column). In this work, we use variable neighborhood search (VNS) to solve this integrated problem. Variable neighborhood search is a relatively new metaheuristic (proposed by Mladenović & Hansen (1997)), and its basic idea is *a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley* (according to Gendreau & Potvin (2010)).

The steps of the basic VNS are shown in Algorithm 1. Here, $N_\kappa(\kappa = 1, ..., \kappa_{max})$ is a set of neighborhood structures, which we will explain together with shaking in Subsection 4.2. The stopping condition may be, for example, the number of iterations (γ) without improvement of x' or the maximum computational time t_{max}. It is worth mentioning that the algorithms we implemented in this section work for both mixed-shelves and dedicated storages. Therefore, our algorithms can also be applied to the integrated batching and routing problem for the traditional manual picking systems.

Algorithm 1: Steps of the basic VNS (based on Mladenović & Hansen (1997))

Initialization. Select the set of neighborhood structures $N_\kappa(\kappa = 1, ..., \kappa_{max})$ that will used in the search; find an initial solution x; choose a stopping condition

Repeat the following until the stopping condition is met:

1. Set $\kappa \leftarrow 1$;
2. Repeat the following steps until $\kappa = \kappa_{max}$:
   (a) *Shaking.* Generate a point x' at random from κ^{th} neighborhood of x ($x' \in N_\kappa(x)$);
   (b) *Local search.* Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
   (c) *Move or not.* If the local optimum x'' is better than the incumbent, move there $x \leftarrow x''$, and continue the search with $N_1(\kappa \leftarrow 1)$; otherwise, set $\kappa \leftarrow \kappa + 1$

In the following subsections, the implementation of each part of VNS for solving our integrated problem is described, including the building of an initial solution (Subsection 4.1), the shaking phase with the definition of the neighborhood structure (Subsection 4.2) and the local search methods (Subsection 4.3).

4.1. Initial solution

Algorithm 2: Greedy algorithm to generate an initial solution

```
   Result: Initial solution x
1  Function INITIAL(orders : 𝒪, depot : 𝒱^D)
2  ω=ASSIGNORDERSTODEPOTS(orders : 𝒪, depot : 𝒱^D)
3  for i ∈ 𝒱^D do
4  |   𝒪_i = 𝒪_i ∪ {o} ∀ω_oi = 1, o ∈ 𝒪
5  |   ℬ_i=ASSIGNORDERSTOBATCHESANDTOURS(𝒪_i)
6  |   x ← x ∪ ℬ_i
7  end
```

In this section a greedy heuristic is presented in Algorithm 2, which generates an initial solution x for our integrated problem. First, the algorithm AssignOrdersToDepots determines the assignment of order o to depot i, while keeping the balanced workload among depots (see line 2 in Algorithm 2). $\omega_{oi} = 1$ if order o is assigned to depot i; and 0 otherwise. Let $\mathcal{B}$ be the set of batches and tours for all orders, while $\mathcal{B}_i$ is the set of batches and tours for all orders assigned to depot i. Second, $\mathcal{B}_i$ for each depot i is determined in AssignOrdersToBatchesAndTours (see line 5). In the following, we will explain both algorithms in detail.

4.1.1. Assignment of orders to depots

Algorithm 3: Greedy assignment of orders to depots

Result: Assignment of variable $\omega_{io}, \forall i \in \mathcal{V}^{\mathrm{D}}, o \in \mathcal{O}$

Function $\textsc{AssignOrdersToDepots}(\mathcal{O};\, \mu_o, \hat{D}^o = \{\hat{d}_{o1}, ..., \hat{d}_{oi}\}\, \forall\, o \in \mathcal{O})$

Initialize $\omega_{oi} = 0\ \ \forall i \in \mathcal{V}^D,\ o \in \mathcal{O};$ weight of orders $w_o = \sum_{p \in \mathcal{P}_o^{\mathrm{C}}} w_p$; total weight at stations $w_i = \sum_{o \in \mathcal{O}} w_o \cdot \omega_{oi}$; sort $\mathcal{O}$ according to μ_o in descending order

for $o \in \mathcal{O}$ **do**

  $i \leftarrow \underset{i \in \mathcal{V}^D}{\operatorname{argmin}}(\hat{d}_{oi})$

  if $\left\lceil \frac{w_i + w_o}{c} \right\rceil \leq \left\lceil \frac{|\mathcal{B}|^*}{|\mathcal{V}^{\mathrm{D}}|} \right\rceil$ **then** `// check balancing`

    $\omega_{oi} \leftarrow 1;\ w_i \leftarrow w_i + w_o;\ \mathcal{O} \leftarrow \mathcal{O} \setminus o$

  else

    $\hat{D}^o \leftarrow \hat{D}^o \setminus \hat{d}_{oi}$

    $\textsc{AssignOrdersToDepots}(\mathcal{O},\, \hat{D}^o)$

  end

end

Algorithm 3 assigns orders to depots. For these types of assignment problems, it is common to first consider the distance from the object to be assigned (orders in our case) to all potential depots/stations (Giosa et al., 2002). For the problem at hand, this is already a challenge as each order consists of several items; further, each item can be stored in different shelves. Hence, the distance from an order to each depot is not well defined. To define a hypothetical loss for a given depot, we take for each item the distance of the shelf storing the item which is closest to the depot. These distances are then added up over all items of the order to get the hypothetical distance from that order to a depot. This definition of the distance from an order to a depot is formalized in Equation (30):

$$\hat{d}_{oi} = \sum_{p \in \mathcal{P}_o^{\mathrm{C}}} \min_{j \in \mathcal{V}_p^{\mathrm{SC}}} (d_{ij}) \qquad \forall o \in \mathcal{O},\, i \in \mathcal{V}^D, \tag{30}$$

where we write $\hat{d}$ to indicate the hypothetical nature of the distance between order o and depot i. These values are given as input for the algorithm.

Given the distances for all possible combinations of (o, i), we follow the approach of Giosa et al. (2002) and calculate for each order the potential savings of assigning that order to its closest depot instead of the second closest. This is done in accordance with the following equation:

$$\mu_o = \hat{D}^o_{(2)} - \hat{D}^o_{(1)}, \tag{31}$$

where $\hat{D}^o$ is the set of distances of depots from order o and subscripts (1) and (2) denote the first- and second-order statistic respectively, i.e. $\hat{D}^o_{(2)}$ denotes the second smallest and $\hat{D}^o_{(1)}$ the smallest distance in $\hat{D}^o$.

The savings are also given as input for the algorithm to prioritize orders, i.e. to determine the sequence in which the orders are assigned to the depots (see line 2). Due to the balancing constraint, orders cannot be assigned arbitrarily to stations, but must be assigned such that the number of batches at each station is as balanced as possible. So, orders where the second best choice would be far worse than the best choice are prioritized and therefore assigned prior to orders where the difference is not too large. A new order can still be assigned to a station if the number of batches required to fulfill all these orders is less than or equal to the minimum required number of batches $|\mathcal{B}^*|$ divided by the number of depots (see line 5). The number of batches required to fulfill all orders can be determined by rounding up the quotient of the total weight of these orders and the capacity c of a cobot. Likewise, the overall minimum number of batches required to carry out all orders can be determined as follows:

$$|\mathcal{B}|^* = \left\lceil \frac{\sum_{o\in\mathcal{O}} \sum_{p\in\mathcal{P}^C_o} w_p}{c} \right\rceil . \tag{32}$$

If an order cannot be assigned to the station closest to it due to the balancing constraint, the station is removed from the set of possible stations for that order (see lines 8–9) and a new savings value for that order is determined. As the new savings value might affect the precedence of the remaining orders, this is done by recursively calling function AssignOrdersToStations with the reduced set of stations and only the orders that are not assigned yet. This procedure is repeated until all orders are assigned.

