Efficient move evaluation and neighborhood exploration for integrated order picking problems

Thibault Prunet, Nabil Absi, Diego Cattaruzza

Abstract

Order Picking (OP) is widely considered as the most resource-intensive activity in warehousing logistics. The optimization of OP planning problems has attracted substantial attention from the literature, and recent studies highlighted the importance of integrating several levels of decisions for OP performances. While the mathematical programming literature on the topic has seen important advances in the recent years by exploiting *problem-specific knowledge* on the warehouse structure, the metaheuristic literature does not fully exploit this structure to propose efficient neighborhoods. In this paper, we contribute at addressing this research gap by introducing generic move evaluation and neighborhood exploration routines for a broad class of integrated OP problems. First, we propose a 2-step constructive heuristic for the Picker Routing Problem (PRP), coined the *Aisle First Cross Second* (AFCS) heuristic. The AFCS heuristic runs in linear time of the size of the instance, and provides upper and lower bounds for the problem with a proven approximation ratio. Then, we introduce a neighborhood exploration scheme that uses the bounds returned by the AFCS as surrogate objective functions to prune the search. To highlight the potential of these methodological contributions, we develop a generic Large Neighborhood Search (LNS) algorithm that is tested on benchmark instances of two important problems of the OP literature with different structures: the *Joint Order Batching and Picker Routing Problem* (JOBPRP) and the *Storage Location Assignment and Picker Routing Problem* (SLAPRP). Numerical experiments show that the LNS matches state-of-the-art methods from the literature for both the JOBPRP and the SLAPRP.

Keywords: Neighborhood, Order Picking, Picker Routing, Storage, Order Batching, Metaheuristic, Large Neighborhood Search.

1 Introduction

Within warehousing activities, Order Picking (OP) is widely considered as the most resource-intensive. In manual picker-to-parts warehouses, where human operators physically move through the facility to fulfill orders, the OP activity alone accounts for 50-75% of the total operating costs (Frazelle 2016). While efficient solution methods have been developed for single OP planning problems, recent researches have highlighted the potential

gains from integrating several levels of decision (van Gils et al. 2018). In this paper, we propose methodological contributions that apply to a broad class of OP problems, refered as *integrated OP problems*, defined as any deterministic planing problem whose objective is to minimize a function that is cumulative on the picking aisle traversals (e.g., traveled distance, picking time). To this end, we study the *Picker Routing Problem* (PRP), which involves finding a minimum length tour within a warehouse, as the main building block to evaluate the cost of solutions for integrated problems. We apply our methodology to two important problems with distinct structures:

- The *Joint Order Batching and Picker Routing Problem* (JOBPRP) which focuses on consolidating customer orders into batches retrieved via a single route.
- The *Storage Location Assignment and Picker Routing Problem* (SLAPRP) which aims at deciding the locations of the different Stock Keeping Units (SKU) within the warehouse while also determining the picking routes to retrieve orders.

The OP literature uses two different modeling paradigms to design efficient algorithms. The most straightforward option, commonly found in the literature, models a picking route as a tour between visited locations. In this paradigm, the elementary modeling unit is the visit to a given *location*, and its position within a route. The advantage of this approach lies in the prolific literature on routing problems, which academics can rely on. The second option involves modeling a picking route as a succession of aisle traversals, thus exploiting *problem-specific knowledge* on the warehouse layout structure. The seminal work of Ratliff and Rosenthal (1983) is the first to use the *aisle* as the elementary modeling unit, disregarding the visit sequence in a route. They introduced a polynomial-time *Dynamic Programming* (DP) algorithm for the PRP in a single-block warehouse. It is important to emphasize that the two paradigms lead to very distinct models and algorithms. Nowadays, most state-of-the-art mathematical-programming-based methods use aisle modeling. For instance Pansart et al. (2018); Schiffer et al. (2022) applied this approach to the PRP, while Briant et al. (2020); Wahlen and Gschwind (2023) use it for the JOBPRP. In terms of metaheuristics, however, most studies do not fully exploit problem-specific knowledge. More precisely, these studies can be classified into three main trends:

- Using location modeling. This is particularly relevant for the JOBPRP, as it can be modeled as a clustered VRP (Aerts et al. 2021). However, this approach overlooks the structure of the problem.
- Using simplified routing policies. In this case the routing problem is not solved to optimality, but with simple *routing policies* (Menéndez et al. 2017; Žulj et al. 2018). While allowing efficient neighborhood implementation, this approach does not permit to fully optimize the routing.
- Recomputing the optimal route to evaluate each move (Silva et al. 2020; D'Haen et al. 2023). Most of the time this is done with location modeling, where the PRP is solved using generic TSP solvers. An improved approach would leverage problem-specific knowledge in the route computation, such as using the DP algorithm by

(Ratliff and Rosenthal 1983; Pansart et al. 2018). The advantage of this approach is that the algorithm always works with TSP-optimal routes, which is not the case when using neighborhoods based on location modeling. However, the routing literature highlighted that fast move evaluation is a critical aspect of performance improvement (Prodhon and Prins 2016). While the DP algorithm of (Ratliff and Rosenthal 1983; Pansart et al. 2018) is very efficient, solving optimally a strongly NP-hard problem to evaluate each move might leave room for further improvement.

To the best of the authors' knowledge, there exists no study aiming at designing efficient move evaluation using problem-specific knowledge for integrated OP problems. Since neighborhoods can become quite complex for OP problems (e.g., several position inserted in a route, or several modified routes), the efficient computation of insertion costs is far from being a computationally easy task. In this paper, we aim at addressing this gap by introducing an efficient neighborhood exploration scheme that relies on novel move evaluation surrogates. To this end, we propose a constructive heuristic for the PRP, coined the *Aisle First Cross Second* (AFCS) heuristic, providing upper and lower bounds on the route cost. The value provided by this heuristic serves as a surrogate objective function for move evaluation, a common technique in the scheduling literature when the objective function is costly to evaluate (Crainic et al. 1993). This neighborhood exploration scheme is embedded into a generic Large Neighborhood Search (LNS) algorithm that presents convincing results on benchmark instances from the literature on both the JOBPRP and SLAPRP. The contributions of the paper are as follow:

1. We introduce a 2-stage constructive heuristic that provides upper and lower bounds for the PRP, along with computing the time-complexity and an approximation ratio.
2. We propose a second heuristic based on the DP algorithm of Ratliff and Rosenthal (1983) to evaluate moves. This heuristic is highly efficient in computational time, but is not guaranteed to return a solution.
3. We propose a novel move evaluation scheme based on a surrogate objective function for integrated OP problems. A neighborhood exploration strategy is developed to efficiently prune dominated parts of the search space.
4. We develop a generic Large Neighborhood Search (LNS) algorithm for the JOBPRP and SLAPRP.
5. We release the open-source package `PickerRouting.jl`[1] that provide a fast implementation and comprehensive interface to the exact and heuristic route evaluation routines.
6. We conduct extensive computational experiments on the performance of the LNS algorithm on benchmark instances from the literature on the JOBPRP and SLAPRP.

[1] `https://github.com/prunett/PickerRouting.jl` (under development)

The reminder of this paper is organized as follows. Section 2 reviews the relevant literature. Section 3 defines the problems. Section 4 introduces the aisle first cross second heuristic for the PRP, as well as upper and lower bounds for the problem. Section 5 proposes the critical heuristic for move evaluation and an efficient neighborhood exploration scheme. Section 6 presents the LNS algorithm. Section 7 details numerical experiments. Finally, Section 8 concludes the work.

2 Related literature

The literature on Order Picking is extensive, and providing an exhaustive review is out of scope of this study. Interested readers are referred to de Koster et al. (2007); van Gils et al. (2018); Boysen et al. (2019); Masae et al. (2020). In this section we exclusively present studies directly related to the present work.

The PRP. The seminal work of Ratliff and Rosenthal (1983) first studied the PRP in a single-block warehouse, and introduced an aisle-based DP model whose underlying idea remains the cornerstone of many modern approaches. Their linear-time DP algorithm has then been extended to two-block warehouses (Roodbergen and de Koster 2001b), and to the general case by Pansart et al. (2018) who propose a fixed-parameter tractable algorithm. Despite the efficiency of DP based approaches, even for industrial-sized instances with side-constraints (Schiffer et al. 2022), many authors and real-life warehouses still rely on simple rule-based heuristics coined *routing policies*. Examples of such routing policies include S-shape, return, midpoint, largest gap or combined, detailed in Roodbergen and de Koster (2001a). Some scholars advocate for heuristic routing policies due the argument that optimal routes can be confusing and error-prone for pickers in practical situations (Petersen and Schmenner 1999), although this claim has been challenged by behavioral factor studies (Elbert et al. 2017). While DP-based exact algorithms solve very efficiently the problem, the PRP has recently been proven NP-hard in the strong sense (Prunet et al. 2025), providing a rationale for the use of heuristics, mainly for time-critical applications such as move evaluations in metaheuristic frameworks.

Integrated OP problems. Recent review studies on OP advocate for the importance of integrated problem solving for operational performance improvements (van Gils et al. 2018; Boysen et al. 2019). This stream of research has experienced high activy in recent years, leading to the emergence of novel problems to deal with the challenges faced by modern e-commerce warehouses, such as the JOBPRP and SLAPRP whose literature are analyzed below. Other interesting applications include the PRP with scattered storage (Weidinger 2018), where SKUs are stored in multiple locations so that the choice of location is optimized alongside routing. (van Gils et al. 2019; Briant et al. 2023) consider order due dates in the JOBPRP, optimizing sequencing decisions while processing batches. Another trend integrates OP problems with vehicle routing decisions to improve the overall supply chain efficiency: Moons et al. (2019) jointly optimize vehicle and picker routing, and D'Haen et al. (2023) further extends the integration to batching.

The JOBPRP. Order batching is an important leverage for OP performance that is implemented in real-life warehouses for decades (Boysen et al. 2019), generating extensive studies in the literature. Most early studies focused on constructive heuristics inspired by the vehicle routing literature, such as seed or savings algorithms, and routing policies for performing the routing (de Koster et al. 1999). Recent works tend to tackle the problem with metaheuristics, but still use heuristic routing policies instead of optimal routing to evaluate the picking distance in most cases (Henn and Wäscher 2012; Menéndez et al. 2017). In the context of JOBPRP, the picking distance within each route is optimized jointly with batching. The JOBPRP has been first investigated by Won and Olafsson (2005) with a saving-based heuristic for batching, and a 2-opt heuristic for routing. The problem has mainly been tackled by metaheuristics such as Tabu search (Kulak et al. 2012), particle swarm optimization (Cheng et al. 2015), iterated local search (Scholz and Wäscher 2017) and Variable Neighborhood Search (VNS) (Aerts et al. 2021). Mathematical programming-based methods gained popularity since the work of Valle et al. (2017) that first proposed an exact method. Briant et al. (2020) and Wahlen and Gschwind (2023) propose efficient heuristic Branch-Cut-and-Price (BPC) algorithms that leverage the problem structure. To the best of the authors knowledge, Wahlen and Gschwind (2023) currently stands as the state-of-the-art solution method for the JOBPRP.

The SLAPRP. The SLAPRP is strongly NP-hard, even within a single-aisle warehouse context (Boysen and Stephan 2013), and is especially challenging due to its high combinatorics (Prunet et al. 2024). The integration of storage and routing decisions has received a limited attention from the research community, with even fewer works studying it with an exact optimization of the picking distance. In terms of exact methods, Boysen and Stephan (2013) propose a DP-based algorithm for a single-aisle warehouse, while Silva et al. (2020) introduce a cubic formulation with very limited computational efficiency, and Prunet et al. (2024) propose both a compact formulation and a Branch-Cut-and-Price algorithm. To the best of the authors' knowledge, Silva et al. (2020) is the only work proposing a metaheuristic approach to solve the problem with optimal routing. Further contributions on the topic include Mantel et al. (2007) and Guo et al. (2021) who solve the routing using heuristic policies.

3 Mathematical description of the PRP and integrated problems

In this section we first define the layout graph representing the warehouse, then we introduce the problems considered in this work, namely the PRP, the JOBPRP and the SLAPRP. The mathematical notation used throughout this paper are summarized in Table 1.