4.1.2. Order batching and routing

In Algorithm 4, the orders $o \in \mathcal{O}_i$ of a depot $i \in \mathcal{V}^{\mathrm{D}}$ need to be assigned to batches while forming tours without violating the batch capacity c. To initialize a new batch (see line 4), the initial weight, tour and empty order set in the batch $\mathcal{O}_{temp}$ are given. An order $o \in \mathcal{O}_i$ is selected by calculating the minimum possible distance from any item of o to any item of the orders $\in \mathcal{O}_{temp}$ in the batch (see line 6; if $\mathcal{O}_{temp}$ is empty, then the order o is randomly chosen), while the load capacity of a cobot is kept (see line 7). As an item can be stored in many shelves, the shelf that minimizes the distance to any of the shelves visited in the current batch is determined. Therefore, each time a new order is assigned to a batch, the tour of this (partial) batch is updated (see line 10). Therefore, for every new item that has to be included in the current tour, and for every shelf this item is stored in, the position-shelf combination that minimizes the distance added to the tour is determined using Algorithm 5 (see lines 6–9). The additional distance created by adding a new node t_j between nodes t_i and t_k can easily be calculated using the following formula:

$$d^{add} = d_{t_i,t_j} + d_{t_j,t_k} - d_{t_i,t_k}, \tag{33}$$

Algorithm 4: Create a set of batches and tours for orders handled in depot i

Result: Set of batches $\mathcal{B}_i$

Function ASSIGNORDERSTOBATCHESANDTOURS($\mathcal{O}_i$)
$batchnr := 1$
while $\mathcal{O}_i != \emptyset$ **do**
    $weight := 0$; $tour := [d_i, d_i]$;$\mathcal{O}_{temp} \leftarrow \emptyset$
    while $weight < c_i$ **do** `// keep batch capacity`
        for $o \in MinDist(\mathcal{O}_i, \mathcal{O}_{temp})$ **do** `// choose a new order`
            if $weight + \sum_{p \in \mathcal{P}_o^{\mathrm{C}}} w_p < c_i$ **then** `// check batch capacity`
                $weight := weight + \sum_{p \in \mathcal{P}_o^{\mathrm{C}}} w_p$
                $\mathcal{O}_{temp} \leftarrow \mathcal{O}_{temp} \cup \{o\}$
                $tour \leftarrow$GREEDYTOUR($tour, o$) `// update the tour`
                $\mathcal{O}_i \leftarrow \mathcal{O}_i \setminus o$
            end
        end
    end
    $\mathcal{B}_{i,batchnr} \leftarrow (\mathcal{O}_{temp}, tour)$; $batchnr := batchnr + 1$
end

Algorithm 5: Create a greedy cobot tour for an order

Result: Updated *tour* by picking order o

Function GREEDYTOUR($tour := [t_1, ..., t_n, t_1]$, o)
Initialize $d_{best}^{add} \leftarrow \infty$
for $p \in \mathcal{P}_o^{\mathrm{C}}$ **do** `// iterate over items of new order`
    for $s \in V_p^{\mathrm{SC}}$ **do**
        if $n_{ps} > 0$ **then** `// check if shelf still has units`
            $j \leftarrow \underset{j \in \{2,...,|tour|-1\}}{\operatorname{argmin}} \left(d_{t_{j-1},s} + d_{s,t_j} - d_{t_{j-1},t_j}\right)$
            $d_j^{add} \leftarrow d_{t_{j-1},s} + d_{s,t_j} - d_{t_{j-1},t_j}$
            if $d_j^{add} < d_{best}^{add}$ **then**
                $d_{best}^{add} \leftarrow d_j^{add}$; $s^* \leftarrow s$; $j^* \leftarrow j$
            end
        end
    end
    $tour \leftarrow [t_1, ..., t_{j^*-1}, s^*, t_{j^*}, ..., t_n, t_1]$ `// insert node`
    $n_{ps^*} \leftarrow n_{ps^*} - 1$ `// decrease supply by one`
end
return *tour*

where the best position in the tour for a new shelf s is that which minimizes this new distance with respect to all possible positions j in the tour (see line 6 of Algorithm 4).

4.2. Shaking

Algorithm 6: Shaking

Result: Solution x' from neighborhood $\mathcal{N}_\kappa(x)$ of current solution x

1 **Function** Shaking(x, κ)
2 Determine total weight w_b per batch $b \in B$; $\mathcal{O}_{temp} \leftarrow \emptyset$
3 **repeat** κ **times**
4     $d \leftarrow$ EliminateBatch($\mathcal{B}$) `// destroy a batch`
5     $\mathcal{B} \leftarrow \mathcal{B} \setminus d$; $\mathcal{O}_{temp} \leftarrow \mathcal{O}_{temp} \cup \mathcal{O}_d$
6 **end**
7 **for** $o \in \mathcal{O}_{temp}$ **do** `// repair orders`
8   **if** $\nexists b \in \mathcal{B} : w_b + w_o \leq c$ **then** `// add orders to a new batch`
9         $b_{|\mathcal{B}|+1} \leftarrow o$; $\mathcal{B} \leftarrow \mathcal{B} \cup b_{|\mathcal{B}|+1}$; $b \leftarrow b_{|\mathcal{B}|+1}$
10   **else** `// add orders to a existing batch`
11         $b \leftarrow$ select a random batch $b \in \mathcal{B}$ for which $w_b + w_o \leq c$
12         $w_b \leftarrow w_b + w_o$; $\mathcal{O}_b \leftarrow \mathcal{O}_b \cup o$
13   **end**
14     $tour_b \leftarrow$ GreedyTour($tour_b, o$)
15 **end**
16 **if** $|B| > |B^*|$ **then**
17     $B \leftarrow$ ReducedBatches(B)
18 **end**

After an initial solution is obtained using the algorithms described in the previous subsection, its neighborhood can be searched for better solutions. The Shaking operator is mainly used to search increasingly distant neighborhoods of the current incumbent solution, which are then searched for a local optimum (Mladenović & Hansen, 1997). The Shaking operator helps escaping local optima by searching a larger neighborhood with different basins. If this neighborhood still does not yield a better solution, an even larger neighborhood is searched until an improvement is made or the most distant neighborhood $\mathcal{N}_\kappa$ is reached.

To search larger neighborhoods, this work proposes a destroy-and-repair operator in shaking (see Algorithm 6). During the destroy phase, κ batches from the set of batches $\mathcal{B}$ are selected and destroyed (see line 4). The selection of a batch happens through Algorithm 7. If there exists a depot i with more batches than any other depots, a batch from i is selected in order not to violate the capacity constraints (see line 2 of Algorithm 7). Moreover, the probability of a batch leaving from depot i being selected is dependent on its workload, so that batches that are already close to the capacity limit have a lower chance of being selected (see line 4). If there is not such depot as described above, then a batch is chosen according to the same principle, but from all batches (see line 6).

The EliminateBatch function returns the respective batch to the Shaking operator, which then excludes the batch from the set of batches (see line 5 of Algorithm 6).

Algorithm 7: Eliminate a random batch

Function EliminateBatch($\mathcal{B}$)
if $\exists\, i \in \mathcal{V}^{\mathrm{D}} : |\mathcal{B}_i| - |\mathcal{B}_j| > 0 \,\forall\, j \in \mathcal{V}^{\mathrm{D}} \setminus i$ **then**
    $i \leftarrow \underset{i \in \mathcal{V}^{\mathrm{D}}}{\operatorname{argmax}}(|\mathcal{B}_i|)$
    $p_b \leftarrow \frac{1-(w_b/c)}{\sum_{\ell \in \mathcal{B}_i} 1-(w_\ell/c)} \quad \forall b \in \mathcal{B}_i$
else
    $p_b \leftarrow \frac{1-(w_b/c)}{\sum_{\ell \in \mathcal{B}} 1-(w_\ell/c)} \quad \forall b \in \mathcal{B}$
end
$d \leftarrow$ pick random b according to p_b
return d

The repair phase in Algorithm 6 then reassigns the orders in the destroyed batches to the remaining batches where the additional weight opposed by the new order cannot violate the capacity constraints (see line 8). In a case that an order in a destroyed batch cannot be reassigned to any existing batches due to the capacity constraint, it will be put in a new batch $b_{|\mathcal{B}|+1}$ (see line 9). Otherwise, it will assigned to a random batch that it still fits into (see lines 11–12).

Every time an order is added to a batch, the GreedyTour algorithm is applied in order to assign shelves to items and update the tour in a greedy manner (see line 14).

If the number of batches is higher than the minimum possible number of batches $|\mathcal{B}|^*$ (see Equation (32)), then we will try to reduce the total number of batches through ReduceBatches in Algorithm 8, which was originally proposed by Alvim et al. (1999) (see lines 16–17 of Algorithm 6).

In Algorithm 8, just like in the Shaking operator, the EliminateBatch function chooses a batch d to make sure that the balancing constraint is not violated (see line 3 in Algorithm 8). The orders in batch d are then split up among the other batches in $\mathcal{B}'$ so that the total weight of the other batches remains as small as possible; however, the capacity constraint may be violated (see lines 4–5). If no batch exceeds the maximum allowed weight, the solution $\mathcal{B}'$ is accepted (see lines 5, 7–8); otherwise, a Repair operator is executed (see line 10). If the final solution still has batches that exceed the maximum weight, the neighborhood solution is discarded, otherwise it is accepted (see lines 11–12 of Algorithm 8).

Algorithm 8: Minimize number of batches

Function $\textsc{ReduceBatches}(\mathcal{B})$
while improvement **do**
  improvement $\leftarrow$ False; $d \leftarrow \textsc{EliminateBatch}(\mathcal{B})$; $\mathcal{B}' \leftarrow \mathcal{B} \setminus d$
  for $o \in \mathcal{O}_d$ **do**
    $b \leftarrow \underset{b\in\mathcal{B}'}{\operatorname{argmin}}(w_o + w_b)$; $\mathcal{O}_b \leftarrow \mathcal{O}_b \cup o$
  end
  if $w_b \leq c \ \forall b \in \mathcal{B}'$ **then**
    $\mathcal{B} \leftarrow \mathcal{B}'$; improvement $\leftarrow$ True
  else
    $\bar{\mathcal{B}} \leftarrow \textsc{Repair}(\mathcal{B}')$
    if $w_b \leq c \ \forall b \in \bar{\mathcal{B}}$ **then**
      $\mathcal{B} \leftarrow \bar{\mathcal{B}}$; improvement $\leftarrow$ True
  end
end
return $\mathcal{B}$

Algorithm 9: Repair an infeasible neighborhood solution

1 **Function** $\textsc{Repair}(\mathcal{B}')$
2 **while not** $w_b \leq c \ \forall b \in \mathcal{B}'$ **and** improvement **do**
3   improvement $\leftarrow$ False
4   $b^{\text{inf}} \leftarrow$ draw next batch from $\{b \in \mathcal{B} : \ w_b > c\}$
5   **for** $b^{\text{other}} \in \mathcal{B}' \setminus b^{\text{inf}}$ **do**
6     $\bar{b}^{\text{inf}}$, $\bar{b}^{\text{other}} \leftarrow$ apply differencing method to b^{other} and b^{inf}
7   **if** $|\bar{b}^{\text{inf}} - \bar{b}^{\text{other}}| < |b^{\text{inf}} - b^{\text{other}}|$ **then**
8       $\mathcal{B}' \leftarrow (\mathcal{B}' \cup \{\bar{b}^{\text{inf}}, \bar{b}^{\text{other}}\}) \setminus \{b^{\text{inf}}, b^{\text{other}}\}$
9   improvement $\leftarrow$ True
10   **end**
11   **end**
12 **end**
13 **return** $\mathcal{B}'$

Within the Repair operator in Algorithm 9, the infeasible batch b^{inf} is fixed while iterating over all other batches to find a batch b^{other} to swap. The weight of b^{other} is as close as possible to the weight of the infeasible batch b^{inf} (see line 6). The reassignment of orders to $\mathcal{B}'$ is performed using the differencing method proposed by Karmarkar & Karp (1982). This algorithm saves the weights of all the items in a sorted list and in each iteration extracts the two largest numbers from it, computes the difference between them and places only the difference back in the list. This represents a decision to put each of the respective orders into different batches. This procedure is repeated until there is only one number left, which is the value of the final subset difference (Karmarkar & Karp, 1982).

The solution obtained by redistributing the orders is accepted if the difference in the weight of the two corresponding batches decreased (see line 7). And the repair procedure is repeated for each batch that is infeasible or has become infeasible by applying the differencing method until all batches meet the capacity constraints or a batch cannot be balanced with any other batch.

4.3. Local search operator

Algorithm 10: Exchange orders

Result: Updated solution x'

1 **Function** ExchangeOrders(x', κ)
2 **repeat** $iter^{max}_{exchange}$ **times**
3 $\quad \mathcal{B}_{temp} \leftarrow \emptyset$
4 $\quad$ **repeat** κ **times**
5 $\quad\quad$ draw orders o_i and o_j from batches b_i and b_j respectively, $i \neq j$
6 $\quad\quad$ **if** *no capacity constraints are violated* **then**
7 $\quad\quad\quad \mathcal{O}_{b_i} \leftarrow (\mathcal{O}_{b_i} \cup o_j) \setminus o_i$
8 $\quad\quad\quad \mathcal{O}_{b_j} \leftarrow (\mathcal{O}_{b_j} \cup o_i) \setminus o_j$ `// exchange orders`
9 $\quad\quad$ **end**
10 $\quad\quad \mathcal{B}_{temp} \leftarrow \mathcal{B}_{temp} \cup \{b_i, b_j\}$
11 $\quad$ **end**
12 $\quad x'' \leftarrow$ OptimizeShelves($\mathcal{B}_{temp}$)
13 $\quad$ **if** $f(x'') < f(x')$ **then**
14 $\quad\quad x' \leftarrow x''$
15 **end**
16 **return** x'

After moving to a different basin of attraction using the Shaking algorithm, we use a local search operator ExchangeOrders in Algorithm 10 to find the local minima. Since the greedy initial solution and the Shaking operator may come up with batches that have to approach many different shelves due to few intersections of shelves storing the different items, this work proposes a swap operation of orders in different batches, which is described in Algorithm 10. By doing so, one may replace an order in a batch with another order in a different batch that reduces the distance of the tour for the corresponding cobots.