Layout Graph. The studied problems are defined on a conventional rectangular multi-block layout, as illustrated in Figure 1. The layout is composed of a set $\mathcal{K} = \{1, \dots, K\}$

Table 1: Mathematical notations

Sets	
$\mathcal{K}$	Set of blocks
$\mathcal{A}^k$	Set of aisles of block $k \in \mathcal{K}$
$\mathcal{A}$	Set of all aisles
$\mathcal{L}^{ki}$	Set of locations in block $k \in \mathcal{K}$, aisle $i \in \mathcal{A}$
$\mathcal{L}$	Set of all locations
$\mathcal{L}^0 = \mathcal{L} \cup \{v^0\}$	Set of all locations including the depot
$\mathcal{G} = (\mathcal{L}^0, \mathcal{E})$	Layout graph
$\mathcal{E}$	Set of edges of the layout graph
$\mathcal{P}$	Set of intersections between the picking aisles and cross aisles
$\mathcal{O}$	Set of orders
Λ	Set of SKUs
Λ^o	Set of SKUs of order $o \in \mathcal{O}$
$\mathcal{S}$	Set of solutions
$\mathcal{R}$	Set of routes
$\mathcal{R}(s)$	Set of routes in solution $s \in \mathcal{S}$
$\mathcal{N}(s)$	Set of neighbors of $s \in \mathcal{S}$
$\mathcal{V}$	Set of locations to visit for a route
$\mathcal{A}^t(s)$	Set of aisles traversed by traversal $t \in \{top, bottom, gap, 1-pass\}$ in partial solution s
$\mathcal{P}(s)$	Subset of intersections visited in partial solution s
$\mathcal{H} = (\mathcal{P}(s), \Sigma)$	Intersection graph
$\widetilde{\mathcal{H}}$	Contracted intersection graph
Σ	Set of edges of the intersection graph
$\widetilde{\Sigma}$	Subset of edges corresponding to `1-pass` and `1-cross`
Parameters	
K	Number of blocks
A	Number of aisles in one block
L	Number of locations in one aisle
D^{loc}	Distance between two consecutive locations
D^{aisle}	Distance between two consecutive aisles
D^{first}	Distance between the first location of an aisle and the cross aisle
v^0	The depot
q_o	Capacity used by order $o \in \mathcal{O}$
Q	Capacity of a picker
c^t_{ki}	Cost of visiting all locations $\mathcal{V} \cap \mathcal{L}^{ki}$ with traversal $t \in \{top, bottom, gap, 1-pass\}$
k^{max}	Farthest visited block in a route
a^{max}	Farthest visited aisle in a route

of blocks, each divided into A parallel aisles. We note a_{ki} the i^{th} aisle of block $k \in \mathcal{K}$, $\mathcal{A}^k = \{a_{ki},\ 1 \leq i \leq A\}$ the set of aisles in block $k \in \mathcal{K}$, and $\mathcal{A} = \bigcup_{k \in \mathcal{K}} \mathcal{A}^k$ the set of all aisles. Blocks are numbered sequentially from bottom to top, while aisles from left to right. We assume that pickers can pick indifferently from the right and left shelves when crossing an aisle. Therefore, we can aggregate the storage spaces that are equivalent in terms of distances (i.e., the ones facing each other), and call *location* such an aggregation. In this case, each aisle $a_{ki} \in \mathcal{A}$ comprises a set $\mathcal{L}^{ki}$ of L locations and we note $\mathcal{L} = \bigcup_{a_{ki} \in \mathcal{A}} \mathcal{L}^{ki}$ the set of all locations. We note D^{loc} the distance between two consecutive locations, D^{aisle} the distance between two consecutive aisles, and D^{first} the distance between the first location in an aisle and the nearest cross aisle. Additionally, we assume that the picker starts and ends its tour at the same location v^0 identified as the *depot*. The depot is located in the first cross aisle, in front of aisle $a_{1depot} \in \mathcal{A}^1$, and separated by a distance D^{depot} from the the start of aisle a_{1depot}. Let $\mathcal{L}^0 = \mathcal{L} \cup \{v^0\}$ represent the set of locations including the depot. The warehouse is then represented by the complete undirected graph $\mathcal{G} = (\mathcal{L}^0, \mathcal{E})$, called the *layout graph*. Each edge $e = (v^1, v^2) \in \mathcal{E}$ is associated with a cost c_e corresponding to the shortest walking distance between locations v^1 and v^2. The red diamonds in Figure 1 represent the *intersections* between picking aisles and cross aisles. The intersection between cross aisle $1 \leq k \leq K+1$ and aisle $1 \leq i \leq A$ is noted p_i^k. Let $\mathcal{P} = \{p_i^k,\ 1 \leq k \leq K+1,\ 1 \leq i \leq A\}$ be the set of all intersections between picking aisles and cross aisles.

The picker routing problem. In the PRP, a subset $\mathcal{V} \subset \mathcal{L}^0$ of locations, including the depot $v^0 \in \mathcal{V}$, needs to be visited. The problem is then defined on the subgraph of $\mathcal{G}$ induced by $\mathcal{V}$, and aims at finding a minimum-cost tour that visits all vertices exactly once. We refer to a PRP solution as a *route*.

The Joint Order Batching and Picker Routing Problem. In the JOBPRP, batching decisions are jointly optimized with picker routing decisions. Let Λ be the set of SKUs present in the warehouse. We are given a set $\mathcal{O}$ of orders, each needs that a subset $\Lambda^o \subset \Lambda$ of SKUs to be picked together. Each order $o \in \mathcal{O}$ is associated with a quantity $q_o > 0$, generally $q_o = |\Lambda^o|$. The goal of the JOBPRP is to assign the orders to a set of batches $\mathcal{R}$ of a limited capacity Q, where SKUs of a given batch are picked together by a single route. The objective is to minimize the total traveled distance.

The Storage Location Assignment and Picker Routing Problem. In the SLAPRP, routing decisions are jointly optimized with storage decisions. We are given a set Λ of SKUs and a set $\mathcal{O}$ of orders. Each order $o \in \mathcal{O}$ is retrieved by a distinct route that needs to pick a subset $\Lambda^o \subset \Lambda$ of SKUs. The objective of the SLAPRP is to assign the set of SKUs Λ to the storage locations Λ such that the total traveled distance required to pick the orders is minimized.

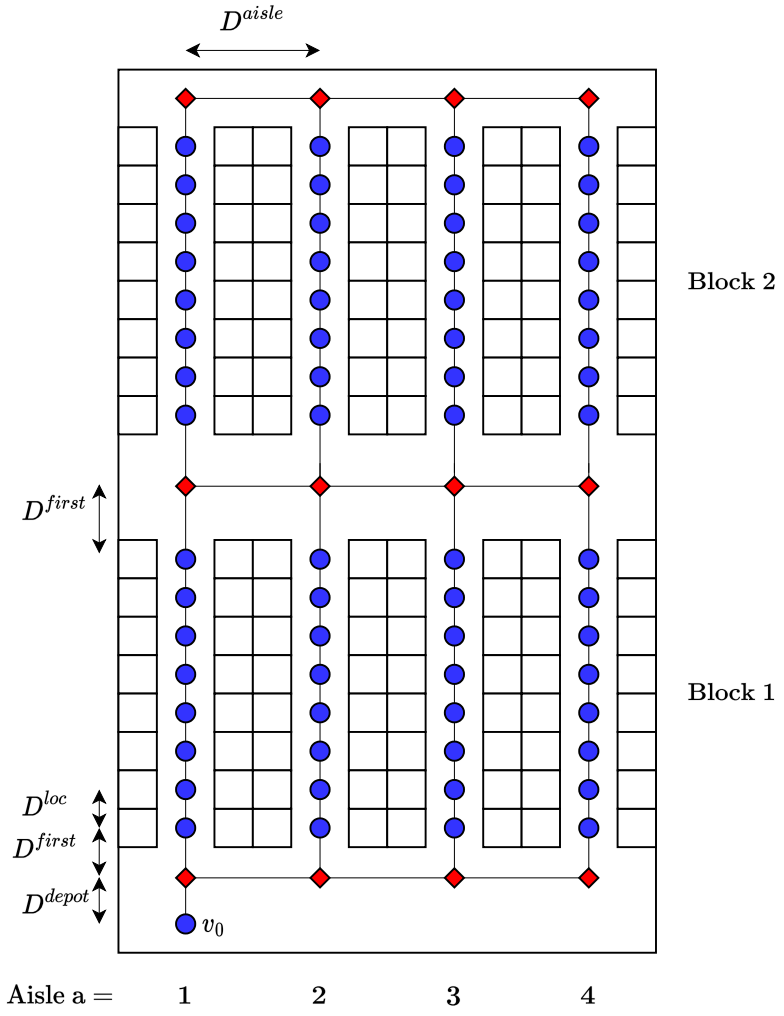


Figure 1: Two-blocks layout. Blue dots represent locations, and red diamonds represent dummy points corresponding to the intersection between aisles and cross aisles.

4 Aisle First Cross Second heuristic

In this section we introduce a constructive heuristic for the PRP, coined the *Aisle First Cross Second* (AFCS) heuristic, and present related methodological results. We first provide an illustrative example in Section 4.1. We formally describe the AFCS heuristic in Section 4.2 and compute its time complexity in Section 4.3. A lower bound for the PRP is derived from the first phase of the AFCS heuristic in Section 4.4, and an approximation ratio is computed in Section 4.5.

4.1 Comprehensive description and illustrative example

Preliminary notions on traversals. First, let us introduce some preliminary definitions and notations. As shown by Ratliff and Rosenthal (1983), there are only six possible

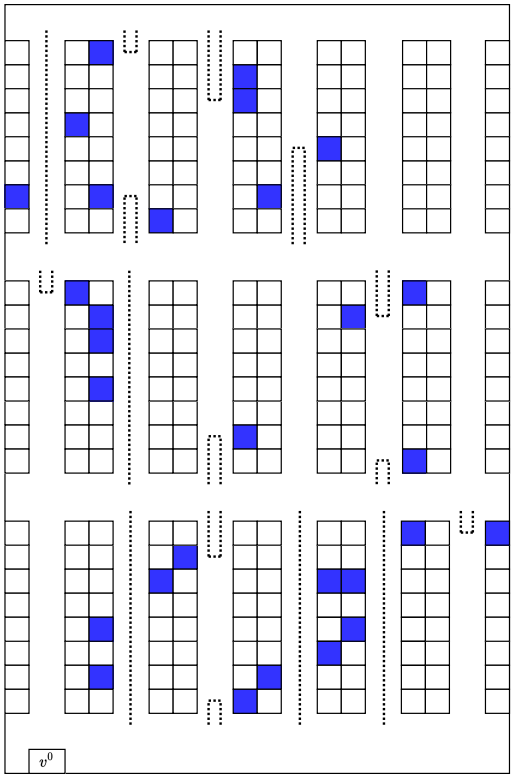
ways to traverse an aisle in an optimal solution: `1-pass`, `2-pass`, `top`, `bottom`, `gap` and `void`. The `1-pass` is a complete traversal of the aisle (e.g., block 1 aisle 2 in Figure 2.a), while `2-pass` represents 2 full traversals. The `top` is a partial traversal entering from the top and returning (e.g., block 1 aisle 6 in Figure 2.a), and `bottom` a partial traversal entering from the bottom and returning (e.g., block 2 aisle 3 in Figure 2.a). `gap` is a partial traversal where the aisle is entered both from the top and bottom (e.g., block 1 aisle 3 in Figure 2.a), while `void` refers to an unvisited aisle (e.g., block 1 aisle 1 in Figure 2.a). These are called *aisle traversals* and correspond to vertical paths. In this section, we disregard `2-pass`, considering that this possibility is substituted by two `1-pass` traversals. Similarly, a cross aisle between two consecutive aisles can be crossed either 0, 1 and 2 times. We refer to `1-cross` (resp. `2-cross`) for cross aisles portions traversed once (resp. twice). These are termed *cross traversals* and correspond to horizontal paths. Finally, for an intersection $p \in \mathcal{P}$ the number of paths connected to p is called the *degree* of p.