In order to systematically change the neighborhood during the descent phase, we select κ pairs of orders from two different batches randomly and exchange them between the batches if possible (see lines 4–5). The exchange is possible if the absolute difference in the weight of the pair of orders still fits in the batch that currently contains the lighter order (see lines 6–8). After that, in order to minimize the number of shelves that are included in a tour, a function OptimizeShelves for the item-shelf assignment is carried out for all the batches that have been changed (see line 12). If the resulting solution yields a better fitness than the current solution, the candidate solution is adopted, otherwise it is discarded (see lines 13–14). The swap operator repeats until a maximum number of iterations $iter^{max}$ is reached to ensure that a local minima is found.

Algorithm 11: Optimization of shelf assignment

Result: Updated solution x'

1 **Function** OptimizeShelves($\mathcal{B}_{temp}$)
2 **foreach** $b \in \mathcal{B}_{temp}$ **do**
3     $\mathcal{P}_b^{\mathrm{C}} = \bigcup_{o \in \mathcal{O}_b} \mathcal{P}_o^{\mathrm{C}}$;$tour_b \leftarrow \emptyset$
4   **while** $\mathcal{P}_b^{\mathrm{C}} \neq \emptyset$ **do**
5         $\mathcal{P}_s^{\mathrm{C}} = \bigcup_{p \in \mathcal{P}_b^{\mathrm{C}}} \{p : s \in \mathcal{V}_p^{\mathrm{SC}}\} \quad \forall s \in \mathcal{V}^{\mathrm{SC}}$
6         $s^* \leftarrow \underset{s \in \mathcal{V}^{\mathrm{SC}}}{\mathrm{argmax}}(|\mathcal{P}_s^{\mathrm{C}}|)$
7         $tour_b \leftarrow tour_b \cup s^*$
8         $\mathcal{P}_b^{\mathrm{C}} \leftarrow \mathcal{P}_b^{\mathrm{C}} \setminus \mathcal{P}_{s^*}^{\mathrm{C}}$
9   **end**
10     $tour' =$ SimulatedAnnealing($tour_b$)
11     $x' \leftarrow x' \cup (b, tour')$
12 **end**
13 **return** x'

Algorithm 12: Simulated annealing algorithm

Result: Improved sequence of visiting pods *tour*

1 **Function** SimulatedAnnealing($tour := [t_1, ..., t_n, t_1]$)
2 $iter \leftarrow 0$; $iter_{no\_change} \leftarrow 0$
3 $T = d_{ij}^{max} - d_{ij}^{min}$ `// difference between max and min distances`
4 **repeat**
5   **for** $i \in \{1, ..., n-2\}$ **do**
6   **for** $j \in \{i+2, ..., n\}$ **do**
7             $tour' \leftarrow [t_1, ..., t_i, t_{j-1}, ..., t_{i+1}, t_{j+1}, ..., t_n, t_1]$ `// 2-opt`
8   **if** $U(0,1) < P(tour, tour', T)$ **then**
9                 $tour \leftarrow tour'$ `// accept solution`
10                 $iter_{no\_change} \leftarrow 0$
11   **else**
12                 $iter_{no\_change} \leftarrow iter_{no\_change} + 1$ `// discard changes`
13   **end**
14   **end**
15   **end**
16     $T \leftarrow \alpha T$
17     $iter \leftarrow iter + 1$
18 **until** $T \leq T^{min}$ **or** $iter \geq iter^{max}$ **or** $iter_{no\_change} \geq iter_{no\_change}^{max}$
19 **return** *tour*

Algorithm 11 first determines for a given batch b and for each shelf the number of items of batch b that are stored in the respective shelf (see lines 2–3 in Algorithm 11). Then, the shelf that contains the most items in batch b is selected and added to the tour (see lines 5–7). All the items of the batch that can be collected at that shelf are then excluded from the set of items that have to be picked up during the batch (line 8), and the procedure repeats until no more items have to be assigned to a shelf.

In the end, the resulting tour is optimized by SimulatedAnnealing (see line 10), where a simulated annealing (SA) algorithm is proposed in combination with a 2-opt operator (see Algorithm 12). The 2-opt operator exchanges two edges of a given tour to generate a neighborhood solution (see line 7). Following the SA framework, this neighborhood solution is accepted with a probability determined by the well-known Metropolis criteria (see line 8). For faster convergence, the number of iterations where no improvement is made is counted (see line 12), and if this number exceeds a threshold, the best solution is returned. Other stopping criteria we use include a minimum temperature which is eventually reached due to the cooling phase in line 16, and a general maximum number of iterations after which the algorithm stops.

5. Computational evaluation

A series of numerical studies was conducted to evaluate (1) the solution qualities of three-index and two-commodity flow formulations, (2) whether the mixed-shelves policy works well for both single-line and multi-line orders (in both small and large warehouses). The computational results in this section were conducted on a PC with an Intel Core i7-7700K CPU @4.20GHz and 32GB RAM running Microsoft Windows 10 in 64-bit mode. Our models were solved by Gurobi 8.1.1 via Python version 3.7. In the following subsections, we first detail instance generation and the setting of parameters in Subsection 5.1. The results of the computational study of our models are presented in Subsection 5.2, while the tuned parameters and the results of VNS are presented in Subsection 5.3 and 5.4 respectively.

5.1. Instance generation and parameter setting

Table 2 shows the parameters for small and large instances, including layout (including number of shelves), number of orders, average number of order lines, three different order sets, number of SKUs, storage policy, number of depots and number of cobots per depot.

We use the same layout for traditional manual warehouses as described in Van Gils et al. (2019). For example, we have a small layout with 360 shelves (6 aisles × 60 shelves per aisle), in Figure 4. In such a layout, we define the width of each cross-aisle as 6 meters and the width of each pick aisle as 3 meters, while we define each shelf with 0.9 meters wide and 1.3 meters deep. Picking aisles are two-sided and wide enough for two-way travel of cobots (for example in Lee & Murray (2019) for the same assumption); therefore we assume that the blocking of cobots to be minimum. Another small layout we test includes 24 shelves (2 aisles × 12 shevels per aisle), while we use a large layout with 3240 shelves (18 aisles × 180 shelvels per aisle).