Descriptive example. Before providing a formal description of the AFCS heuristic, let us introduce it through a comprehensive "pen and paper" example illustrated in Figure 2. The AFCS is a constructive heuristic that works in two steps:

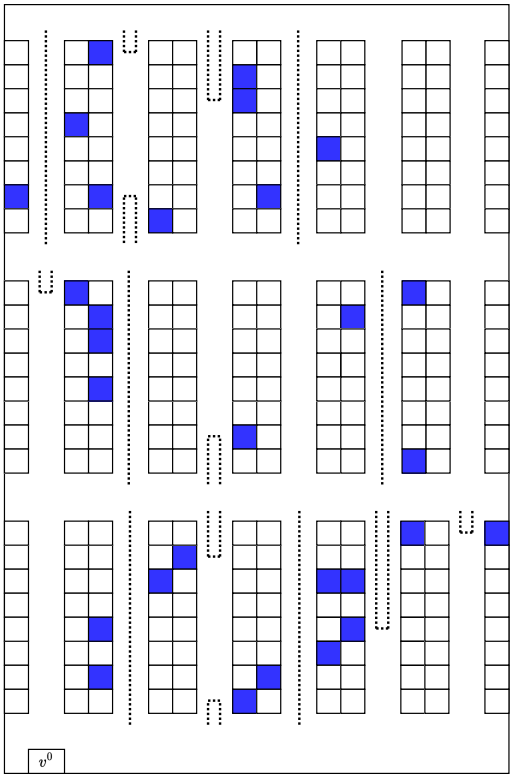
- First, the AFCS builds the aisle traversals, as depicted in Figure 2.a and Figure 2.b. This is achieved by first computing the different traversal costs (i.e., `top`, `bottom`, `gap` and `1-pass`) for each aisle. Each aisle is considered independently and its least-cost traversal, i.e., the lowest cost needed to retrieve the SKUs in the aisle, is determined as shown in Figure 2.a. Then, some aisle traversals are modified to ensure the future feasibility of the route, in particular aiming for an even number of `1-pass` in each block. Otherwise it will not be possible to build a valid tour, as the picker would not be able to return to the depot. This adjustment is made optimally, that is with the minimum cost increment. After this stage, as illustrated in Figure 2.b, the aisle traversals are fixed and provide a lower bound on the total cost (see Section 4.4).
- Second, the AFCS builds the cross traversals, as illustrated by Figure 2.c and Figure 2.d. This phase starts by adding the `1-cross` traversals between consecutive `1-pass` in each block. After this step, the solution is composed of partial subtours, with each intersection point having an even degree, as illustrated in Figure 2.c. Subsequently, the procedure connects all partial subtours by `2-cross` traversals to create a valid tour as a solution, as depicted in Figure 2.d. This operation is optimally done, that is with the minimum cost increment.

4.2 Formal description of the AFCS heuristic

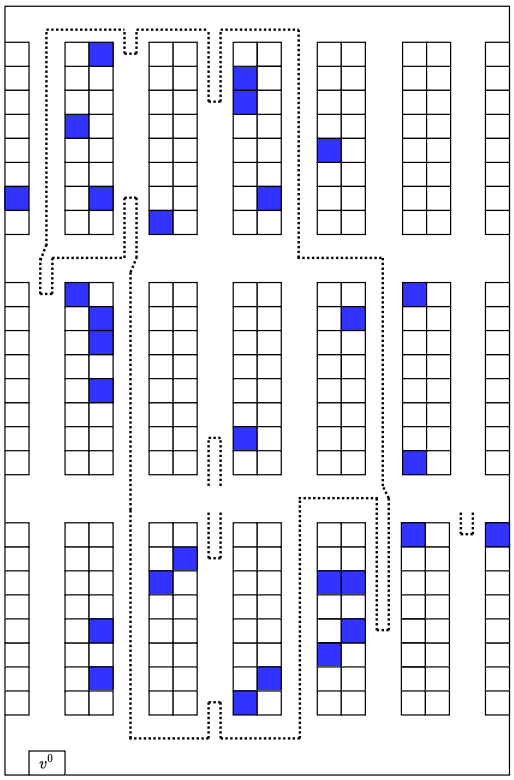
Let us consider an instance of the PRP induced by the set $\mathcal{V}$. For an aisle $a_{ki} \in \mathcal{A}$, we note c^t_{ki} the cost to visit all locations $\mathcal{V} \cap \mathcal{L}^{ki}$ of a_{ki} when using traversal $t \in \{top, bottom, gap, 1-pass\}$. Since the cost of `1-pass` traversals is constant, we note



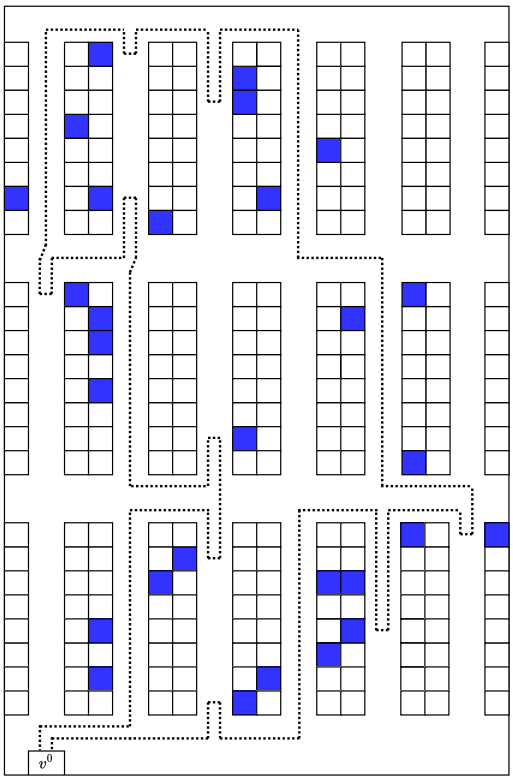
(a) Compute best aisle traversals



(b) Ensure vertical feasibility



(c) Connect `1-pass` traversals



(d) Connect the isolated partial subtours

Figure 2: Illustrative example of the AFCS heuristic

$c^{1pass} = (L-1)D^{loc} + 2D^{first}$. Additionally, we note k^{max} the farthest visited block (i.e., $\mathcal{V} \cap \mathcal{L}^{k^{max}i} \neq \emptyset$ and $\mathcal{V} \cap \mathcal{L}^{ki} = \emptyset$ for all $k^{max} < k \leq K$, and $1 \leq i \leq A$).

The AFCS heuristic builds a valid PRP solution in two steps: first, building the aisle traversals, and second, the cross traversals. These procedures are detailed in the following.

Step 1: Constructing the aisle traversals. This procedure determines the traversal costs for each aisle and identifies the aisles traversed by `1-pass`. The other aisles are traversed by the minimum cost traversal among `top`, `bottom` and `gap`. This is performed according to Algorithm 1.

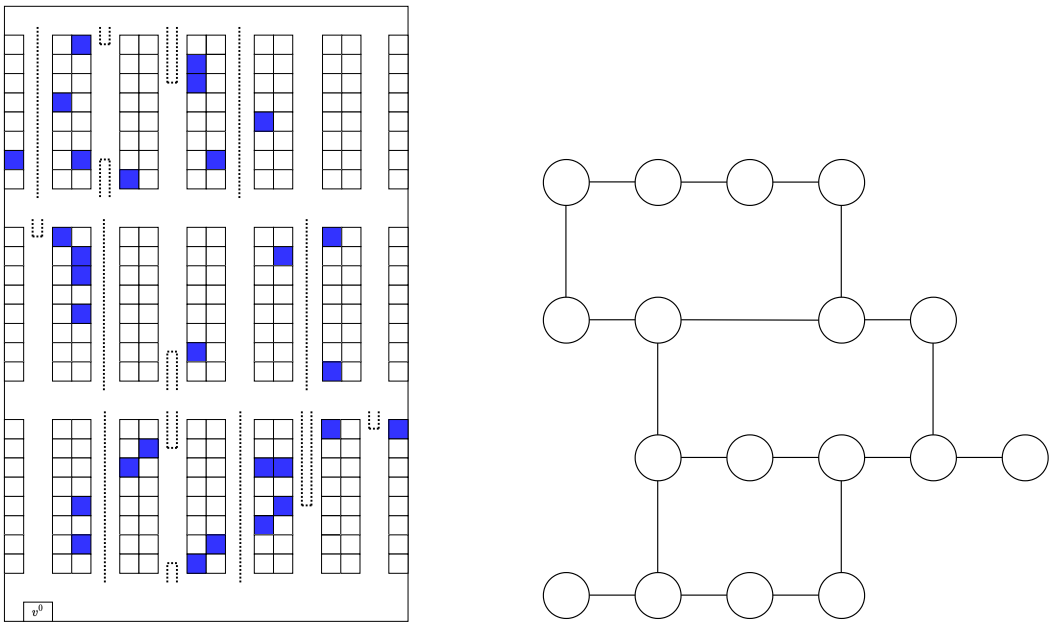
Algorithm 1: Step 1 of the AFCS heuristic

Input: $\mathcal{V}$

1: Compute c_{ki}^{top}, c_{ki}^{bottom} and c_{ki}^{gap} for all $1 \leq k \leq K$ and $1 \leq i \leq A$
2: **for** $1 \leq k \leq k^{max}$ **do**
3: | **for** $1 \leq i \leq A$ **do**
4: | | $c_{ki} \leftarrow min(\{c_{ki}^{top}, c_{ki}^{bottom}, c_{ki}^{gap}\})$
5: | $\sigma^k \leftarrow$ sorted permutation of $\mathcal{A}^k$ by non-increasing order of c_{ki}^{nopass}
6: | $c_{k\sigma^k(1)} \leftarrow c^{1pass}$
7: | $c_{k\sigma^k(2)} \leftarrow c^{1pass}$
8: | **if** $(k = k^{max}) \wedge (2c^{pass} + \sum_{3 \leq i \leq A} c_{k\sigma^k(i)} > \sum_{1 \leq i \leq A} c_{ki}^{bottom})$ **then**
9: | | $c_{ki} \leftarrow c_{ki}^{bottom}$ for all $1 \leq i \leq A$
10: | $i \leftarrow 3$
11: | **while** $2c^{1pass} \leq c_{k\sigma^k(i)} + c_{k\sigma^k(i+1)}$ **do**
12: | | $c_{k\sigma^k(i)} \leftarrow c^{1pass}$
13: | | $c_{k\sigma^k(i+1)} \leftarrow c^{1pass}$
14: | | $i \leftarrow i + 2$
15: Return c

First, the procedure computes the traversal costs for each aisle (line 1). For each block $k \in \mathcal{K}$, all aisles are at the beginning traversed by the cheapest traversal among `top`, `bottom` and `gap`, referred to as the *no-pass traversal* (lines 3-4). Subsequently, the aisles are sorted in the non-increasing order of no-pass costs (line 5). Then, the two most expensive no-cost aisles are transformed into `1-pass` (lines 6-7), since there should be at least one `1-pass` traversal to reach subsequent blocks. For the last block, it is not mandatory to add any `1-pass`, in this case all the aisles of the block must be traversed by `bottom` traversal, the least cost option between the two is determined in lines 8-9. In lines 10-14, no-pass traversals are transformed into `1-pass` in pairs, if and only if this reduces the overall traversal cost of the block. Finally, all the traversal costs (and the chosen traversal types) are returned in line 14.

Step 2: Preliminary notions. This procedure constructs the cross traversals required to build a feasible tour. First, let us introduce some definitions. We denote s the partial



(a) Partial subtours after step 1 (b) Corresponding intersection graph

Figure 3: Intersection graph

solution, $\mathcal{A}^{1pass}(s)$ (resp. $\mathcal{A}^{top}(s)$, $\mathcal{A}^{bottom}(s)$, $\mathcal{A}^{gap}(s)$) the set of aisles traversed by `1-pass` (resp. `top`, `bottom`, `gap`) as determined in Step 1. Let $\mathcal{P}(s) \subset \mathcal{P}$ be the subset of intersections visited by the aisle traversals determined in Step 1, along with the one in front of the depot. In other words, $\mathcal{P}(s) = \{p_i^k, \forall a_{ki} \in \mathcal{A}^{1pass}(s) \cup \mathcal{A}^{gap} \cup \mathcal{A}^{bottom}(s)\} \cup \{p_i^{k+1}, \forall a_{ki} \in \mathcal{A}^{1pass}(s) \cup \mathcal{A}^{gap}(s) \cup \mathcal{A}^{top}(s)\} \cup \{p_{i^{depot}}^k\}$.

We define the *intersection graph* as the undirected graph $\mathcal{H} = (\mathcal{P}(s), \Sigma)$ where the set of edges is composed of vertical edges corresponding to `1-pass`, and horizontal edges corresponding to cross traversals between consecutive intersections, i.e. $\Sigma = \{(p_i^k, p_i^{k+1}), \forall a_{ki} \in \mathcal{A}^{1pass}(s)\} \cup \{(p_i^k, p_j^k) \in \mathcal{P}(s), \forall 1 \leq k \leq K+1, 1 \leq i < j \leq A \ : \ p_u^k \notin \mathcal{P}(s), \forall i < u < j\}$. The subsequent steps of the heuristic consists in adding edges from Σ to the route until it becomes feasible. Figure 3 provides an illustration.

Step 2: Constructing `1-cross` traversals. First we choose the edges in Σ that will correspond to `1-cross` traversals (i.e., single horizontal traversals). For each cross aisle $1 \leq k \leq K+1$, this is achieved by counting the number of `1-pass` traversals, both above and below cross aisle k. An edge $(p_i^k, p_j^k) \in \Sigma$ is traversed by a `1-cross` if, and only if, there is an odd number of `1-pass` aisles connected to cross aisle k between the left border of the warehouse and p_i^k. More formally, (p_i^k, p_j^k) is traversed by `1-cross` if $|\{a_{k'i'}, \forall k' \in \{k-1, k\}, 1 \leq i' \leq i\} \cap \mathcal{A}^{1pass}(s)| \equiv 1 \, (mod\ 2)$. We call $\widetilde{\Sigma}$ the set of edges corresponding to `1-pass` and `1-cross`, as illustrated in Figure 4. Note that all vertices of the graph $(\mathcal{P}(s), \widetilde{\Sigma})$ have an even degree, and the partial route is now composed of several cycles, disconnected from each other.

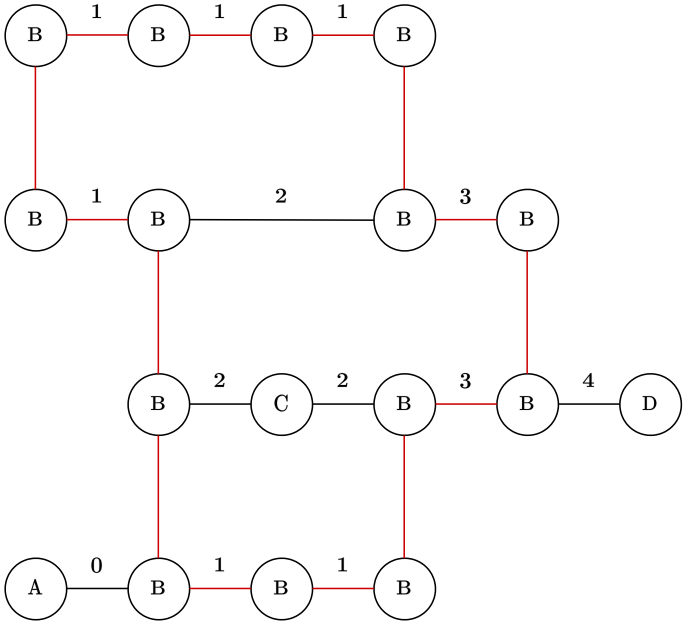


Figure 4: Intersection graph with edges from Σ in red. The horizontal red edges correspond to `1-cross` traversals, and the numbers to the amount of `1-pass` at the left of each edge. The labels of the vertices correspond to the components after contraction.

Step 2: Constructing `2-cross` traversals. The objective now is to connect the isolated partial subtours using `2-cross` traversals, while minimizing the incurred cost. If we contract $\mathcal{H}$ over all the edges in $\widetilde{\Sigma}$, as illustrated in Figure 5, we obtain a graph $\widetilde{\mathcal{H}}$ where each vertex corresponds to an isolated subtour component. The only way to connect these components is by adding `2-cross` traversals between them (i.e., choosing edges of the contracted graph) to ensure they are all connected. Hence, this problem can be modeled as a Minimum Spanning Tree (MST) Problem on $\widetilde{\mathcal{H}}$. We add to the route the `2-cross` corresponding to the edges of the MST to obtain a feasible solution.

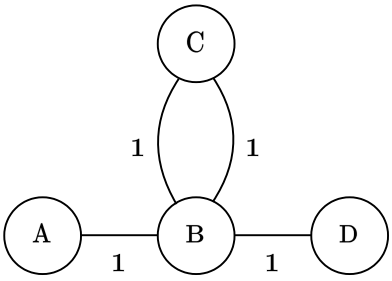


Figure 5: Intersection graph contracted over $\widetilde{\Sigma}$

4.3 Complexity of the AFCS heuristic

Let us introduce Theorem 1 on the complexity of the AFCS heuristic.

Theorem 1. *The AFCS heuristic runs in $\mathcal{O}(|\mathcal{V}| + KA)$ time complexity.*

We emphasize that this result is not straightforward, as there are several operations in the algorithm that would require an increased time complexity in a naive implementation. Specifically, the sorting operation in Algorithm 1 would usually run in $\mathcal{O}(Alog(A))$ for each iteration, summing up to $\mathcal{O}(KAlog(A))$ in total. In addition, the resolution of the MST that would classically require a $\mathcal{O}(KAlog(KA))$ time complexity if implemented with algorithm like Prim's algorithm using binary heaps.

Proof. The complexity of each steps is computed as follows. The first operation of Step 1 consists in computing the `top`, `bottom`, and `gap` traversal costs of each aisle, achievable in $\mathcal{O}(|\mathcal{V}| + KA)$ time complexity according to Heßler and Irnich (2022).

For the reminding complexity of *Step 1*, we refer to Algorithm 1. The procedure browses through the blocks and, except for the sorting component (line 5), everything is either computed in $\mathcal{O}(1)$ time (lines 6-9) or A time (lines 3-4 and 10-14). With this implementation, the sorting operation (line 5) would run in $\mathcal{O}(Alog(A))$ time. However, the loop lines (10-14) show that the first iterations set the aisles to `1-pass` because it is cheaper, and then do not modify them. This mechanism does not actually need all aisles to be sorted, it is sufficient to identify *1.* the aisles where `1-pass` is cheaper than the other traversals, *2.* Among the aisles where `1-pass` is cheaper, the one with the lowest no-pass cost, and *3.* Among the aisles where the no-pass cost is lower than c^{1pass}, the one with the highest no-pass cost. Moreover, lines (8-9) require the computation of the two cheapest no-pass costs. These data can all be computed by scanning the list of aisles once, in $\mathcal{O}(A)$ time. Therefore, Step 1 runs in $\mathcal{O}(KA)$ time complexity.

In terms of complexity, the first part of *Step 2* builds the intersection graph, which can be done in $\mathcal{O}(KA)$ operations, and iterates over the horizontal edges of $\mathcal{H}$, performing a constant number of operations in each iteration. Since $\mathcal{H}$ has at most KA edges, the procedure runs in $\mathcal{O}(KA)$ time.

The second part of *Step 2* involves two processes. First, it contracts $\mathcal{H}$ over a subset of its edges, which is achievable in $\mathcal{O}(KA)$ time. Then it solves a MST problem, which would classically require $\mathcal{O}(KAlog(KA))$ time on a generic graph with Prim's algorithm. However, we will show that a problem-specific implementation of Prim's algorithm can run in $\mathcal{O}(KA)$ time complexity. This result relies on the use of a specific data structure to implement the priority queue storing the nodes of $\widetilde{\mathcal{H}}$ that are not part of the tree. First, we observe that the edges of $\widetilde{\mathcal{H}}$ can only take values in $\{1, 2, \ldots, A\}$. Hence, we can build a priority queue as follows. Let Q_a be a linked list of all elements of priority $1 \leq a \leq A$. The collection $(Q_a)_{1 \leq a \leq A}$ can potentially contain all elements, and the creation of an empty instance is performed in $\mathcal{O}(A)$ operations.

We will now show that this data structure allows performing all priority queue operations used by Prim's algorithm in amortized constant time, namely `insert-element`, `decrease-key`, and `find-min`.

- Inserting an element in the queue can naturally be done in $\mathcal{O}(1)$, it is sufficient to insert it in the set corresponding to its priority.

- To decrease the key of an element, we only perform an insertion in the set of its new priority, which is performed in $\mathcal{O}(1)$. Note that, in this case, the element is not removed from its previous set and appears now in (at least) two sets. The obsolete occurrence of the element will be deleted at a later stage in the algorithm.

- The `find-min` operation is the most crucial part of this implementation since it should also account for obsolete elements. The operations proceeds as follows. First, it determines $a^{min} = \min\{1 \leq a \leq A,\, Q_a \neq \emptyset\}$ the set of minimal priority that is non empty. This is executed by iterating through the different values and testing if the corresponding sets are empty. For each element $h \in \widetilde{\mathcal{H}}$, the iteration that inserts h will go up to the priority of h, which is at most the largest value of the edges connected to h. By analyzing $\mathcal{H}$, we observe that the sum of all the edge values is at most $(K+1)A$. Hence finding a^{min} at each iteration will, in total, be performed in $\mathcal{O}(KA)$ operations. Then, once a^{min} is determined, the procedure goes through all the elements of $Q_{a^{min}}$ until it finds one that is not yet in the tree. Indeed, there may be redundant occurrences of already inserted elements since the decrease key operation does not delete them. These elements are deleted when encountered. We will now prove that deleting these elements can be performed in amortized constant time. In the worst case scenario, a node $h \in \widetilde{\mathcal{H}}$ can see its priority decrease once for each of the edges connected to it. Therefore, h will have at most $d(h) - 1$ copies that need to be deleted from Q. We observe that $\mathcal{H}$ possesses at most $\mathcal{O}(KA)$ edges by construction, hence deleting all redundant elements from Q will take at most $\mathcal{O}(KA)$ operations. This concludes the proof that the `find-min` operation is executed in amortized $\mathcal{O}(1)$ time.

Therefore, the procedure performs at most $\mathcal{O}(KA)$ iterations, each consisting of amortized $\mathcal{O}(1)$ operations. Hence, Step 2 runs in $\mathcal{O}(KA)$ time, which concludes the proof. □

4.4 Lower Bound for the PRP

In a rectangular warehouse, we refer to the *Vertical cost (V-cost)* of a path as the sum of the picking aisle costs, while the *Horizontal cost (H-cost)* is the sum of the cross aisle costs. In terms of distance, the V-cost (resp. H-cost) would be the sum of the distances walked along the y-axis (resp. x-axis). In this section, we will prove that the first step of the AFCS heuristic builds a lower bound on the H-cost, from which we can derive a lower bound for the PRP. Additionally, we will prove that the second stage of the heuristic builds the H-cost optimally once the aisle traversals are fixed.

Lower bound on the V-cost. First, let us introduce the following two lemmas before stating the main proposition.

Lemma 1. *A valid route in $\mathcal{G}$ contains an even number of `1-pass` traversals in each block.*

Proof. A valid route starts and ends at the depot, in the front cross aisle. Therefore, each time an aisle is completely traversed in block $k \in \mathcal{K}$, another one must be traversed so that the picker can reach the depot at the end. □

Lemma 2. *For a block $1 \leq k < k^{max}$, the aisle traversals of block k contain at least two* `1-pass`*.*

Proof. Since k^{max} must be visited, and block k is below k^{max}, it is clear that the aisle traversals of k should contain at least one `1-pass`. According to Lemma 1, at least two `1-pass` should be used in k. □

Proposition 1 (Lower bound on the V-cost in one block)**.** *For a block $k \in \mathcal{K}$, let $(c_{ki})_{1 \leq i \leq A}$ be the traversal costs returned by Step 1 of the AFCS heuristic. Then a lower bound on the V-cost in block k can be computed as follows:*

$$LB_k^V = \sum_{1 \leq i \leq A} c_{ki}$$

Proof. We will first deal with the special case of the last block $k = k^{max}$. There exist two possibilities: either at least one aisle is traversed by `1-pass`, or none of the aisles are traversed by `1-pass`. The first possibility will be treated as the general case. If no aisle is traversed by `1-pass`, then cross aisle $k^{max} + 1$ will not be reached, meaning that `top` and `gap` traversals are unavailable. In this case, all aisles are traversed by `bottom`. In Algorithm 1, this case is represented in lines (8-9), and since the inequality in line (8) holds, the traversals will not be modified by the while loop, hence $LB_{k^{max}}^V$ is a lower bound on the cost in the block. In the general case, the algorithm starts by setting each aisle to its minimum no-pass traversal (lines 3-4). Therefore, at this point, k is traversed by the best traversals in the case of no `1-pass`. According to Lemma 2, k must be traversed by at least two `1-pass`, which is done in Algorithm 1, lines (6-7). Since the two aisles set to `1-pass` are those with the highest no-pass costs, k is traversed by the best traversals in the case of two `1-pass`. According to Lemma 1, there is an even number of `1-pass` in each aisle. Since Algorithm 1 (lines 11-14) transforms aisles to `1-pass` two-by-two, each time selecting the pair with the highest no-pass costs, ensuring that the traversals of block k remain, at each iteration, the best traversals for a given number of `1-pass`. The while loop continues until it no longer improves the solution, stopping at the number of `1-pass` that minimizes the overall aisle traversal costs within the block. Hence, Algorithm 1 builds the traversals of block k that minimize the V-cost in the block, while ensuring they can lead to a feasible route. Therefore, LB_k^V is a valid bound. □

Lower bound on the H-cost. In the following, we denote a^{max} the farthest aisle visited, i.e., $\mathcal{V} \cap \mathcal{L}^{ka^{max}} \neq \emptyset$ and $\mathcal{V} \cap \mathcal{L}^{ka} = \emptyset$ for all $1 \leq k \leq K$ and $a^{max} < a \leq A$. Proposition 2 holds.

Proposition 2 (Lower bound on the H-cost)**.** *The following bound is valid:*

$$LB^H = 2a^{max} D^{aisle}$$

Proof. This result is straightforward since the picker travels a H-cost of LB^H to pick an item in a^{max} and return. □

The bound introduced in Proposition 2 represents the cost of a round trip to the farthest aisle. We observe that this bound can be considered as tight for the single block case, but might be of poor quality in the worst case. It is indeed possible to create a PRP instance where LB^H is arbitrary far from the actual H-cost of an optimal route. However, it seems challenging to find a better bound when decoupling vertical and horizontal costs.