We generate different sets of orders (small instances: 10 and 20 orders; large instances: 100, 200 and 300 orders) randomly based on a given average number

	Small instances		Large instances
Layout	2×12 shelves	6×60 shelves	18×180 shelves
# orders	10, 20	10, 20	100, 200, 300
# order lines (average)	1.6, 5	1.6, 5	1.6, 5
Order sets	a, b, c	a, b, c	a, b, c
# SKUs	24	360	3240
Storage policy	dedicated, mixed(5)	dedicated mixed(5) mixed(10)	dedicated, mixed(10) mixed(20)
# depots	2	2	2
# robots per depot	1	1	1
# instance	24	36	54

Table 2: Parameters for instance generation.

of order lines between 1 and 10. According to Van Gils et al. (2019), the number of order lines impacts the number of batches created and consequently the complexity of the problem. Therefore, we test two different average numbers of order line per order: 1.6 (for e-commerce companies; see De Koster et al. (2012)) and 5. We don't test larger average numbers of order lines than 5, since we want to keep the weight of each order smaller than the capacity of each cobot. We set the capacity of each cobot as in the real-world application to be 18kg according to Wulfraat (2016). That means the batch capacity is limited to 18kg. We call the orders with an average of 1.6 order lines *single-line orders*, and the orders with an average of 5 order lines *multi-line orders*. In order to get rid of randomness by generating orders, we generate three different sets (a,b,c) for each order size. The weights of items are generated randomly between 0.5kg and 3kg (small-sized items according to pratical experience).

The number of shelves sharing an SKU also seems to impact the complexity of the integrated problems (storage policy). We test two storage policies, namely dedicated and mixed-shelves. Recall that *dedicated storage* means that the items of an SKU are stored in only one shelf, while *mixed-shelves storage* means that the items of an SKU can be spread over several shelves. We also set the number of shelves on which the items of an SKU can be stored to 5, 10 and 20 to see the complexity and efficiency of mixed-shelves storage policy. Without loss of generality, we set the number of depots to 2, with one cobot per depot for all instances. In our study, we do not intend to support decisions on the number of cobots and the number of depots in practice.

Note that we get the shortest path for each two nodes i and j with the Dijkstra algorithm by using NetworkX (Hagberg et al. (2008)).

5.2. Results of mathematical models

Table 3 shows the results of the comparison between the two-commodity model (in short: two) and the three-index model (in short: three) for small

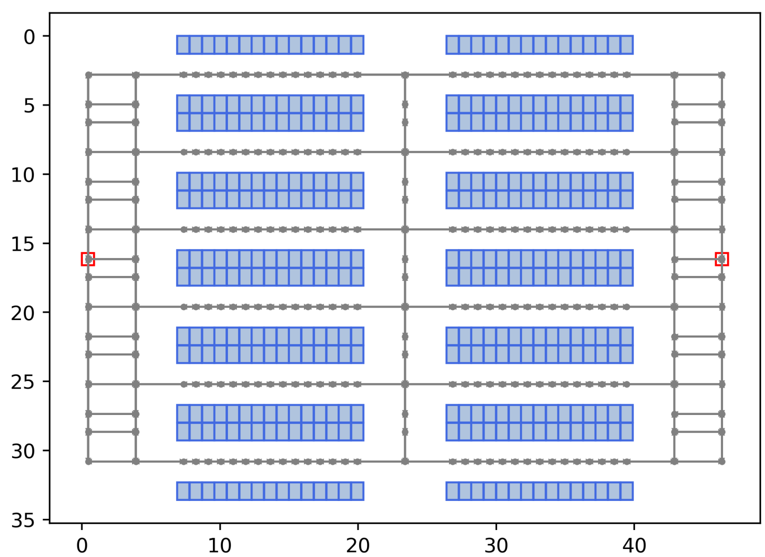

Figure 4: The layout with six pick aisles and one cross-aisle; The depots are marked in red on the left- and right-hand sides.

instances of 24 and 360 shelves respectively. Note that we use alternative II in constraint set C in Section 3.1 to determine the minimum number of batches for each cobot r, $|\mathcal{B}_r^*|$ (see Equation 19). It is possible that there are no feasible solutions to achieve $|\mathcal{B}_r^*|$. In this case, the minimum number of batches will be increased by one or two, etc. The results of applying both storage policies, namely dedicated and mixed-shelves (in short: mixed(5), mixed(10)) storage policies, for single- and multi-line orders are also shown in the table. Based on the results, we can gain some knowledge of the models of integrated order batching and routing. First, the two-commodity flow formulation provides better performance by achieving lower gaps with smaller computational time for solving our integrated problem. The improvements in the time and optimum gap are marked in green. Second, based on the results we can get the following factors that increase the complexity of the model:

- the number of orders
- the storage policy
- the average number of order lines
- the number of shelves.

Instances with 24 shelves			10 orders single-line	10 orders multi-line	20 orders single-line	20 orders multi-line
dedicated	three	gap (%)	0	6.2	6.1	37.0
		time (s)	10.47	9212	7216	t.l.
	two	gap (%)	0	1.3	0	22.1*
		time (s)	0	5755	491.7	t.l.
mixed(5)	three	gap (%)	0	48.5	11.4	-
		time (s)	5286	t.l.	t.l.	t.l.
	two	gap (%)	0	27.8	0	-
		time (s)	216.3	t.l.	2964	t.l.
Instances with 360 shelves						
dedicated	three	gap (%)	0	6.64	0	-
		time (s)	3	t.l.	2606	t.l.
	two	gap (%)	0	5	0	-
		time(s)	1.3	t.l.	1500	t.l.
mixed(5)	three	gap (%)	33	-	-	
		time (s)	t.l.	t.l.	t.l.	t.l.
	two	gap (%)	0	-	28.3	-
		time (s)	566.5	t.l.	t.l.	t.l.
mixed(10)	three	gap (%)	42.5	-	-	-
		time (s)	t.l.	t.l.	t.l.	t.l.
	two	gap (%)	0	-	-	-
		time (s)	2636	t.l.	t.l.	t.l.

Table 3: Comparison of results of three-index and two-commodity models (the average gap and running time of instance sets a, b, c) for instances with 24 and 360 shelves. "-" means that there is no feasible integer solution found within the time limit (t.l.: 14,400 seconds). "*" means that some instances cannot find a feasible solution within the time limit. Note that the model for each instance was solved using eight threads.

The number of orders. If more orders are processed in a model, then more computational time is needed. The reason is obvious, since the set of physical items P^C for picking is larger for more orders, assuming all other factors are the same.

The average number of order lines. The models with single-line orders are more easily solved than those with multi-line orders, since single-line orders include smaller sets of physical items $\mathcal{P}^{\mathrm{C}}$ for picking (see Table 3, columns single-line vs. multi-line), assuming the other factors are the same. Therefore the number of decision variables x_{ijrb} and z_{psrb} increases. Note that we should consider the number of orders and the average number of order lines together, since they influence the set of physical items for picking and the number of batches created, and consequently the complexity of the problem. Therefore, the instances of 10 multi-line orders are more difficult to solve than the instances of 20 single-line orders.

The storage policy. The models using the dedicated storage policy require less computational time compared with the models using the mixed-shelves storage policy, assuming the other factors are the same. In Table 3, we can see that if more shelves contain items of an SKU (mixed(5) and mixed(10) vs. dedicated), the problem is more complex, since the set of possible shelves including item p, V_p^{SC}, is larger for all picking items in mixed-shelves storage. Therefore the number of decision variables x_{ijrb} and z_{psrb} increases.