Lower bound for the PRP. In the following we introduce Theorem 2, the main result of this section.

Theorem 2 (Lower bound for the PRP)**.** *The cost of a solution of the PRP on an instance defined over the locations in $\mathcal{V}$ is no smaller than:*

$$LB = LB^H + \sum_{k \in \mathcal{K}} LB_k^V + 2D^{depot}$$

Proof. This result directly follows from Propositions 1 and 2. The first term is a lower bound on the H-cost, and the second one is the sum of lower bounds on the V-cost of each block. The last term corresponds to the visit of the depot. □

Corollary 1. *The first step of the AFCS heuristic is optimal at minimizing vertical cost while ensuring feasibility, and the second step of the AFCS minimizes optimally the horizontal cost when aisle traversals are already determined.*

Proof. The construction of Step 1 provides a lower bound on the V-cost according to Proposition 1. Since it also leads to a feasible solution by performing Step 2, Step 1 is optimal when minimizing the V-cost.

The first operation of Step 2, i.e., the construction of the `1-cross` traversals, is the only way to ensure that each intersection has an even degree. Indeed, in each cross aisle the only intersections of odd degree are the ones corresponding to `1-pass`. Hence, the only way to restore the parity of their degrees is to connect to `1-pass` intersections with `1-cross` traversals. Since they are all located on a line, the only way to connect two intersections of odd degree is to connect the two successive ones, which is the operation in Step 2. Then, the construction of the `2-cross` traversal can be modeled as an MST problem, which we solve to optimality. Therefore, Step 2 optimally minimizes the H-cost when aisle traversals are fixed. □

4.5 Approximation ratio for the AFCS heuristic

According to Corollary 1, each step of the AFCS is optimal. While it is easy to show that the AFCS as a whole is not optimal, the natural question that arises is how bad can be, in the worst case, the quality of the solution provided by the AFCS heuristic. Theorem 3 answers this interrogation by providing an approximation ratio for the algorithm.

Theorem 3 (Approximation ratio of the AFCS heuristic)**.** *Let us consider an instance of the PRP induced by a non empty set $\mathcal{V} \subset \mathcal{L}^0$, such that $v^0 \in \mathcal{V}$ and let z^* be the cost of its optimal solution. The cost of the solution returned by the AFCS heuristic is no larger than:*

$$\Big(1 + \frac{KAD^{aisle}}{Kc^{1pass} + AD^{aisle}}\Big)z^*$$

With $c^{1pass} = (L-1)D^{loc} + 2D^{first}$ representing the cost for a full aisle traversal.

Proof. Let z be the value of the solution with the AFCS heuristic, the approximation ratio can be computed as:

$$\frac{z}{z^*} \leq \frac{z}{LB} \leq \frac{LB + \Delta}{LB} \leq 1 + \frac{\Delta}{LB} \tag{1}$$

Next, we distinguish between two cases:

- If the last visited block contains at least two `1-pass` traversals in the AFCS solution, then we have:

  $$LB = \sum_{1 \leq k \leq k^{max}} LB_k^V + 2a^{max}D^{aisle} + 2D^{depot}$$

  Since we have $2D^{depot} > 0$ and each block is traversed by at least two `1-pass` traversal so that $LB_k^V \geq 2c^{1pass}$ for all $1 \leq k \leq k^{max}$, we can reformulate the equation:

  $$LB \geq 2k^{max}c^{1pass} + 2a^{max}D^{aisle} \tag{2}$$

  Furthermore, we observe that, by construction of the AFCS solution, $\Delta \leq 2k^{max}a^{max}D^{aisle}$. With these results, Equation (1) becomes:

  $$\frac{z}{z^*} \leq 1 + \frac{k^{max}a^{max}D^{aisle}}{k^{max}c^{1pass} + a^{max}D^{aisle}} \tag{3}$$

  By multiplying each side of the fraction by $\frac{KA}{a^{max}k^{max}}$, which is valid since $\mathcal{V}$ is supposed to be non empty, so that $k^{max} > 0$ and $a^{max} > 0$, and we get:

  $$\frac{z}{z^*} \leq 1 + \frac{KAD^{aisle}}{\frac{KAc^{1pass}}{a^{max}} + \frac{KAD^{aisle}}{k^{max}}}$$

  Since $a^{max} \leq A$ and $k^{max} \leq K$, we can simplify the bottom terms by:

  $$\frac{z}{z^*} \leq 1 + \frac{KAD^{aisle}}{Kc^{1pass} + AD^{aisle}}$$

- If the last visited block is only visited by `bottom` traversals in the AFCS solution, the proof is very similar to the first case, with minor differences considering the last block. First, when looking at LB, the last block is not crossed entirely, so that the term k^{max} is replaced by $k^{max} - 1$ in Equation (2). Then, we observe that the

last block is visited by `bottom` only, so that there are only k^{max} cross aisles visited ($k^{max}+1$ in the general case), so the difference between the LB and the solution cost is at most $\Delta \leq 2(k^{max}-1)a^{max}D^{aisle}$. In this case, the terms k^{max} in Equation (3) are replaced by $k^{max}-1$. At this stage we need to investigate two cases. If $k^{max} \geq 2$, we can directly divide by $k^{max}-1$ and the rest of the proof is similar to the first case. If $k^{max} = 1$, we cannot divide by $k^{max}-1$, but we observe that in Equation (3), with $(k^{max}-1)$ instead of k^{max}, the fraction is equals to 0 and the result remains valid.

□

Observation. We note that this ratio only depends on the geometry of the considered warehouse layout, and not on the locations to visit. To illustrate with some examples, let us consider the JOBPRP benchmark sets where all instances in one set use the same layout, the ratio is equal to 1.52 for the instances of Henn and Wäscher (2012) and Žulj et al. (2018), and 1.71 for the instances of Muter and Öncan (2015). For the other benchmark sets, the geometry of the layout depends on the instance, and the ratio as well. For the SLAPRP instances of Silva et al. (2020), the ratio is between 1.09 and 1.67. For the PRP instances of Theys et al. (2010), the ratio is between 1.27 and 7.31, worsening as the number of blocks increases.

Corollary 2 (Approximation ratio for single-block warehouses)**.** *For a single-block warehouse, the cost of the solution returned by the AFCS heuristic is no larger than twice the optimal objective value.*

Proof. This result is directly implied by replacing the parameters with their respective values in the expression of the approximation ratio. □

5 Move evaluation and neighborhood exploration

In this section, we propose an efficient procedure to explore neighborhoods for OP problems where the objective is to minimize a cumulative function of the aisle costs. First, let us introduce some notations.

Consider an OP problem whose set of feasible solutions is noted $\mathcal{S}$, and $\mathcal{N} : s \in \mathcal{S} \rightarrow \mathcal{N}(s) \subset \mathcal{S}$ a neighborhood function for the problem. For a solution $s \in \mathcal{S}$, the objective is to efficiently explore $\mathcal{N}(s)$ to find the least-cost neighbor of s. The transformation of s into one of its neighbors $s' \in \mathcal{N}(s)$ is referred to as a *move* and noted $s \rightarrow s'$. We denote $\mathcal{R}$ the set of routes of s. Since a route might be empty, we suppose without loss of generality that $\mathcal{R}(s)$ is of constant size for all solutions. The move $s \rightarrow s'$ modifies a subset $\mathcal{R}^{s \rightarrow s'} \subset \mathcal{R}$ of routes in the solution. For each modified route $r \in \mathcal{R}^{s \rightarrow s'}$, we say that an aisle $a_{ki} \in \mathcal{A}$ is modified by $s \rightarrow s'$ if, and only if, the visited locations have changed. We emphasize that an aisle whose visited locations remain unchanged is not considered modified, even if the traversal of the aisle is modified, as this information

cannot be known a priori. Finally, we note $\mathcal{A}^{s \to s'}(r)$ the set of modified aisles by the move $s \to s'$ in route $r \in \mathcal{R}$.

The remaining of this section is organized as follows. First, we derive routines to underestimate and overestimate moves in Section 5.1, based on the AFCS heuristic. Then, we introduce another move overestimation routine based on the dynamic programming algorithm of Ratliff and Rosenthal (1983); Pansart et al. (2018) in Section 5.2. Finally, these results are embedded into an efficient neighborhood exploration scheme in Section 5.3, where the exploration of the neighborhood is pruned using the underestimation and overestimation routines.

5.1 Move underestimation and overestimation with the AFCS bounds

As presented in Section 4.4, the first step of the AFCS heuristic provides a lower bound on the cost of a PRP solution, which can be used as a cost underestimation routine for each modified route $r \in \mathcal{R}^{s \to s'}$. Similarly, the second step of the AFCS yields a valid PRP solution, providing a cost overestimation for each modified route. Hence, by summing over all modified routes, the routines can be applied to underestimate and overestimate moves.

Implementation details and time complexity. Although the complexity of both steps of the AFCS heuristic run in $\mathcal{O}(|\mathcal{V}| + KA)$ time on a PRP instance, a careful implementation may decrease this complexity in certain cases when applying the AFCS to move evaluation. For a route $r \in \mathcal{R}^{s \to s'}$, we suppose that all parameters computed during the first phase of AFCS are kept in memory. Then, we observe that, for each block, we may update parameters related to the modified aisles $\mathcal{A}^{s \to s'}(r)$ only:

- The aisle traversal costs only change for modified aisles.
- As explained in the complexity proof in Section 4.3, the aisles that are set to 1-pass are only determined by the cost of two specific aisles, i.e. the 1-pass aisle with the lowest no-pass cost, and the no-pass aisle with the highest no-pass cost. If this information is stored, they can be updated by only examining modified aisles.
- The only case where the procedure needs to pass through all aisles is if the block is traversed by bottom only. We observe that, in the case where $\mathcal{N}$ only consists of insertions of locations to visit, as it is the case with the classical LNS neighborhoods, it does not occur that a block that was not visited by bottom-only becomes bottom-only.

Hence, for the LNS neighborhoods, the move underestimation routine may run in $\mathcal{O}(|V| + |\mathcal{A}^{s \to s'}(r)|)$.

5.2 Move overestimation with the critical upper bound

In this section we introduce an efficient heuristic to compute an overestimation of moves, coined the *critical upper bound* heuristic. The provided bound can be interpreted as the increase of the critical path in the extended state space of the DP algorithm proposed by Pansart et al. (2018), due to the alteration in edge costs. For a modified route $r \in \mathcal{R}^{s \to s'}$, we suppose that the routing and cost of r are known. Algorithm 2 outlines a routine that returns an overestimation of the modified route cost. Note that this heuristic may fail to return a solution (and therefore a bound) in some cases. Especially, it cannot be used as a constructive heuristic from scratch as it will always fail to return a solution when no previous routing is provided. Figure 6 illustrates the different cases met by this heuristic.

Algorithm 2: Critical upper bound heuristic

Input: r, costs before modification $c^t_{ki}(r)$ for $t \in \{top, bottom, gap\}$

1: $Delta \leftarrow 0$
2: $r' \leftarrow r$
3: Compute $c^{top}_{ki}(r')$, $c^{bottom}_{ki}(r')$ and $c^{gap}_{ki}(r')$ for $a_{ki} \in \mathcal{A}^{s \to s'}(r)$
4: **for** $a_{ki} \in \mathcal{A}^{s \to s'}(r)$ **do**
5: **if** a_{ki} *is visited by* r **then** // case (a)
6: $t \leftarrow a_{ki}$ traversal in r
7: $\Delta \leftarrow \Delta + c^t_{ki}(r') - c^t_{ki}(r)$
8: **else**
9: $\delta \leftarrow +\infty$
10: **if** $d(p^k_i) > 0$ **then** // case (b)
11: $\delta \leftarrow min(\delta, c^{bottom}_{ki}(r'))$
12: **if** $d(p^{k+1}_i) > 0$ **then** // case (c)
13: $\delta \leftarrow min(\delta, c^{top}_{ki}(r'))$
14: **if** $d(p^k_i)d(p^{k+1}_i) > 0$ **then** // case (d)
15: $\delta \leftarrow min(\delta, c^{gap}_{ki}(r'))$
16: $\Delta \leftarrow \Delta + \delta$
17: **Return** Δ

First, the procedure computes the traversal costs for the modified aisles (line 3). Then, for each modified aisle, it checks if the aisle was previously visited by r (line 5). If that is the case, we use the modified cost of the traversal used in r (line 6 and 7). Otherwise, the procedure checks if it is possible to visit the aisle with bottom (lines 10 and 11), top (lines 12 and 13), or largest gap (lines 14 and 15) traversals. This is done by checking if the two intersections of the start and end of the aisle (i.e., p^k_i and p^{k+1}_i) are visited or not in r, in other words if their degrees are greater than 0. If neither of those points was visited, Δ is set to $+\infty$ (lines 9 and 16), leading the procedure to fail to return a finite bound.