The number of shelves (Layout). In Table 3, it is easy to see that the models for a larger warehouse are more difficult to solve, since the set of shelves including all items in $\mathcal{P}^{\mathrm{C}}$ is larger. Therefore, the number of decision variables x_{ijrb} and z_{psrb} increases.

	10 orders		20 orders	
	single-line	multi-line	single-line	multi-line
Instances of 24 shelves				
dedicated	59.2 (0%)	220.2 (1.3%)	105.8 (0%)	364.0* (22%)
mixed(5)	44.5 (0%)	139.0 (27.8%)	71.0 (0%)	-
%	-24.8	-36.9	-32.9	-
Instances of 360 shelves				
dedicated	232.23 (0%)	801.4 (9.3%)	352.3 (0%)	-
mixed(5)	106.6 (0%)	-	177.0 (28.3%)	-
%	-54.1	-	-49.8	-
mixed(10)	89.97 (0%)	-	-	-
%	-61.2	-	-	-

Table 4: The average distances for instance set a, b, c of 24 and 360 shelves solved with the two-commodity formulation with the gap in %. The lines beginning with "%" show the pecentage improvement in the results of mixed-shelves storage compared with the dedicated storage policy. "-" means that there is no feasible integer solution found within the time limit (t.l.: 14,400 seconds). "*" means that some instances cannot find a feasible solution within the time limit.

In Table 4, the mixed-shelves policy definitely works better than the dedicated policy for single-line orders (in other words, shorter distances). In a larger warehouse (with 360 shelves), the saving is more significant. The reason is obvious: First, the distances between visiting shelves are larger in a larger warehouse. Second, it is as expected that if more shelves contain items of an SKU, more distance can be saved for picking an item of this SKU (see mixed(5) vs. mixed(10) in *Instances of 360 shelves*), since the probability is higher in mixed(10) that a shelf near-by contains the required item.

Based on the results of the two-commodity flow model in Tables 3 and 4, 60% of instances (36 out of 60) have not been solved optimaly by Gurobi within the run time limit of 4 hours. Among these, for 23 instances no feasible integer solution has been found. Layout, storage policy, the number of orders and the average number of order lines have effects on the number of non-optimal solutions. All of those instances with no feasible integer solution are those in which we apply mixed-shelves storage. This demonstrates the complexity of the problem. Therefore it is worth developing an efficient metaheuristic to gain more results.

5.3. Parameter tuning

It is often beneficial to tune algorithm parameters. In our VNS, parameter tuning is performed on the small instances (in total 60 instances). In our experiment, we tune two parameters in Algorithm 1, namely γ (parameter defining the number of iterations without improvement of x') and κ_{max} (parameter defining the neighborhood structures). Note that we don't tune local search parameters in our experiment, such as $iter^{max}_{exchange} = 5, T^{min} = 0.1, iter^{max} = 1000, iter^{max}_{no\_change} = 15, \alpha = 0.975$, which we apply in the local search improve routes within a batch (Algorithms 10 and 12), since they are not related to other algorithms. We test γ with the values 100, 200, 300 and 400, while κ_{max} is set to be 3 and 4. And we set t_{max} to be 3600 seconds. Five repetitions are run to eliminate randomness. Consequently, we have in total $60 \times 4 \times 2 \times 5 = 2400$ observations. Our tuning results show that the computational time increases about linearly with the increasing values of γ, but different γ-values have little effect on solution quality. So, we choose $\gamma = 100$ for the rest of the experiment. From our observation, $\kappa_{max} = 4$ brings a small improvement, averaging 2.15%, compared with the results of $\kappa_{max} = 3$ within a similar computational time using the same value of γ. So we will choose $\kappa_{max} = 4$ in the further experiment.

5.4. Results of our VNS

	10 orders				20 orders			
	single-line		multi-line		single-line		multi-line	
Instances of 24 shelves								
dedicated	**1**	59.2 (0%)	**6**	220.2 (0%)	**3**	106.4 (0.6%)	**8**	315.4 (299.6* (-17.7%))
mixed(5)	**2**	44.5 (0%)	**7**	132.2 (-4.9%)	**5**	72.8 (2.4%)	**9**	200.7
%		-24.8		-40		-31.6		-36.4
Instances of 360 shelves								
dedicated	**1**	240.06 (3%)	**6**	801.4 (0%)	**3**	356.0 (1%)	**8**	1299
mixed(5)	**2**	110.1 (3%)	**7**	406.8	**5**	170.6 (-3.6%)	**9**	673.5
%		-53.4		-49.3		-52.1		-48.2
mixed(10)	**4**	94.6 (5%)	**10**	305.7	**11**	150.9	**12**	543.28
%		-60.6		-62		-58		-58.2

Table 5: The average distances for instance set a, b, c of 24 and 360 shelves solved with the VNS in five repetitions with parameters $\gamma = 100, \kappa_{max} = 4$. The lines beginning with % shows the percentage improvement of the results of mixed-shelves storage compared with the dedicated storage policy. Each number in brackets is the percentage difference between the VNS solution and the solution solved by Gurobi in Table 4, if there is a feasible integer solution. The blue numbers in cells sort the instances based on the computational time solved with Gurobi. The number marked by * is the comparable average distance for those instances with feasible solutions solved by Gurobi.

In Table 5, the average distances for small instances with $\kappa_{max} = 4$ and $\gamma = 100$ are shown. Each result represents the average value of three instance sets a, b, c in five repetitions. For the instance sets where we can get optimal solutions with the two-commodity model within 4 hours, our VNS provides solutions with the gap within 5% (instance sets of 24 shelves with numbers 1 to 5, and instance sets of 360 shelves with numbers 1 to 4, in Table 5). Note that the number of each instance set represents its complexity for each layout (24 and 360 shelves respectively). For the instance sets with numbers 6 to 8, some of which give us feasible solutions within 4 hours from the two-commodity model, our VNS provides either the same or better solutions. The computational time of our VNS is very competitive compared with the time solved by the solver (see Figure 5). It decreases substantially when the problem is solved by the VNS. Note that the time is not directly comparable, since the model for each instance was solved using eight threads, while the VNS is solved with one thread. As shown in Figure 5, the computational time of each instance solved by the VNS is less than 15 minutes. Figure 6 shows that the computational time applying the VNS is dependent on the layout and the number of orders. The storage policy has a small effect on the computational time. Our VNS reduces the complexity of the mixed-shelves storage (not as in the model), since the GreedyTour algorithm provides the same complexity for both storage policies ($\mathcal{O}(|\mathcal{P}_o^C|)$). The computational time strongly increases as the number of orders increases, since this increases the number of batches and the complexity of the routing problem. The similar increase in complexity caused by the increasing number of orders can be found in the results of the Gurobi solver (see Table 4). Our results show that the VNS algorithm is an efficient tool for solving the integrated order batching and routing in multi-depot mixed-shelves warehouses, at least for our small instances.