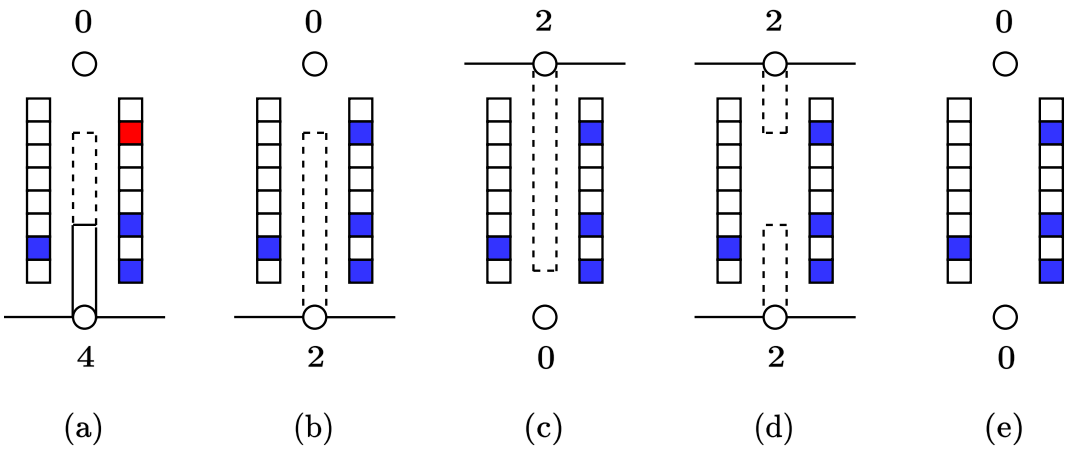


Figure 6: Illustration of the critical heuristic. Full lines represent the route before the move, and dashed lines are the new traversals. In (a) the aisle was already visited, the red stop is added by the move. In (b), (c) (d), (e) the aisle was not visited before. The aisle can be visited by bottom in (b), top in (c), gap in (d). In (e) the heuristic fails to return a bound.

Implementation details and complexity. It is straightforward from Algorithm 2 that the procedure runs in $\mathcal{O}(|\mathcal{V}| + |\mathcal{A}^{s \to s'}(r)|)$ time, as it only iterates through modified aisles. Furthermore, few operations are performed at each iteration, so that the procedure is very efficient in practice.

5.3 Neighborhood exploration

In this section we present an algorithm that efficiently use the results from Sections 5.1 and 5.2 to prune the neighborhood exploration. The underlying idea is to compute the bounds in the hierarchical order, starting from the fastest (critical upper bound) to the more time consuming (AFCS upper bound), and to use bound information to prune dominated moves. More precisely, the procedure works as follows:

1. For each move $s' \in \mathcal{N}(s)$, each modified route $r \in \mathcal{R}_{s \to s'}(s')$ and each modified aisle, compute the bottom, top and largest gap traversal costs. The computation of these parameters is required for move estimation, and for the computation of the exact route cost.

2. For each move $s' \in \mathcal{N}(s)$, compute $\Delta^{UB}_{crit}(s')$, and store the best upper bound found Δ^{UB}_{best}. If the neighborhood only consists in adding visits to the route, as it will be the case for the LNS developed in Section 6, the procedure can directly return a move that does not deteriorate the solution, in other word if $\Delta^{UB}_{crit} = 0$.

3. For each move $s' \in \mathcal{N}(s)$, compute $\Delta^{LB}(s')$.

4. For each move $s' \in \mathcal{N}(s)$, first check if the move is dominated: if $\Delta^{LB}(s') > \Delta^{UB}_{best}$ discard the move, otherwise compute Δ^{UB}_{AFCS} and update Δ^{UB}_{best} accordingly.

5. Among the nondominated moves, select the best one according to the best upper bound among AFCS and critical.

6. Implement the selected move and compute the exact cost of the route using Pansart et al. (2018), to ensure that we always work with PRP-optimal solutions.

6 Large neighborhood search algorithm

In this section we propose a generic LNS algorithm for OP planning problems. Some algorithmic components are specific to the studied problems, namely the JOBPRP and SLAPRP, but most of this section applies to a large class of problems. To remain generic we call *elements* the solution components that are removed from the solution (i.e. the orders for the JOBPRP and the SKUs for the SLAPRP) and *position* the possible insertions for the elements. In other words, a position refers to a storage location for the SLAPRP and to a route for the JOBPRP. An *insertion* is then a pair element/position.

The LNS metaheuristic was introduced by Shaw (1998) for routing problems, where it enjoys a large popularity due to high performances (Ropke and Pisinger 2006). Basically, it consists in sequentially destroy (resp. repair) a solution using removal (resp. insertion) operators in the aim of finding one of good quality. Algorithm 3 outlines the main steps of our LNS approach. First, in line (1), an initial solution is generated (see Section 6.1). The main loop, from lines (2) to (9), iterates until the termination criterion is met. In line (4), a removal operator ω^- is randomly selected from a set Ω^- of operators (see Section 6.2). Similarly, in line (5), an insertion operator ω^+ is randomly selected from a set Ω^+ (see Section 6.3). In line (6), a destruction size ϕ is randomly selected from the interval $[\phi^{min}, \phi^{max}]$. ϕ corresponds to the percentage of elements to be removed from the current solution.

The selected operators are executed on the current solution s' in line (7). First, $\phi\%$ of the elements are removed from s' with the operator ω^-, and placed in an *insertion pool*. Then, the elements within the insertion pool are inserted in s' with the operator ω^+. In line (8-9), the local search procedure is applied to the solution depending on a certain criterion (see Section 6.4). In line (10-11), an acceptance criterion tests whether the new solution should replace the current solution for the next iteration (see Section 6.5). In line (12) a post-processing routine is applied in order to improve the best solution found (see Section 6.6).

6.1 Initial solution

For the SLAPRP, the initial solution is determined by randomly assigning SKUs to the different locations. For the JOBPRP, the number of batches needs to be optimized as well. Therefore, the initial solution is computed using the saving heuristic, and the resulting number of batches is used throughout the run. The savings heuristic starts by assigning each order to a separate route. At each iteration, the algorithm computes the distance *saving* of merging each pair of routes that does not violate the capacity constraint. The

Algorithm 3: LNS algorithm

```
s ← initial_solution()
while termination_criterion() == false do
    s' ← s
    randomly select a removal operator ω⁻ ∈ Ω⁻
    randomly select an insertion operator ω⁺ ∈ Ω⁺
    randomly select a destruction size φ ∈ [φ^min, φ^max]
    s' ← ω⁺(ω⁻(s', φ))
    if local_search_criterion() == true then
        s' ← local_search(s')
    if acceptance_criterion(s') == true then
        s ← s'
post_processing()
return the best solution found
```

two routes inducing the largest saving are merged. The algorithm iterates until there are no more routes that can be merged.

6.2 Removal operators

The LNS uses the following removal operators:

- *Random removal*, where $\phi\%$ elements are randomly selected to be removed from the solution.
- *Related removal* that removes elements that are considered "related" to each other. For two orders $o_i, o_j \in \mathcal{O}$, we note d_i (resp. d_j) the cost of retrieving o_i (resp. o_j), d_{ij} the cost of retrieving both o_i and o_j and $H(o_i, o_j)$ the Hausdorff distance between o_i and o_j, defined as follows. For each location visited by o_i, select the closest location visited by o_j. The largest of these distances is called the Hausdorff distance (Aerts et al. 2021). The relatedness function between o_i and o_j is defined as:

$$R(i,j) = \alpha(1 - \frac{d_i + d_j - d_{ij}}{min(d_i, d_j)}) + \beta\frac{H(o_i, o_j)}{d^{max}} + \gamma\frac{|q_i - q_j|}{max(q_i, q_j)} \tag{4}$$

  Where the first term corresponds to the normalized savings between o_i and o_j as defined in (Žulj et al. 2018), the second term to the normalized Hausdorff distance between o_i and o_j as defined in (Aerts et al. 2021), with d^{max} representing the maximal distance between two locations, and the last term to the difference between the orders consumed capacity. Note that all terms are scaled between 0 and 1. To avoid unnecessary computations, the relatedness matrix is only computed once at the start of the run, since it only depends on characteristics of the instance. For

two SKUs $s_i, s_j \in \Lambda$, the relatedness function is defined as:

$$R(i,j) = \lambda \frac{d_{\mathcal{L}(s_i)\mathcal{L}(s_j)}}{d^{max}} + \mu(1 - \frac{|\mathcal{O}(s_i) \cap \mathcal{O}(s_j)|}{max(|\mathcal{O}(s_i)|, |\mathcal{O}(s_j)|)}) \quad (5)$$

Where the first term $d_{\mathcal{L}(s_i)\mathcal{L}(s_j)}$ corresponds to the distance between the locations of the two SKUs in the current solution, and the second term to the number of orders the two SKUs have in common.

- *Largest aisle reduction removal* where, for each element, the procedure computes the reduction in the total number of visited aisles when removing the element, and removes the elements with the largest scores.

Blink mechanism. The blink mechanism was introduced by Christiaens and Vanden Berghe (2020) to diversify the search process of their LNS insertion operators developed for the Vehicle Routing Problem. Each time an element is selected for removal, there is a probability p^{blink} that the element is skipped, and instead, the next best element is selected.

6.3 Insertion operators

The LNS uses the following insertion operators.

- *Random element, best insertion.* A random element is selected from the insertion pool and inserted in the position minimizing the cost.
- *Best element, best insertion.* At each iteration the operator computes, for each element in the insertion pool, the cost of all feasible insertions, and performs the best overall insertion.
- *Largest element, best insertion.* This operator first sorts the element of the insertion pool by non-increasing order of size (i.e., number of picking lines associated with the element). Then, elements are sequentially inserted at their best positions. If several elements have the same size, one is randomly chosen among them.
- *k-regret insertion.* At each insertion, the operator computes the insertion cost of each element in each feasible insertion. The element that has the largest difference between the cost of its best insertion and the cost of its kth best insertion is inserted at its best position. In this work, we use the 2-regret and 3-regret operators.

Blink mechanism. Similarly to removal methods (see Section 6.2), the blink mechanism is used when selecting an element to insert, in order to diversify the search by adding randomness.

Implementation details. An important feature of the studied problems is the variability in the *size* of elements. We define the size of an element as the number of order lines added in the solution when inserting the element: for the JOBPRP, the size of an order is the number of SKUs, and for the SLAPRP, the size of an SKU is the number of orders picking this SKU. When comparing moves, the cost of inserting an element is always divided by its size, such that the largest elements are not left for the end, where their insertion becomes more challenging (e.g., due to capacity constraints).

6.4 Local search

Local search is employed to strengthen the intensification capabilities of the LNS, using a *Tabu Search (TS)* algorithm. TS is a local search metaheuristic framework first introduced by Glover (1986) that has been successfully applied to a large variety of combinatorial optimization problems. At each iteration, the TS procedure explores the neighborhood $\mathcal{N}(s)$ of the current solution s and selects the best neighbor. In our implementation, we use the *swap* and *relocate* neighborhoods. The swap considers each feasible swap of two elements in the solution and the relocate tries to move each element into every feasible position. To prevent cycling in the search process, recently visited solutions are prohibited and inserted in a tabu list for η iterations. To improve the diversification of the TS procedure, at each iteration a random move is selected in the set of nondominated moves (see Section 5), rather than always picking the most promising one.

Implementation details Local search is first called to improve the initial solution, and then, with a probability p^{local} whenever the solution returned by the removal/insertion operators is feasible. The TS runs for 500 iterations, and the tabu size is defined as the minimum between the size of the instance (number of orders for the JOBPRP, number of SKUs for the SLAPRP) and 100. The tabu list is emptied whenever all moves of the neighborhood are either infeasible or tabu.

6.5 Acceptance criterion

At each LNS iteration, the acceptance criterion determines if the newly generated solution should replace the current solution for the next iterations. First, the cost of the generated solution is computed exactly by utilizing the DP algorithm of Pansart et al. (2018) on each route of the solution. The cost of the solution is then modified according to Equation (6) to account for the elements that could not be inserted in the solution, with B representing the insertion pool of the current solution, E the set of all the elements, and ψ a constant parameter.

$$\text{modified cost} = \text{cost} \times (1 + \psi \frac{|B|}{|E|}) \tag{6}$$

Santini et al. (2018) performed a comparative analysis of different acceptance criteria for the ALNS on CVRP instances. They advocate for the *record-to-record travel*, which

we use in our algorithm. A newly generated solution is accepted to replace the current solution if its cost does not exceed $T\%$ more than the cost of the best-known solution. T decreases linearly over time, from an initial temperature T_0 to 0 when the time limit is reached.

6.6 Post processing

At the end of the run, we solve a set-covering problem to improve the best solution found. The column pool used consists of all the routes generated during the search process. Since the number of routes in the pool can be quite large, we limit the running time of the set-covering program to one minute.

7 Numerical experiments

In this section, we report a summary of the experiments performed on the AFCS heuristic for the PRP and on the LNS algorithm for the JOBPRP and SLAPRP. All implementations are coded in Julia 1.7.2, and CPLEX 12.10 is used to solve the set-covering problems. We set CPLEX to run in single-thread mode, and the MIP emphasis parameter to emphasize "hidden feasibility"over optimality. All other parameters are set to default settings. The experiments are performed in *single-thread computation* on an Intel Xeon E5-2660 v3 CPU clocked at 2.6 GHz, with a memory limit of 8 GB. All computing times are expressed in seconds. The detailed computational results and the source code for the implementation will made publicly available after the publication of the article version of this chapter. In all computational experiments, the gap between a solution z and the best known solution z^{BKS} is computed as $100(z - z^{BKS})/z^{BKS}$. A time limit of 5 minutes is used for all runs, from which one minute is dedicated to the resolution of the set covering model for the JOBPRP. Alternatively, a run terminates after 20000 iterations without improvement of the best solution found.

Concerning the values of the numerical parameters, we use the following values for the JOBPRP runs: $\phi^{min} = 0.02$, $\phi^{max} = 0.2$, $\alpha = 10$, $\beta = 3$, $\gamma = 1$, $p^{blink} = 0.05$ for removal operators and $p^{blink} = 0.02$ for insertion operators. The probability to call the local search is $p^{local} = 0.002$. The initial temperature is $T_0 = 0.01$ and the penalty is $\psi = 10$. All operators are equiprobable. For the SLAPRP runs, we use the following parameter values: $\lambda = 1$, $\mu = 1$, $p^{local} = 5 \times 10^{-4}$, $T_0 = 0.02$. All other parameters use the same values as in the JOBPRP runs.

In this Section, we first describe the instances we use for the experiments in Section 7.1. We report numerical experiments on JOBPRP instances on the benchmark set of Henn and Wäscher (2012) in Section 7.2, the benchmark set of Muter and Öncan (2015) in Section 7.3, the benchmark set of Wahlen and Gschwind (2023) in Section 7.4 and the benchmark set of Žulj et al. (2018) in Section 7.5. Finally, we present results on the SLAPRP instance set of Silva et al. (2020) in Section 7.6.

7.1 Benchmark instances

The following benchmark instances are used for the numerical experiments:

- **The JOBPRP set of Henn and Wäscher (2012).** The set is defined on a single-block warehouse with $A = 10$ aisles and $L = 45$ locations per aisle. The demand consists of $|\mathcal{O}| \in \{20, 30, 40, \ldots, 100\}$ orders, each consisting of 14.3 SKUs on average. The picker capacity is $Q \in \{30, 45, 60, 75\}$ and the storage assignment is made using two different strategies: *class-based* and *uniformly distributed*. The complete set comprises 5760 instances.
- **The JOBPRP set of Muter and Öncan (2015) extended by Wahlen and Gschwind (2023).** The set, noted M&Ö, is defined on a single-block warehouse with $A = 10$ aisles and $L = 10$ locations per aisle. The demand consists of $|\mathcal{O}| \in \{20, 30, 40, \ldots, 100\}$ orders that have an average size of 6.1 SKUs. The picker capacity is originally of $Q \in \{24, 36, 48\}$, but it has been further extended to $Q \in \{60, 72\}$ by Wahlen and Gschwind (2023). The complete set comprises 450 instances.
- **The JOBPRP set of Žulj et al. (2018).** The set is defined on a single-block warehouse with $A = 10$ aisles and $L = 10$ locations per aisle. The demand consists of $|\mathcal{O}|$ orders for a picker capacity Q, with the following pairs $(|\mathcal{O}|, Q)$ used to generate instances: (200, 6), (200, 9), (200, 12), (200, 15), (300, 6), (400, 6), (500, 6) and (600, 6). Order sizes are uniformly distributed in $\{1, \ldots, 5\}$.
- **The JOBPRP set of Wahlen and Gschwind (2023).** The set is defined on a single-block warehouse with $A = 10$ aisles and $L = 10$ locations per aisle. The demand consists of $|\mathcal{O}| \in \{100, 125, 150, 175, 200, 225, 250\}$ orders. Two methods are used to generate the orders: *uniform* where each order has a fixed size of 6, and *random* where the order sizes are randomly drawn from $\{2, \ldots, 10\}$. The picker capacity takes values in $Q \in \{24, 36, 48, 60, 72\}$. Hence, this benchmark set contains the largest instances for the JOBPRP among the ones presented in this section.
- **The large SLAPRP set of Silva et al. (2020).** It contains 486 instances defined on single-block layouts with $A \in \{5, 10, 20\}$ aisles and $L \in \{10, 50\}$ locations per aisle. The demand includes a number $|\mathcal{O}| \in \{10, 30, 50\}$ of orders, each of them consisting of $q_o \in \{5, 20, 50\}$ SKUs. Three different scenarios are used to generate orders, *random* where the SKUs are equiprobable, and two other scenarios of *skewed* demands where the distribution of the demand follows the Pareto principle.

7.2 Benchmark on the set of Henn & Washer

In this section, we test our algorithm on the instance set of Henn and Wäscher (2012). The comparison is made against the results of Wahlen and Gschwind (2023) that propose two column generation-based heuristics: the SC-2 heuristic that only solves the root node

of the search tree and launches a set covering model with all the routes found, and BPC-DF-2 heuristic that is a diving approach exploring the search tree depth first. The authors limited the computing time to two minutes. To the best of our knowledge, Wahlen and Gschwind (2023) proposes the state-of-the-art results on all the JOBPRP benchmark sets, outperforming the existing literature. Hence, we will benchmark against their solution methods on all instance sets. Table 2 presents the results on the instances of Henn and Wäscher (2012), and reports, for each method, the average gap to the best known solution in percentage and the average running time. For our algorithm, the column *gap LNS* reports the gap after termination of the LNS, before launching the set covering model. The other columns reports results for the complete algorithm. The column *iteration* presents the total number of iterations (i.e., both LNS and TS). The column *Inst* reports the number of instances in each line.

Table 2: Results on the benchmark set of Henn and Wäscher (2012)

	Instances		W&G SC-2		W&G BPC-DF-2		Our method			
storage	Q	Inst	gap	time	gap	time	gap	gap LNS	time	iterations
CBD	30	720	0.0	0.1	0.0	0.3	0.0	0.1	39.5	31154
	45	720	0.0	1.3	0.0	11.6	0.2	0.5	84.2	37783
	60	720	0.1	18.5	0.1	45.4	0.4	0.6	106.2	38300
	75	720	0.4	46.4	0.2	64.5	0.5	0.6	125.6	39919
UD	30	720	0.1	0.1	0.1	0.1	0.1	0.2	40.4	31782
	45	720	0.1	1.8	0.1	18.6	0.3	0.6	87.4	38040
	60	720	0.2	32.3	0.1	59.2	0.5	0.6	118.3	41414
	75	720	0.6	56.4	0.3	75.5	0.6	0.6	133.4	40237
Total		5760	0.2	19.6	0.1	34.4	0.3	0.5	91.9	37329

We observe that our algorithm slightly underperforms in terms of solution quality when compared to SC-2 and BPC-BF-2, although solutions deviate to a minor extent ($\approx 0.3\%$) from the best known ones. Furthermore, we observe that the solution times of the column generation-based approaches remain unmatched.

7.3 Benchmark on the set of Muter & Öncan

In this section, we report the experiments conducted on the instance set of Muter and Öncan (2015). This set was originally using the capacity values $Q \in \{24, 36, 48\}$, but it has been extended by Wahlen and Gschwind (2023) to include larger instances of capacity $Q \in \{40, 72\}$. We compare our solution method against the two column generation-based approaches of Wahlen and Gschwind (2023). Note that, to the best of our knowledge, Wahlen and Gschwind (2023) proposes the state-of-the-art results on this benchmark set. Table 3 summarizes the results of these experiments, the column definitions are introduced in Section 7.2.

From Table 3, we observe that the BPC-DF-2 method provides the best solutions for smaller values of the capacities (i.e., 24 and 36). However, the LNS algorithm scales better for the instances with larger capacities, with a similar running time. This result is

Table 3: Results on the benchmark set of Muter and Öncan (2015)

Instances		W&G SC-2		W&G BPC-DF-2		Our method			
Q	Inst	gap	time	gap	time	gap	gap LNS	time	iterations
24	90	0.1	4.9	0.0	31.5	0.3	0.7	82.7	42201
36	90	0.7	53.8	0.2	70.3	0.6	0.6	100.6	38187
48	90	2.1	73.1	1.5	88.3	0.5	0.6	109.1	38283
60	90	8.7	89.7	7.3	94.3	0.5	0.5	111.2	32838
72	90	10.0	102.2	9.8	106.0	0.4	0.4	139.2	32233
Total	450	4.3	64.8	3.8	78.1	0.5	0.6	108.6	36749

unsurprising, as larger capacities generate more challenging pricing problems for column generation-based methods.

7.4 Benchmark on the set of Wahlen & Gschwind

In this section, we evaluate our algorithm on the instances introduced by Wahlen and Gschwind (2023). Table 4 summarizes the obtained results, the columns definitions are provided in Section 7.2. In this benchmark set, two different methods are used to generate the demand: either the order size is *uniform* (fixed at 6) for all orders, or it is *randomly* drawn from $\{2, \ldots, 10\}$.

Table 4: Results on the benchmark set of Wahlen and Gschwind (2023)

Instances			W&G SC-2		W&G BPC-DF-2		Our method			
q_o	Q	Inst	gap	time	gap	time	gap	gap LNS	time	iterations
random	24	100	0.8	111.6	0.8	120.0	1.3	1.8	277.2	43837
	36	100	6.0	120.0	8.9	120.0	1.4	1.5	293.6	40126
	48	100	16.3	120.0	16.3	120.0	1.2	1.2	301.7	37507
	60	100	17.1	120.0	17.1	120.0	0.5	0.5	302.3	30579
	72	100	16.8	120.0	16.8	120.0	0.0	0.0	303.9	23575
uniform	24	100	0.6	79.3	0.8	101.8	0.6	1.2	287.7	21885
	36	100	6.4	119.2	7.8	120.0	1.1	1.1	294.4	20297
	48	100	16.4	120.0	17.6	120.0	0.8	0.8	301.7	16552
	60	100	19.8	120.0	19.8	120.0	0.6	0.7	270.1	14768
	72	100	19.7	120.0	19.7	120.0	0.1	0.2	301.2	12722
Total		700	11.9	115.7	12.4	118.5	0.8	0.9	293.8	27973

From Table 4, we observe that our algorithm scales better with the increase of picker capacity compared to the method developed in Wahlen and Gschwind (2023). This result is unsurprising, as the increase of capacity leads to more challenging subproblems for column generation-based approaches. Apart from the instances with a capacity of 24, the LNS outperforms both methods from Wahlen and Gschwind (2023), although with a longer running time.

7.5 Benchmark on the set of Zulj, Kramer and Schneider

In this section, we test our algorithm on the instance set introduced by Žulj et al. (2018). The authors propose a hybrid LNS and TS metaheuristic that share certain algorithmic components with our implementation. However, they only perform experiments with the S-shape and largest gap routing, representing a more constraint version of the problem. We compare our method with the two column generation-based algorithms of Wahlen and Gschwind (2023) that solve the routing optimally. Note that, in their numerical experiments, the solution methods of Wahlen and Gschwind (2023) clearly outperform the one from Žulj et al. (2018) when S-shape and largest gap routing are used. To the best of our knowledge, Wahlen and Gschwind (2023) presents the state-of-the-art results on this benchmark set. Table 5 presents the results of the experiments, with the columns definitions introduced in Section 7.2.

Table 5: Results on the benchmark set of Žulj et al. (2018)

	Instances		W&G SC-2		W&G BPC-DF-2		Our method			
Q	$\lvert\mathcal{O}\rvert$	Inst	gap	time	gap	time	gap	gap LNS	time	iterations
6	200	10	0.7	2.8	0.7	111.3	0.2	1.0	273.2	45468
	300	10	0.5	12.1	0.6	120.0	0.2	1.5	291.3	24292
	400	10	0.4	17.0	0.6	120.0	0.2	1.8	303.1	17122
	500	10	0.3	31.3	0.6	120.0	0.3	2.4	304.1	10942
	600	10	0.3	78.1	1.4	120.0	0.3	2.4	304.2	6608
9	200	10	0.1	33.1	0.9	120.0	0.9	2.0	271.3	39390
12	200	10	0.7	119.3	1.1	120.0	1.1	1.5	298.1	41908
15	200	10	2.0	119.9	1.4	120.0	1.0	1.0	307.1	36545
Total		60	0.6	51.7	0.9	118.9	0.5	1.7	294.1	27784.0

From Table 5, we observe that our method provides slightly improved results compared to the SC-2 algorithm, albeit with longer running times. For this set of instances, the set covering post processing clearly improves our results, passing from an average gap of 1.7% to 0.5%. This result is unsurprising considering the small capacity of the instances, with most of the set consisting of a capacity of six, resulting in a large number of routes in the solutions. This is typically the situation where mathematical programming-based methods are more adapted.