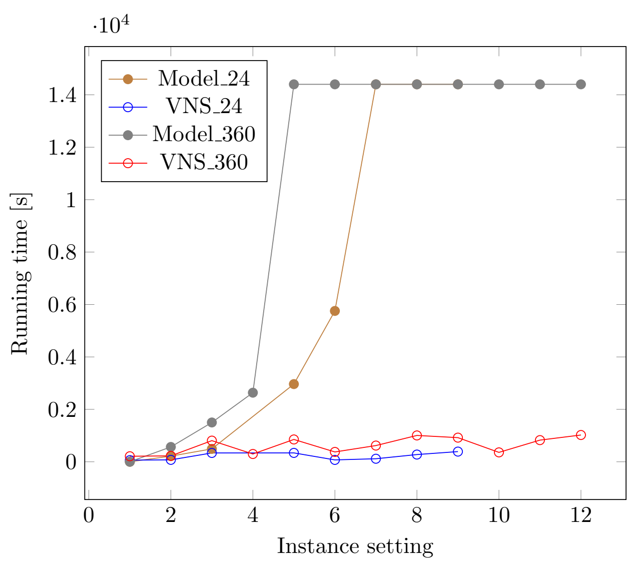

Figure 5: The comparison of running time between the two-commodity flow network model and our VNS. The number in the x-axis corresponds the instance set with blue number in Table 5.

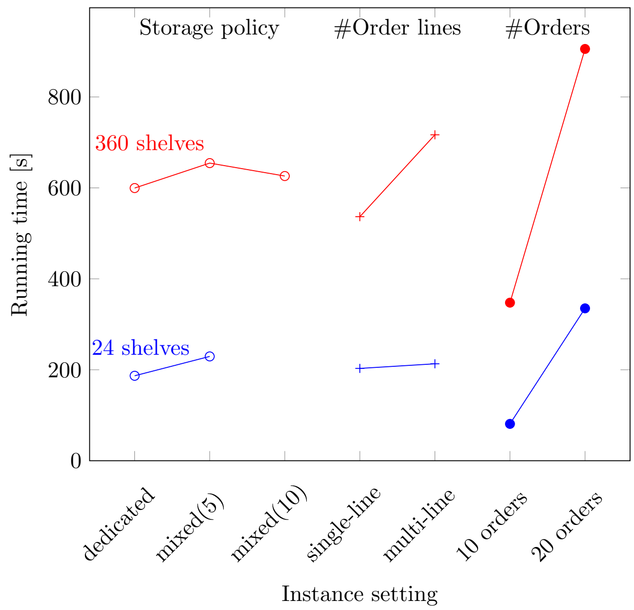

Figure 6: The average computational time of the VNS algorithm for instances which are classified by factors including the storage policy (dedicated vs. mixed(5) vs. mixed(10)), the average number of order lines (single-line, multi-line), the number of orders (10 orders vs. 20 orders) and the number of shelves (24 shelves vs. 360 shelves).

Furthermore, we can continue with our statement in Subsection 5.2 that the mixed-shelves storage policy works well for both single-line and multi-line orders. However, the items in a multi-line order are spread among several shelves, so several shelves should be visited. Using mixed-shelves storage, the probability is higher that a shelf nearby contains the required items compared with dedicated storage. Furthermore, in our AVG-assisted system, the limited capacity of cobots does not allow multi-line orders with very large order lines to be accepted, unless the splitting of orders is allowed (see e.g. in Xie et al. (2020)). Order splitting is outside the scope of this paper. Due to this setting of limited capacity of cobots, multi-line orders in mixed-shelves storage can save up to 62% on driving distances compared with the results in dedicated storage. Similar savings (up to 67%) can also be achieved for the multi-line orders in large instances (see Table 6). Based on the results of small instances, the computational time might be strongly increased for the large instances, since they have a larger layout and larger numbers of orders. Therefore, we limit the computational time for solving the large instances to 1 hour.

6. Conclusions

In this work, we formulate and solve integrated order batching and routing in AGV-assisted warehouses. In such warehouses, each AGV (in our case, a cobot) is sent from its depot to collect the items of orders in different loca-

	100 orders		200 orders		300 orders	
	single-line	multi-line	single-line	multi-line	single-line	multi-line
dedicated	5263	21,069.3	10,090.1	42,249	15,190	57,275.6
mixed(10)	2605	10,633	4589.5	23,266	7804	34,514.8
%	-50	-49.5	-54.5	-45	-48.7	-40
mixed(20)	1775.9	7280	3516.9	15,600	5682.6	24,308
%	-66.3	-65.4	-65.2	-63	-62.6	-57.6

Table 6: The average distances for instance set a, b, c of 3240 shelves solved with the VNS algorithm. The lines beginning with "%" show the percentage improvement in the results of mixed-shelves storage compared with the dedicated storage policy. The time limit is set to 1 hour.

tions in the storage area and goes back to its depot. Each order is collected in a bin, and that is efficient for packing. The tour for a cobot is limited by its load capacity. We consider different depots (a.k.a. loading stations for cobots) while the items for each SKU are stored in different shelves in the storage area (a.k.a. the mixed-shelves storage policy). In such an environment, we formulate the new integrated order batching and routing with three-index and two-commodity network flow formulations to minimize the driving distances for cobots. We experimentally show that the two-commodity network flow formulation works better for our integrated problem by achieving lower gaps with smaller computational time. With mixed-shelves storage and multiple lines of orders, the complexity of the problem increases significantly. Therefore, we propose a novel variable neighborhood search (VNS) algorithm to solve the integrated problem more efficiently. According to our experiments, our VNS can obtain near-optimal results (less than 5% gap) for small instances, which can be solved optimally in the two-commodity model within 4 hours. For those small instances where we can obtain feasible solutions within 4 hours, our VNS algorithm provides a better solution with an improvement of up to 17.7% within 15 minutes, if a comparable feasible integer solution can be obtained from the Gurobi solver within 4 hours. Also, we apply our VNS to solve large instances within an acceptable computational time. Based on our VNS solutions, we can see that mixed-shelves storage works well both for the single-line and multi-line small-sized orders we tested. Up to 67% can be saved on driving distances by applying the mixed-shelves storage policy compared with our reference storage policy.

Since an AGV-assisted warehouse system is a relatively new type of warehousing system, the concepts specific to such a system have not received much scholarly attention. Each picker works in a particular area in the storage area (called a *zone*). For example, the determination of the number of zones (see e.g. De Koster et al. (2012) for the manual system) and the distribution of SKUs in each zone might be interesting topics. Mixed-shelves storage can be used as a good instrument to balance the workload of pickers, since the items of a SKU can be stored in different zones, while it reduces the waiting time for several cobots for a single picker.

References

Alvim, A., Glover, F. S., Ribeiro, C. C., & Aloise, D. J. (1999). Local search for the bin packing problem. In *Extended Abstracts of the 3rd Metaheuristics International Conference* (pp. 7–12). CiteSeer.

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, *53*, 917–945. doi:10.1287/trsc.2018.0873.

Baldacci, R., Hadjiconstantinou, E., & Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, *52*, 723–738. doi:10.1287/opre.1040.0111.

Baldacci, R., Toth, P., & Vigo, D. (2007). Recent advances in vehicle routing exact algorithms. *4OR*, *5*, 269–298. doi:10.1007/s10288-007-0063-3.

Boysen, N., Briskorn, D., & Emde, S. (2017). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, *262*, 550–562. doi:10.1016/j.ejor.2017.03.053.