7.6 Benchmark on the SLAPRP instances

In this section, we test our algorithm on the large SLAPRP instance set of Silva et al. (2020). The authors propose a General Variable Neighborhood Search (GVNS) metaheuristic for the problem, and test their algorithm with different routing methods (i.e., return, S-shape midpoint, largest gap and optimal). The optimal routing is computed using the Lin-Kernighan-Helsgaun (LKH) heuristic (Helsgaun 2000) developed to solve the TSP, called from an external library. Since this implementation is slow, they propose an improved version of the algorithm: first, route costs are computed using the midpoint heuristic until convergence of the GVNS, then the GVNS is relaunched using the LKH

for route computation to improve the solution. Table 6 presents the results of three solution methods. Silva et al. (opt) presents results obtained by the GVNS of Silva et al. (2020) with the LKH heuristic, and Silva et al. (opt + mp) the results when combining the LKH with the midpoint policy. Our method presents the algorithm described in the present paper. For each solution method we report the average gap to the best known solution and the average running time in seconds. The algorithms of Silva et al. (2020) are launched five times on each instance, so we report the average values in the results.

Table 6: Results on the benchmark set of Silva et al. (2020)

Instance					Silva et al. (opt.)		Silva et al. (opt. + mp)		Our method		
A	$\|\mathcal{O}\|$	q_o	picks	Inst	gap	time	gap	time	gap	time	iterations
5	10	5	50	18	1.2	7203.0	0.0	3352.0	2.6	84.0	23936
		20	200	18	47.4	7562.0	2.0	4775.0	5.3	294.0	17265
		50	500	18	10.2	7456.0	2.5	6393.0	10.2	300.0	6645
	30	5	150	18	3.2	7212.0	0.9	4460.0	2.3	274.0	20703
		20	600	18	16.9	7485.0	5.1	7173.0	11.5	300.0	5349
		50	1500	18	7.4	7453.0	1.2	7201.0	6.0	300.0	2321
	50	5	250	18	2.8	7216.0	0.6	5047.0	2.7	298.0	13951
		20	1000	18	5.8	7307.0	1.8	7201.0	9.9	300.0	3837
		50	2500	18	5.6	7719.0	0.7	7202.0	5.1	300.0	1905
10	10	5	50	18	1.2	7207.0	0.1	4343.0	2.1	182.0	23701
		20	200	18	96.6	7686.0	1.8	6586.0	5.6	300.0	8082
		50	500	18	53.4	7717.0	3.0	7201.0	12.8	300.0	3088
	30	5	150	18	4.1	7221.0	1.5	5717.0	1.8	300.0	12802
		20	600	18	37.0	7431.0	3.0	7201.0	12.4	300.0	3269
		50	1500	18	15.7	7268.0	1.7	7202.0	5.9	301.0	1521
	50	5	250	18	3.1	7234.0	0.7	7145.0	1.9	300.0	7619
		20	1000	18	18.4	7409.0	9.0	7201.0	21.7	300.0	1776
		50	2500	18	18.7	7201.0	1.0	7204.0	8.8	301.0	1453
20	10	5	50	18	1.5	7215.0	0.0	5919.0	1.4	226.0	17384
		20	200	18	115.6	7994.0	3.3	7201.0	1.2	300.0	4647
		50	500	18	80.7	7212.0	3.1	7201.0	9.6	302.0	1659
	30	5	150	18	4.1	7242.0	2.0	7201.0	1.8	300.0	7738
		20	600	18	40.5	7587.0	2.0	7202.0	11.0	301.0	1741
		50	1500	18	34.2	7207.0	5.7	7205.0	17.4	302.0	1121
	50	5	250	18	4.1	7289.0	1.6	7201.0	3.4	300.0	4457
		20	1000	18	32.4	7664.0	1.8	7203.0	21.6	301.0	1374
		50	2500	18	24.7	7210.0	2.0	7212.0	14.5	305.0	735
Total				486	25.4	7393.0	2.2	6524.0	7.8	284.0	7410

From Table 6, we observe that our method outperforms the GVNS with LKH heuristic, reducing the average gap from 25.4% to 7.8% on average. However, it is slightly less performing than the GVNS with both LKH and midpoint heuristic achieving an average gap of 2.2%. However, our approach is faster, as we limited the run time to 300 seconds, whereas the GVNS of (Silva et al. 2020) runs for 2h on most instances. There is probably room for improvement for our method, as the LNS components have mostly been designed

for the JOBPRP.

8 conclusion

In this paper, we introduced a novel constructive heuristic for the PRP in rectangular multi-block warehouses, coined AFCS, that builds first the aisle traversals and then the cross aisle traversals. The AFCS runs in linear time of the size of the instance, and returns solutions with a performance guarantee. The AFCS provides upper and lower bounds on the picking distance of a route. From these results, we proposed move evaluation routines relying on the bounds returned by the AFCS as surrogate objective functions. Additionally, we introduce another move evaluation heuristic based on the DP algorithm of Pansart et al. (2018). An efficient neighborhood exploration routine is developed to prune dominated parts of the neighborhood exploration based on bounds information. These contributions are integrated in an LNS algorithm that is benchmarked on JOBPRP and SLAPRP instances from the literature, achieving competitive results against state-of-the-art methods. This study comforts results from the literature that problem-specific knowledge, and especially the use of aisle modeling, is a key element to design efficient algorithms for OP problems.

In the future, we expect further research on integrated problems in warehousing logistics, as they improve the OP performances (van Gils et al. 2018). A natural extension of the present work is the joint optimization of storage, batching, and routing decisions. The resulting problem would provide new insights on the interactions between the three decisions, and is industrially relevant to the replenishment of the picking area (Prunet et al. 2024). Another promising direction would be the adaptation of our methodology to other OP problems, such as the integration of order sequencing decisions (D'Haen et al. 2023; Briant et al. 2023) or scattered storage (Weidinger 2018).

References

Aerts, B., Cornelissens, T., and Sörensen, K. (2021). The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem. *Computers & Operations Research*, 129:105168.

Boysen, N., de Koster, R., and Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2):396–411.

Boysen, N. and Stephan, K. (2013). The deterministic product location problem under a pick-by-order policy. *Discrete Applied Mathematics*, 161(18):2862–2875.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L., and Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research*, 285(2):497–512.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L., and Ogier, M. (2023). Lower and upper bounds for the joint batching, routing and sequencing problem. arXiv:2303.17834 [math].

Cheng, C.-Y., Chen, Y.-Y., Chen, T.-L., and Jung-Woon Yoo, J. (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.

Christiaens, J. and Vanden Berghe, G. (2020). Slack Induction by String Removals for Vehicle Routing Problems. *Transportation Science*, 54(2):417–433. Publisher: INFORMS.

Crainic, T. G., Gendreau, M., Soriano, P., and Toulouse, M. (1993). A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research*, 41(4):359–383.

de Koster, M. B. M., Van Der Poort, E., and Wolters, M. (1999). Efficient orderbatching methods in warehouses. *International Journal of Production Research*, 37(7):1479–1504.

de Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.

D'Haen, R., Braekers, K., and Ramaekers, K. (2023). Integrated scheduling of order picking operations under dynamic order arrivals. *International Journal of Production Research*, 61(10):3205–3226.

Elbert, R. M., Franzke, T., Glock, C. H., and Grosse, E. H. (2017). The effects of human behavior on the efficiency of routing policies in order picking: The case of route deviations. *Computers & Industrial Engineering*, 111:537–551.

Frazelle, E. (2016). *World-class warehousing and material handling*. McGraw-Hill Education, New York, second edition edition.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Guo, X., Chen, R., Du, S., and Yu, Y. (2021). Storage assignment for newly arrived items in forward picking areas with limited open locations. *Transportation Research Part E: Logistics and Transportation Review*, 151:102359.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Henn, S. and Wäscher, G. (2012). Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Operational Research*, 222(3):484–494.

Heßler, K. and Irnich, S. (2022). A note on the linearity of Ratliff and Rosenthal's algorithm for optimal picker routing. *Operations Research Letters*, 50(2):155–159.

Kulak, O., Sahin, Y., and 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(1):52–80.

Mantel, R. J., Schuur, P. C., and Heragu, S. S. (2007). Order oriented slotting: a new assignment strategy for warehouses. *European J. of Industrial Engineering*, 1(3):301.

Masae, M., Glock, C. H., and Grosse, E. H. (2020). Order picker routing in warehouses: A systematic literature review. *International Journal of Production Economics*, 224:107564.

Menéndez, B., Pardo, E. G., Alonso-Ayuso, A., Molina, E., and Duarte, A. (2017). Variable Neighborhood Search strategies for the Order Batching Problem. *Computers & Operations Research*, 78:500–512.

Moons, S., Braekers, K., Ramaekers, K., Caris, A., and Arda, Y. (2019). The value of integrating order picking and vehicle routing decisions in a B2C e-commerce environment. *International Journal of Production Research*, 57(20):6405–6423. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207543.2019.1566668.

Muter, I. and Öncan, T. (2015). An exact solution approach for the order batching problem. *IIE Transactions*, 47(7):728–738.

Pansart, L., Catusse, N., and Cambazard, H. (2018). Exact algorithms for the order picking problem. *Computers & Operations Research*, 100:117–127.

Petersen, C. G. and Schmenner, R. W. (1999). An Evaluation of Routing and Volume-based Storage Policies in an Order Picking Operation. *Decision Sciences*, 30(2):481–501. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-5915.1999.tb01619.x.

Prodhon, C. and Prins, C. (2016). Metaheuristics for Vehicle Routing Problems. In Siarry, P., editor, *Metaheuristics*, pages 407–437. Springer International Publishing, Cham.

Prunet, T., Absi, N., and Cattaruzza, D. (2024). The Storage Location Assignment and Picker Routing Problem: A Generic Branch-Cut-and-Price Algorithm. arXiv:2407.13570 [cs].

Prunet, T., Absi, N., and Cattaruzza, D. (2025). A note on the complexity of the picker routing problem in multi-block warehouses and related problems. *Annals of Operations Research*.

Ratliff, H. D. and Rosenthal, A. S. (1983). Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 31(3):507–521.

Roodbergen, K. J. and de Koster, R. (2001a). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, 39(9):1865–1883.

Roodbergen, K. J. and de Koster, R. (2001b). Routing order pickers in a warehouse with a middle aisle. *European Journal of Operational Research*, 133(1):32–43.

Ropke, S. and Pisinger, D. (2006). A unified heuristic for a large class of Vehicle Routing Problems with Backhauls. *European Journal of Operational Research*, 171(3):750–775.

Santini, A., Ropke, S., and Hvattum, L. M. (2018). A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, 24(5):783–815.

Schiffer, M., Boysen, N., Klein, P. S., Laporte, G., and Pavone, M. (2022). Optimal Picking Policies in E-Commerce Warehouses. *Management Science*, page mnsc.2021.4275.

Scholz, A. and Wäscher, G. (2017). Order Batching and Picker Routing in manual order picking systems: the benefits of integrated routing. *Central European Journal of Operations Research*, 25(2):491–520.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, Lecture Notes in Computer Science, pages 417–431, Berlin, Heidelberg. Springer.

Silva, A., Coelho, L. C., Darvish, M., and Renaud, J. (2020). Integrating storage location and order picking problems in warehouse planning. *Transportation Research Part E: Logistics and Transportation Review*, 140:102003.

Theys, C., Bräysy, O., Dullaert, W., and Raa, B. (2010). Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, 200(3):755–763.

Valle, C. A., Beasley, J. E., and da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, 262(3):817–834.

van Gils, T., Caris, A., Ramaekers, K., and 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(3):814–830.

van Gils, T., Ramaekers, K., Caris, A., and de Koster, R. B. M. (2018). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15.

Wahlen, J. and Gschwind, T. (2023). Branch-Price-and-Cut-Based Solution of Order Batching Problems. *Transportation Science*, 57(3):756–777.

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

Won, J. and Olafsson, S. (2005). Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, 43(7):1427–1442. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207540410001733896.

Žulj, I., Kramer, S., and Schneider, M. (2018). A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *European Journal of Operational Research*, 264(2):653–664.