Boysen, N., de Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, *277*, 396–411. doi:10.1016/j.ejor.2018.08.023.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L., & Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research*, *285*, 497–512. doi:10.1016/j.ejor.2020.01.059.

Chen, T.-L., Cheng, C.-Y., Chen, Y.-Y., & Chan, L.-K. (2015). An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, *159*, 158–167. doi:10.1016/j.ijpe.2014.09.029.

Cheng, C.-Y., Chen, Y.-Y., Chen, T.-L., & Yoo, J. J.-W. (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. *International Journal of Production Economics*, *170*, 805–814. doi:10.1016/j.ijpe.2015.03.021.

Daniels, R. L., Rummel, J. L., & Schantz, R. (1998). A model for warehouse order picking. *European Journal of Operational Research*, *105*, 1–17. doi:10.1016/S0377-2217(97)00043-X.

De Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*, 481–501. doi:10.1016/j.ejor.2006.07.009.

De Koster, R. B., Le-Duc, T., & Zaerpour, N. (2012). Determining the number of zones in a pick-and-sort order picking system. *International Journal of Production Research*, *50*, 757–771. doi:10.1080/00207543.2010.543941.

Ene, S., & Öztürk, N. (2012). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology*, *60*, 787–797. doi:10.1007/s00170-011-3593-y.

Fabric (2020). The impact of Covid-19 on online grocery: Why the coronavirus pandemic in 2020 will change the grocery industry forever in the United States. URL: https://getfabric.com/the-impact-of-covid-19-on-online-grocery/.

Finke, G. (1984). A two-commodity network flow approach to the traveling salesman problem. *Congressus Numerantium*, *41*, 167–178.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness* volume 174. Freeman.

Gendreau, M., & Potvin, J.-Y. E. (2010). *Handbook of metaheuristics* volume 3. Springer. doi:10.1007/978-3-319-91086-4.

Giosa, I., Tansini, I., & Viera, I. (2002). New assignment algorithms for the multi-depot vehicle routing problem. *Journal of the Operational Research Society*, *53*, 977–984. doi:10.1057/palgrave.jors.2601426.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference* (pp. 11–15). Pasadena, CA, USA.

Karmarkar, N., & Karp, R. M. (1982). *The differencing method of set partitioning*. Computer Science Division (EECS), University of California.

Kulak, O., Sahin, Y., & Taner, M. E. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flexible Services and Manufacturing Journal*, *24*, 52–80. doi:10.1007/s10696-011-9101-8.

Kytöjoki, J., Nuortio, T., Bräysy, O., & Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, *34*, 2743–2757. doi:10.1016/j.cor.2005.10.010.

Lee, H.-Y., & Murray, C. C. (2019). Robotics in order picking: Evaluating warehouse layouts for pick, place, and transport vehicle routing systems. *International Journal of Production Research*, *57*, 5821–5841. doi:10.1080/00207543.2018.1552031.

Letchford, A. N., Nasiri, S. D., & Theis, D. O. (2013). Compact formulations of the Steiner traveling salesman problem and related problems. *European Journal of Operational Research*, *228*, 83–92. doi:10.1016/j.ejor.2013.01.044.

Li, J., Huang, R., & Dai, J. B. (2017). Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. *International Journal of Production Research*, *55*, 447–461. doi:10.1080/00207543.2016.1187313.

Lin, C.-C., Kang, J.-R., Hou, C.-C., & Cheng, C.-Y. (2016). Joint order batching and picker Manhattan routing problem. *Computers & Industrial Engineering*, *95*, 164–174. doi:10.1016/j.cie.2016.03.009.

Matusiak, M., de Koster, R., Kroon, L., & Saarinen, J. (2014). A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. *European Journal of Operational Research*, *236*, 968–977. doi:10.1016/j.ejor.2013.06.001.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097–1100. doi:10.1016/S0305-0548(97)00031-2.

Montoya-Torres, J. R., Franco, J. L., Isaza, S. N., Jiménez, H. F., & Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, *79*, 115–129. doi:10.1016/j.cie.2014.10.029.

Ramos, T. R. P., Gomes, M. I., & Póvoa, A. P. B. (2020). Multi-depot vehicle routing problem: A comparative study of alternative formulations. *International Journal of Logistics Research and Applications*, *23*, 103–120. doi:10.1080/13675567.2019.1630374.

Scholz, A., Schubert, D., & Wäscher, G. (2017). Order picking with multiple pickers and due dates: Simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. *European Journal of Operational Research*, *263*, 461–478. doi:10.1016/j.ejor.2017.04.038.

Tompkins, J. A. (2010). *Facilities planning*. (4th ed.). John Wiley & Sons.

Tsai, C.-Y., Liou, J. J., & Huang, T.-M. (2008). Using a multiple-GA method to solve the batch picking problem: Considering travel distance and order due time. *International Journal of Production Research*, *46*, 6533–6555. doi:10.1080/00207540701441947.

Valle, C. A., Beasley, J. E., & da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, *262*, 817–834. doi:10.1016/j.ejor.2017.03.069.

Van Gils, T., Caris, A., Ramaekers, K., & Braekers, K. (2019). Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research*, *277*, 814–830. doi:10.1016/j.ejor.2019.03.012.

Van Gils, T., Ramaekers, K., Caris, A., & De Koster, R. B. (2018). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, *267*, 1–15. doi:10.1016/j.ejor.2017.09.002.

Weidinger, F. (2018). Picker routing in rectangular mixed shelves warehouses. *Computers & Operations Research*, *95*, 139–150. doi:10.1016/j.cor.2018.03.012.

Weidinger, F., Boysen, N., & Schneider, M. (2019). Picker routing in the mixed-shelves warehouses of e-commerce retailers. *European Journal of Operational Research*, *274*, 501–515. doi:10.1016/j.ejor.2018.10.021.

Won, J., & Olafsson, S. (2005). Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, *43*, 1427–1442. doi:10.1080/00207540410001733896.

Wulfraat, M. (2016). Locus Robotics: An independent consultant review of autonomous robots. URL: https://mwpvl.com/html/locus_robotics_-_independent_consultant_review.html.

Xie, L., Thieme, N., Krenzler, R., & Li, H. (2020). Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems. *European Journal of Operational Research*, . doi:https://doi.org/10.1016/j.ejor.2020.05.032.

Appendix A. Components of an AVG-assisted picking system

Firstly, we define some terms related to orders in Table A.7.

Term	**Description**
SKU	stock keeping unit
order line	one SKU with the ordered quantity
item	a physical unit of one SKU
pick order	a set of order lines from a customer's order
single-line order	single SKU per order
multi-line order	multiple SKUs per order
backlog	all unfulfilled orders

Table A.7: Terms related to orders.

The central components of an AGV-assisted picking system are listed in Table A.8.

Component	**Description**
shelves	storing items
bin	temporary storage items for an order during picking
storage area	the inventory area where the shelves are stored
human workers	
picker	working in a given part of the storage area to pick items from shelves
packer	packing order items in packing stations
packing stations	where packers pack the pick order items
queue	a list of cobots waiting in front of a packing station
picking locations	somewhere in front of the shelves in the storage area where pickers pick the order items
cobots	robots that transport items between the storage area and workstations. Each of them has a limited weight capacity and belongs to a certain packing station.

Table A.8: The central components of an AGV-assisted picking system.